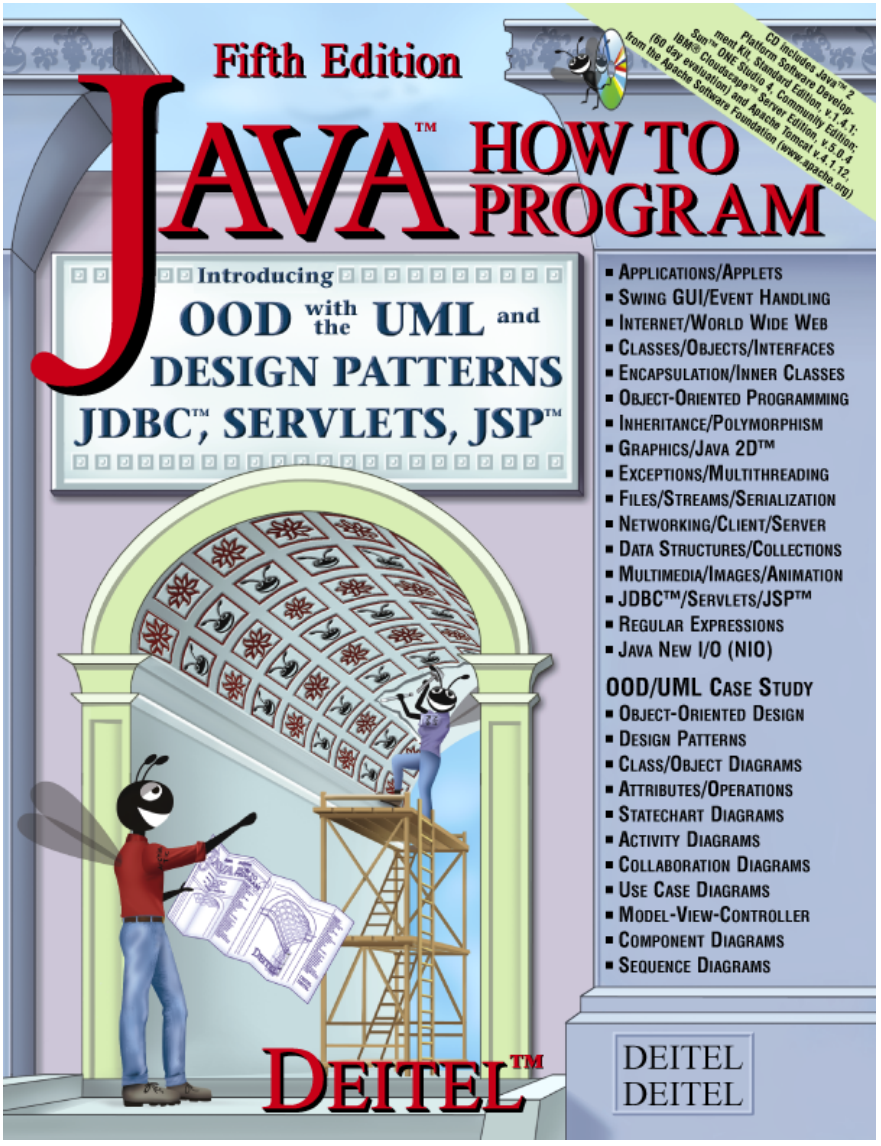
Instructor's Manual

for



Harvey M. Deitel
Paul J. Deitel

Java How to Program, 5/e: Instructor's Manual Contents

1

Introduction to Computers, the Internet and the Web

Objectives

- To understand basic computer science concepts.
- To become familiar with different types of programming languages.
- To introduce a typical Java development environment.
- To understand Java's role in developing distributed client/server applications for the Internet and Web.
- To introduce object-oriented design with the UML and design patterns.

Our life is frittered away by detail … Simplify, simplify.
Henry Thoreau

High thoughts must have high language.
Aristophanes

The chief merit of language is clearness.
Galen

My object all sublime
I shall achieve in time.
W. S. Gilbert

He had a wonderful talent for packing thought close, and rendering it portable.
Thomas Babington Macaulay

Egad, I think the interpreter is the hardest to be understood of the two!
Richard Brinsley Sheridan

SELF-REVIEW EXERCISES

1.1 Fill in the blanks in each of the following statements:

a) The company that popularized personal computing was _____.

ANS: Apple

b) The computer that made personal computing legitimate in business and industry was the _____.

ANS: IBM Personal Computer

c) Computers process data under the control of sets of instructions called _____.

ANS: programs

d) The six key logical units of the computer are the _____, _____, _____, _____, _____ and _____.

ANS: input unit, output unit, memory unit, arithmetic and logic unit, central processing unit, secondary storage unit.

e) The three classes of languages discussed in the chapter are _____, _____ and _____.

ANS: machine languages, assembly languages, high-level languages.

f) The programs that translate high-level language programs into machine language are called _____.

ANS: compilers

g) The _____ allows computer users to locate and view multimedia-based documents on almost any subject over the Internet.

ANS: World Wide Web

h) Java _____ typically are stored on your computer and are designed to execute independent of World Wide Web browsers.

ANS: applications

i) _____ allows an applet or application to perform multiple activities in parallel.

ANS: Multithreading

1.2 Fill in the blanks in each of the following sentences about the Java environment:

a) The _____ command from the Java 2 Software Development Kit executes an applet.

ANS: appletviewer

b) The _____ command from the Java 2 Software Development Kit executes an application

ANS: `java`

c) The _____ command from the Java 2 Software Development Kit compiles a Java program.

ANS: `javac`

d) A(n) _____ file is required to invoke a Java applet.

ANS: HTML

e) A Java program file must end with the _____ file extension.

ANS: `.java`

f) When a Java program is compiled, the file produced by the compiler ends with the _____ file extension.

ANS: `.class`

g) The file produced by the Java compiler contains _____ that are interpreted to execute a Java applet or application.

ANS: bytecodes

1.3 Fill in the blanks in each of the following statements (based on Sections 1.15 and 1.16):

a) Over the past decade, the software-engineering industry has made significant progress in the field of _____—proven architectures for constructing flexible and maintainable object-oriented software.

ANS: design patterns

b) Objects have the property of _____—although objects may know hot to communicate with one another across well-defined interfaces, objects normally are not allowed to know how other objects are implemented.

ANS: information hiding

c) Java programmers concentrate on creating their own user-defined types, called _____.

ANS: classes

d) Classes can also have relationships with other classes. These relationships are called _____.

ANS: associations

e) The process of analyzing and designing a system from an object-oriented point of view is called _____.

ANS: object-oriented analysis and design (OOAD)

EXERCISES

1.4 Categorize each of the following items as either hardware or software:

a) CPU

ANS: hardware

b) Java compiler

ANS: software

c) Java interpreter

ANS: software

d) input unit

ANS: hardware

e) editor

ANS: software

1.5 Fill in the blanks in each of the following statements:

a) The logical unit of the computer that receives information from outside the computer for use by the computer is the _____.

ANS: input unit.

b) The process of instructing the computer to solve specific problems is called _____.

ANS: computer programming.

c) _____ is a type of computer language that uses English-like abbreviations for machine language instructions.

ANS: A high-level language.

d) _____ is a logical unit of the computer that sends information which has already been processed by the computer to various devices so that the information may be used outside the computer.

ANS: The output unit.

e) _____ is a logical unit of the computer that retains information.

ANS: Memory unit and secondary storage unit.

f) _____ is a logical unit of the computer that performs calculations.

ANS: Arithmetic and logical unit (ALU).

g) _____ is a logical unit of the computer that makes logical decisions.

ANS: Arithmetic and logical unit (ALU).

h) _____ languages are most convenient to the programmer for writing programs quickly and easily.

ANS: High-level language.

i) The only language that a computer can directly understand is called that computer's _____.

ANS: machine language.

j) _____ is a logical unit of the computer coordinates the activities of all the other logical units.

ANS: Central processing unit (CPU).

1.6 Distinguish between the terms fatal error and nonfatal error. Why might you prefer to experience a fatal error rather than a nonfatal error?

ANS: Fatal run-time errors cause program to terminate immediately without having successfully performed their jobs. Nonfatal run-time errors allow programs to run to completion, often producing incorrect results. Fatal errors may be preferable, because they are obvious and easy to find.

1.7 Use your Web browser to visit the following Web sites and familiarize yourself with the Java resources available to you on the World Wide Web:

a) `java.sun.com`
b) `java.sun.com/applets`
c) `developer.java.sun.com/developer` [*Note:* You may need to register to access this site. However, registration is free.]
d) `www.javalobby.org`
e) `www.jguru.com`
f) `www.javaworld.com`
g) `www.fawcette.com/javapro`

1.8 Fill in the blanks in each of the following statements (based on Sections 1.15 and 1.16):

a) _____ design patterns describe techniques to instantiate objects (or groups of objects).

ANS: Creational

b) The _____ is now the most widely used graphical representation scheme for modeling object-oriented systems.

ANS: Unified Modeling Language

c) Java classes contain _____ (which implement class behaviors) and _____ (which implement class data).

ANS: methods, fields

d) _____ design patterns allow designers to organize classes and objects into larger structures.

ANS: Structural

e) _____ design patterns assign responsibilities to objects.

ANS: Behavioral

f) In Java, the unit of programming is the _____, from which _____ are eventually instantiated.

ANS: class, objects

1.9 Why is it valuable to study design patterns?

ANS: Using design patterns can substantially reduce the complexity of the design process. Design patters benefit system developers by: helping to construct reliable software using proven architectures and accumulated industry expertise; promoting design reuse in future systems; helping to identify common mistakes and pitfalls that occur when building systems; helping to design systems independently of the language in which

they will ultimately be implemented; establishing a common design vocabulary among developers, and shortening the design phase in a software-development process.

2

Introduction to Java Applications

Objectives

- To be able to write simple Java applications.
- To be able to use input and output statements.
- To become familiar with primitive types.
- To understand basic memory concepts.
- To be able to use arithmetic operators.
- To understand arithmetic-operator precedence.
- To be able to write decision-making statements.
- To be able to use relational and equality operators.

Comment is free, but facts are sacred.
C. P. Scott

The creditor hath a better memory than the debtor.
James Howell

When faced with a decision, I always ask, "What would be the most fun?"
Peggy Walker

He has left his body to science—
and science is contesting the will.
David Frost

Equality, in a social sense, may be divided into that of condition and that of rights.
James Fenimore Cooper

SELF-REVIEW EXERCISES

2.1 Fill in the blanks in each of the following statements:

a) A _____ begins the body of every method, and a(n) _____ ends the body of every method.

ANS: left brace (`{`), right brace (`}`)

b) Every statement ends with a(n) _____.

ANS: semicolon (`;`)

c) The _____ statement (presented in this chapter) is used to make decisions.

ANS: `if`

d) _____ begins an end-of-line comment.

ANS: `//`

e) _____, _____, _____ and _____ are called white space.

ANS: Blank lines, space characters, newline characters and tab characters

f) Class _____ contains methods that display message dialogs and input dialogs.

ANS: `JOptionPane`

g) _____ are reserved for use by Java.

ANS: Keywords

h) Java applications begin execution at method _____.

ANS: `main`

i) Methods _____ and _____ display information in the command window.

ANS: `System.out.print` and `System.out.println`

j) A(n) _____ method is called by using its class name followed by a dot (`.`) and its method name.

ANS: `static`

2.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Comments cause the computer to print the text after the `//` on the screen when the program executes.

ANS: False. Comments do not cause any action to be performed when the program is executed. They are used to document programs and improve their readability.

b) All variables must be given a type when they are declared.

ANS: True

c) Java considers the variables `number` and `NuMbEr` to be identical.

ANS: False. Java is case sensitive, so these variables are distinct.

d) False. The remainder operator can also be used with noninteger operands in Java.

ANS: False. The modulus operator can also be used with noninteger operands in Java.

e) The arithmetic operators `*`, `/`, `%`, `+` and `-` all have the same level of precedence.

ANS: False. The operators `*`, `/` and `%` are on the same level of precedence, and the operators `+` and `-` are on a lower level of precedence.

f) Method `Integer.parseInt` converts an integer to a `String`.

ANS: False. `Integer.parseInt` method converts a `String` to an integer (`int`) value.

2.3 Write Java statements to accomplish each of the following tasks:

a) Declare variables `c`, `thisIsAVariable`, `q76354` and `number` to be of type `int`.

ANS:
```
int c, thisIsAVariable, q76354, number;
```
or
```
int c;
int thisIsAVariable;
int q76354;
int number;
```

b) Display a dialog asking the user to enter an integer and assign the result to `String` variable `value`.

ANS:
```
String value =
   JOptionPane.showInputDialog( "Enter an integer" );
```

c) Convert the `String` in part (b) to an integer, and store the converted value in integer variable `age`.

ANS:
```
int age = Integer.parseInt( value );
```

d) If the variable `number` is not equal to 7, display `"The variable number is not equal to 7"` in a message dialog. Use the version of the message dialog that requires two arguments.

ANS:
```
if ( number != 7 )
   JOptionPane.showMessageDialog( null,
      "The variable number is not equal to 7" );
```

e) Print `"This is a Java program"` on one line in the command window.

ANS:
```
System.out.println( "This is a Java program" );
```

f) Print `"This is a Java program"` on two lines in the command window; the first line should end with `Java`. Use only one statement.

ANS:
```
System.out.println( "This is a Java\nprogram" );
```

2.4 Identify and correct the errors in each of the following statements:

a)
```
if ( c < 7 );
   JOptionPane.showMessageDialog( null, "c is less than 7" );
```

ANS: Error: Semicolon after the right parenthesis of the condition `( c < 7 )` in the `if`.
Correction: Remove the semicolon after the right parenthesis. [*Note*: As a result, the output statement will execute regardless of whether the condition in the `if` is true.]

b)
```
if ( c => 7 )
   JOptionPane.showMessageDialog( null,
      "c is equal to or greater than 7" );
```

ANS: Error: The relational operator => is incorrect.
Correction: Change => to >=.

2.5 Write declarations, statements or comments that accomplish each of the following tasks:

a) State that a program will calculate the product of three integers.
b) Declare the variables `x`, `y`, `z` and `result` to be of type `int`.
c) Declare the variables `xVal`, `yVal` and `zVal` to be of type `String`.
d) Prompt the user to enter the first value, read the value from the user and store it in the variable `xVal`.
e) Prompt the user to enter the second value, read the value from the user and store it in the variable `yVal`.
f) Prompt the user to enter the third value, read the value from the user and store it in the variable `zVal`.
g) Convert `xVal` to an `int`, and store the result in the variable `x`.
h) Convert `yVal` to an `int`, and store the result in the variable `y`.
i) Convert `zVal` to an `int`, and store the result in the variable `z`.
j) Compute the product of the three integers contained in variables `x`, `y` and `z`, and assign the result to the variable `result`.
k) Display a dialog containing the message `"The product is "` followed by the value of the variable `result`.
l) Terminate the program and indicate successful termination.

ANS:

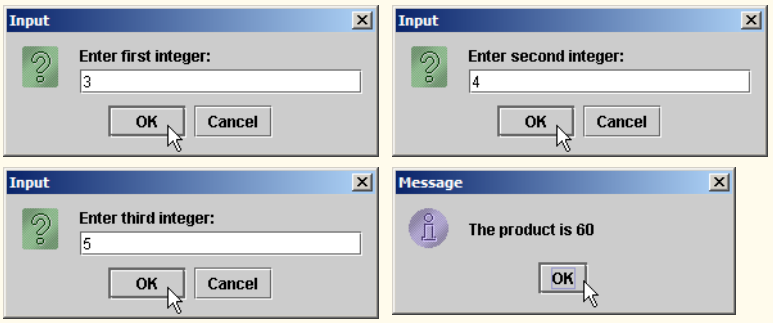
a)
```
// Calculate the product of three integers
```

b)
```
int x, y, z, result;
```
or
```
int x;
```

```
      int y;
      int z;
      int result;
   c) String xVal, yVal, zVal;
      or
      String xVal;
      String yVal;
      String zVal;
   d) xVal = JOptionPane.showInputDialog( "Enter first integer:" );
   e) yVal = JOptionPane.showInputDialog( "Enter second integer:" );
   f) zVal = JOptionPane.showInputDialog( "Enter third integer:" );
   g) x = Integer.parseInt( xVal );
   h) y = Integer.parseInt( yVal );
   i) z = Integer.parseInt( zVal );
   j) result = x * y * z;
   k) JOptionPane.showMessageDialog( null, "The product is " + result );
   l) System.exit( 0 );
```

2.6 Using the statements you wrote in Exercise 2.5, write a complete program that calculates and prints the product of three integers. :

```
// Ex. 2.6: Product.java
// Calculate the product of three integers.

// Java packages
import javax.swing.JOptionPane;

public class Product {

   public static void main( String args[] )
   {
      int x;       // first number
      int y;       // second number
      int z;       // third number
      int result;  // product of numbers

      String xVal;  // first string input by user
      String yVal;  // second string input by user
      String zVal;  // third string input by user

      xVal = JOptionPane.showInputDialog( "Enter first integer:" );
      yVal = JOptionPane.showInputDialog( "Enter second integer:" );
      zVal = JOptionPane.showInputDialog( "Enter third integer:" );

      x = Integer.parseInt( xVal );
      y = Integer.parseInt( yVal );
      z = Integer.parseInt( zVal );

      result = x * y * z;

      JOptionPane.showMessageDialog( null, "The product is " + result );

      System.exit( 0 );

   } // end method main

} // end class Product
```



EXERCISES

2.7 Fill in the blanks in each of the following statements:

a) _____ are used to document a program and improve its readability.

ANS: Comments

b) An input dialog capable of receiving input from the user is displayed with method _____ of class _____.

ANS: `showInputDialog`, `JOptionPane`.

c) A decision can be made in a Java program with an _____.

ANS: `if`

d) Calculations are normally performed by _____ statements.

ANS: assignment

e) A dialog capable of displaying a message to the user is displayed with method _____ of class _____.

ANS: `showMessageDialog`, `JOptionPane`

2.8 Write Java statements that accomplish each of the following tasks:

a) Display the message `"Enter two numbers"`, using class `JOptionPane`.

ANS:
```
JOptionPane.showMessageDialog( null, "Enter two numbers" );
```

b) Assign the product of variables b and c to variable a.

ANS:
```
a = b * c;
```

c) State that a program performs a sample payroll calculation (i.e., use text that helps to document a program).

ANS:
```
// This program performs a simple payroll calculation.
```

2.9 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Java operators are evaluated from left to right.

ANS: False. Some operators (e.g., assignment, =) evaluate from right to left.

b) The following are all valid variable names:`_under_bar_`, `m928134`, `t5`, `j7`, `her_sales$`, `his_$account_total`, `a`, `b$`, `c`, `z`, `z2`.

ANS: True.

c) A valid Java arithmetic expression with no parentheses is evaluated from left to right.

ANS: False. The expression is evaluated according to operator precedence.

d) The following are all invalid variable names: `3g`, `87`, `67h2`, `h22` and `2h`.
ANS: False. Identifier `h22` is a valid variable name.

2.10 Fill in the blanks in each of the following statements:
a) The arithmetic operations that have the same precedence as multiplication are the _____.
ANS: division (`/`) and modulus (`%`).
b) When parentheses in an arithmetic expression are nested, the _____ set of parentheses is evaluated first?
ANS: innermost
c) A location in the computer's memory that may contain different values at various times throughout the execution of a program is called a _____.
ANS: variable

2.11 What displays in the message dialog when each of the given Java statements is performed? Assume that `x = 2` and `y = 3`.
a)
```
JOptionPane.showMessageDialog( null, "x = " + x );
```
ANS: **x** = 2
b)
```
JOptionPane.showMessageDialog( null,
   "The value of x + x is " + ( x + x ) );
```
ANS: **The value of x + x is 4**
c)
```
JOptionPane.showMessageDialog( null, "x =" );
```
ANS: **x** =
d)
```
JOptionPane.showMessageDialog( null,
   ( x + y ) + " = " + ( y + x ) );
```
ANS: 5 = 5

2.12 Which of the following Java statements contain variables whose values are changed or replaced?
a)
```
p = i + j + k + 7;
```
b)
```
JOptionPane.showMessageDialog( null,
   "variables whose values are destroyed" );
```
c)
```
JOptionPane.showMessageDialog( null, "a = 5" );
```
d)
```
stringVal = JOptionPane.showInputDialog( "Enter string:" );
```
ANS: (a) and (d).

2.13 Given that $y = ax^3 + 7$, which of the following are correct Java statements for this equation?
a)
```
y = a * x * x * x + 7;
```
b)
```
y = a * x * x * ( x + 7 );
```
c)
```
y = ( a * x ) * x * ( x + 7 );
```
d)
```
y = ( a * x ) * x * x + 7;
```
e)
```
y = a * ( x * x * x ) + 7;
```
f)
```
y = a * x * ( x * x + 7 );
```
ANS: (a), (d) and (e).

2.14 State the order of evaluation of the operators in each of the following Java statements, and show the value of `x` after each statement is performed:
a)
```
x = 7 + 3 * 6 / 2 - 1;
```
ANS: `*` is first, `/` is second, `+` is third, and `-` is fourth. Value of `x` is 15.
b)
```
x = 2 % 2 + 2 * 2 - 2 / 2;
```
ANS: `%` is first, `*` is second, `/` is third, `+` is fourth, `-` is fifth. Value of `x` is 3.
c)
```
x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );
```

ANS:
```
x = ( 3 * 9 * ( 3 + ( 9 * 3 / ( 3 ) ) ) );
              5   6     4     2   3   1
```
Value of x is 324.

2.15 Write an application that displays the numbers 1 to 4 on the same line, with each pair of adjacent numbers separated by one space. Write the program using the following techniques;

a) using one `System.out` statement.

ANS:

```
1  // Exercise 2.15a Solution: One.java
2  // Prints the numbers 1 through 4, separated by
3  // one space using one System.out statement.
4
5  public class One {
6
7     public static void main( String args[] )
8     {
9        // one System.out statement
10       System.out.println("1 2 3 4");
11
12       System.exit( 0 );
13    }
14
15 }  // end class One
```

```
1 2 3 4
```

b) using four `System.out` statements.

ANS:

```
1  // Exercise 2.15b Solution: Four.java
2  // Prints the numbers 1 through 4, separated by
3  // one space using four system.out statements.
4
5  public class Four {
6
7     public static void main( String args[] )
8     {
9        // print the string without a carriage return
10       System.out.print("1 ");
11       System.out.print("2 ");
12       System.out.print("3 ");
13
14       // notice the println
15       System.out.println("4");
16
17       System.exit( 0 );
18    }
19
20 }  // end class Four
```

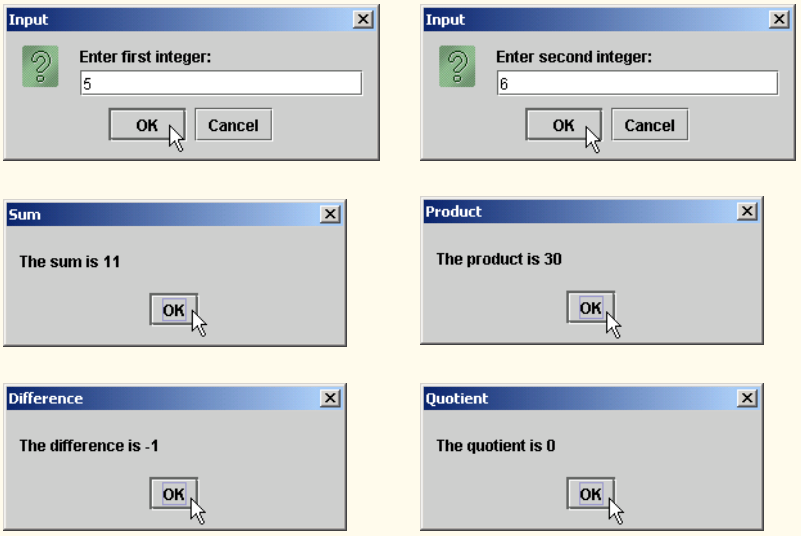
```
1 2 3 4
```

2.16 Write an application that asks the user to enter two numbers, obtains the numbers from the user and prints the sum, product, difference and quotient (division) of the numbers. Use the techniques shown in Fig. 2.9.

ANS:

```
1  // Exercise 2.16 Solution: Calculate.java
2  // Prints the sum, product, difference and quotient of
3  // two numbers.
4
5  import javax.swing.JOptionPane;
6
7  public class Calculate {
8
9     public static void main( String args[] )
10    {
11       String firstNumber,    // first string entered by user
12              secondNumber;   // second string entered by user
13       int number1,           // first number
14           number2;           // second number
15       int sum, product, difference, quotient;
16
17       // read first number from user as a string
18       firstNumber = JOptionPane.showInputDialog( "Enter first integer:" );
19
20       // read second number from user as a string
21       secondNumber = JOptionPane.showInputDialog(
22          "Enter second integer:" );
23
24       // convert numbers from type String to type int
25       number1 = Integer.parseInt( firstNumber );
26       number2 = Integer.parseInt( secondNumber );
27
28       // calculate
29       sum = number1 + number2;
30       product = number1 * number2;
31       difference = number1 - number2;
32       quotient = number1 / number2;
33
34       // display results
35       JOptionPane.showMessageDialog( null, "The sum is " + sum, "Sum",
36          JOptionPane.PLAIN_MESSAGE );
37       JOptionPane.showMessageDialog( null, "The product is " + product,
38          "Product", JOptionPane.PLAIN_MESSAGE );
39       JOptionPane.showMessageDialog( null, "The difference is " +
40          difference, "Difference", JOptionPane.PLAIN_MESSAGE );
41       JOptionPane.showMessageDialog( null, "The quotient is " + quotient,
42          "Quotient", JOptionPane.PLAIN_MESSAGE );
43
44       System.exit( 0 );
45    }
```

```
46
47 } // end class Calculate
```

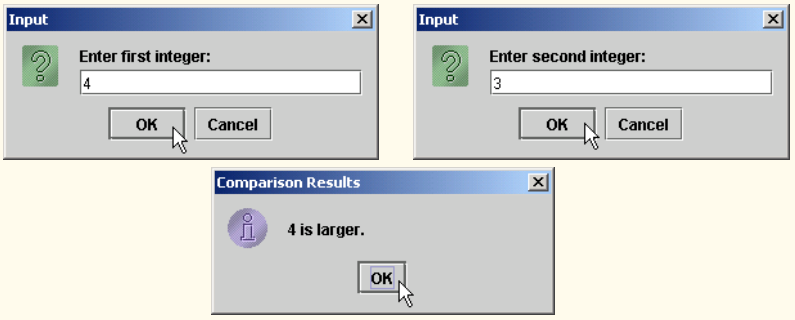


2.17 Write an application that asks the user to enter two integers, obtains the numbers from the user and displays the larger number followed by the words "`is larger`" in an information message dialog. If the numbers are equal, print the message "`These numbers are equal.`" Use the techniques shown in Fig. 2.20.

ANS:

```
1  // Exercise 2.17 Solution: Larger.java
2  // Program that determines the larger of two numbers.
3
4  import javax.swing.JOptionPane;
5
6  public class Larger {
7
8     // main method begins execution of Java application
9     public static void main( String args[] )
10    {
11       String firstNumber;
12       String secondNumber;   // second string entered by user
13       String result;
14       int number1;
15       int number2;
16
17       // read first number from user as a string
18       firstNumber = JOptionPane.showInputDialog( "Enter first integer:" );
19
20       // read second number from user as a string
21       secondNumber =
22          JOptionPane.showInputDialog( "Enter second integer:" );
```

```
23
24          // convert numbers from type String to type int
25          number1 = Integer.parseInt( firstNumber );
26          number2 = Integer.parseInt( secondNumber );
27
28          // initialize result to empty String
29          result = "";
30
31          if ( number1 > number2 )
32             result = number1 + " is larger.";
33
34          if ( number1 < number2 )
35             result = number2 + " is larger.";
36
37          if ( number1 == number2 )
38             result = "These numbers are equal.";
39
40          // Display results
41          JOptionPane.showMessageDialog( null, result, "Comparison Results",
42             JOptionPane.INFORMATION_MESSAGE );
43
44          System.exit( 0 ); // terminate application
45
46       } // end method main
47
48    } // end class Larger
```



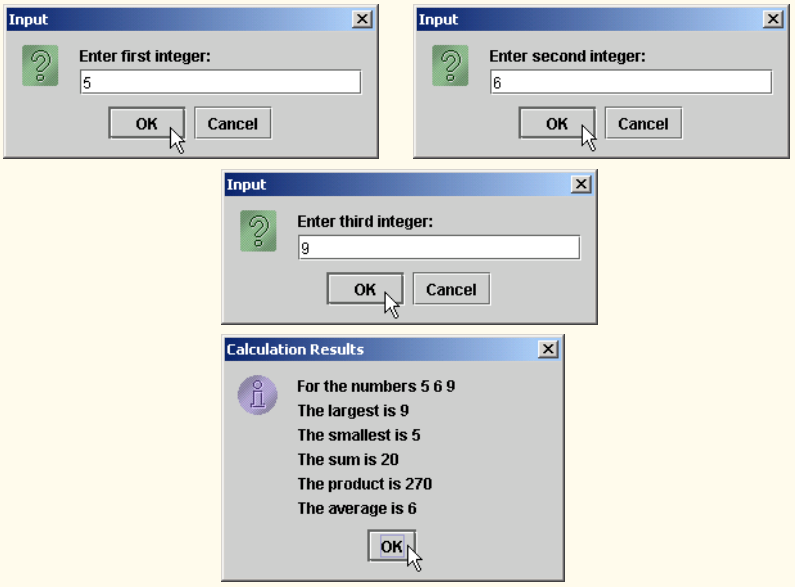
2.18 Write an application that inputs three integers from the user and displays the sum, average, product, smallest and largest of the numbers in an information message dialog. Use the GUI techniques shown in Fig. 2.20. [*Note*: The calculation of the average in this exercise should result in an integer representation of the average. So, if the sum of the values is 7, the average should be 2, not 2.3333….]

ANS:

```
1  // Exercise 2.18 Solution: Calculate2.java
2  // Make simple calculations on three integers.
3
4  import javax.swing.JOptionPane;
5
```

```
6  public class Calculate2 {
7
8     public static void main( String args[] )
9     {
10       String firstNumber,    // first string entered by user
11              secondNumber,   // second string entered by user
12              thirdNumber;    // third string entered by user
13       String result;
14       int number1,           // first number
15           number2,           // second number
16           number3;           // third number
17
18       int sum, largest, smallest, product, average;
19
20       // read first number from user as a string
21       firstNumber = JOptionPane.showInputDialog( "Enter first integer:" );
22
23       // read second number from user as a string
24       secondNumber = JOptionPane.showInputDialog(
25          "Enter second integer:" );
26
27       // read third number from user as a string
28       thirdNumber = JOptionPane.showInputDialog( "Enter third integer:" );
29
30       // convert numbers from type String to type int
31       number1 = Integer.parseInt( firstNumber );
32       number2 = Integer.parseInt( secondNumber );
33       number3 = Integer.parseInt( thirdNumber );
34
35       // initialize largest and smallest
36       largest = number1;
37       smallest = number2;
38
39       // determine correct values
40       if ( number2 >= number1 ) {
41          largest = number2;
42          smallest = number1;
43       }
44
45       if ( number3 > largest )
46          largest = number3;
47
48       if ( number3 < smallest )
49          smallest = number3;
50
51       // perform calculations
52       sum = number1 + number2 + number3;
53       product = number1 * number2 * number3;
54       average = sum / 3;
55
56       // print results
57       result = "For the numbers " + number1 + " " + number2 + " " +
58          number3 + "\n" + "The largest is " + largest + "\n" +
59          "The smallest is " + smallest + "\n" + "The sum is " + sum +
```

```
60             "\n" + "The product is " + product + "\n" +
61             "The average is " + average + "\n";
62
63          // display results
64          JOptionPane.showMessageDialog( null, result, "Calculation Results",
65             JOptionPane.INFORMATION_MESSAGE );
66
67          System.exit( 0 );
68       }
69
70    }  // end class Calculate2
```



2.19 Write an application that inputs from the user the radius of a circle as an integer and prints the circle's diameter, circumference and area. Use the value 3.14159 for π. Use the GUI techniques shown in Fig. 2.9. [*Note*: You may also use the predefined constant `Math.PI` for the value of π. This constant is more precise than the value 3.14159. Class `Math` is defined in the `java.lang` package, so you do not need to `import` it.] Use the following formulas (*r* is the radius):

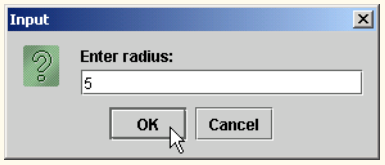
diameter = $2r$
circumference = $2\pi r$
area = πr^2

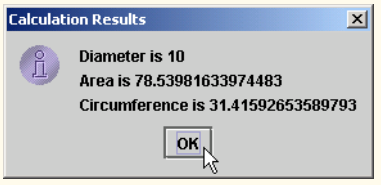
Do not store the results of each calculation in a variable. Rather, add the result of each directly to a string that will be used to display the results.

ANS:

```
1  // Exercise 2.19 Solution: Circle.java
2  // Program that calculates area, circumference
3  // and diameter for a circle.
```

```
4
5  import javax.swing.JOptionPane;
6
7  public class Circle {
8
9     // main method begins execution of Java application
10    public static void main( String args[] )
11    {
12       String input;    // string entered by user
13       String result;   // output display string
14       int radius;      // radius of circle
15
16       // read from user as a string
17       input = JOptionPane.showInputDialog( "Enter radius:" );
18
19       // convert number from type String to type int
20       radius = Integer.parseInt( input );
21
22       result = "Diameter is " + ( 2 * radius ) +
23          "\nArea is " + ( Math.PI * radius * radius ) +
24          "\nCircumference is " + ( 2 * Math.PI * radius );
25
26       // Display results
27       JOptionPane.showMessageDialog( null, result, "Calculation Results",
28          JOptionPane.INFORMATION_MESSAGE );
29
30       System.exit( 0 );  // terminate application
31
32    }  // end method main
33
34 }  // end class Circle
```





2.20 Write an application that displays in the command window a box, an oval, an arrow and a diamond using asterisks (*), as follows:

```
*********          ***             *               *
*       *        *     *          ***             * *
*       *       *       *        *****           *   *
*       *       *       *          *            *     *
*       *       *       *          *           *       *
*       *       *       *          *            *     *
*       *       *       *          *             *   *
*       *        *     *           *              * *
*********          ***             *               *
```

ANS:

```
1  // Exercise 2.20 Solution: Shapes.java
2  // Program draws four shapes to the command window.
3
4  public class Shapes {
5
6     public static void main( String args[] )
7     {
8        System.out.println( "*********     ***       *        *    " );
9        System.out.println( "*       *   *     *    ***      * *   " );
10       System.out.println( "*       *  *       *  *****    *   *  " );
11       System.out.println( "*       *  *       *   *     *      * " );
12       System.out.println( "*       *  *       *   *     *     *" );
13       System.out.println( "*       *  *       *   *     *      * " );
14       System.out.println( "*       *  *       *   *     *    *  " );
15       System.out.println( "*       *   *     *    *     *     * * " );
16       System.out.println( "*********     ***       *        *    " );
17    }
18
19 }  // end class Shapes
```

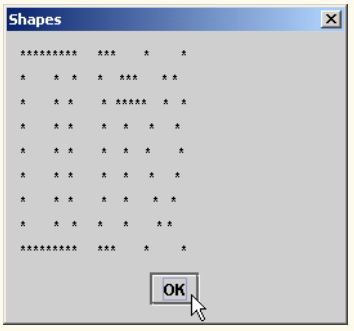
```
*********     ***       *        *
*       *   *     *    ***      * *
*       *  *       *  *****    *   *
*       *  *       *   *     *     *
*       *  *       *   *     *      *
*       *  *       *   *     *     *
*       *  *       *   *      *   *
*       *   *     *    *       * *
*********     ***       *        *
```

2.21 Modify the program you created in Exercise 2.20 to display the shapes in a `JOptionPane.PLAIN_MESSAGE` dialog. Does the program display the shapes exactly as in Exercise 2.20?

ANS:

```
1  // Exercise 2.21 Solution: Shapes2.java
2  // Program draws four shapes in a Plain Message Dialog
3
4  // Java extension pachages
5  import javax.swing.JOptionPane;
6
7  public class Shapes2 {
8
9     public static void main( String args[] )
10    {
11       // create a multiline String to draw the shapes
12       String shapeString =
13          "*********     ***       *        *    " + "\n" +
14          "*       *   *     *    ***      * *   " + "\n" +
15          "*       *  *       *  *****    *   *  " + "\n" +
```

```
16             "*        *  *        *    *      *     * " + "\n" +
17             "*        *  *        *    *    *        *" + "\n" +
18             "*        *  *        *    *      *     * " + "\n" +
19             "*        *  *        *    *       *   *  " + "\n" +
20             "*        *   *      *     *        * *    " + "\n" +
21             "**********     ***        *         *     " + "\n";
22
23          // display a dialog with the shapes
24          JOptionPane.showMessageDialog( null, shapeString,
25             "Shapes", JOptionPane.PLAIN_MESSAGE );
26
27          System.exit( 0 );
28       }
29
30    }  // end class Shapes2
```



2.22 What does the following code print?

```
System.out.println( "*\n**\n***\n****\n*****" );
```

ANS:

```
*
**
***
****
*****
```

2.23 What does the following code print?

```
System.out.println( "*" );
System.out.println( "***" );
System.out.println( "*****" );
System.out.println( "****" );
System.out.println( "**" );
```

ANS:

```
*
***
*****
****
**
```

2.24 What does the following code print?

```
System.out.print( "*" );
System.out.print( "***" );
System.out.print( "*****" );
System.out.print( "****" );
System.out.println( "**" );
```

ANS:

```
***************
```

2.25 What does the following code print?

```
System.out.print( "*" );
System.out.println( "***" );
System.out.println( "*****" );
System.out.print( "****" );
System.out.println( "**" );
```

ANS:

```
****
*****
******
```

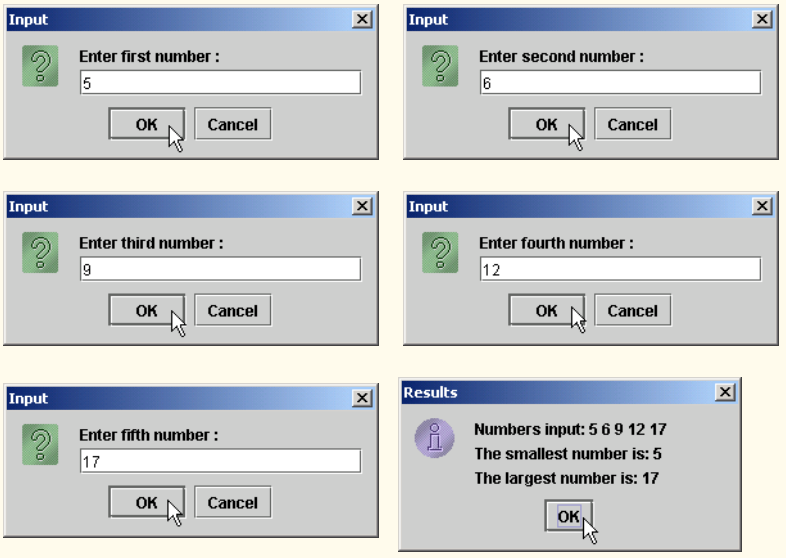
2.26 Write an application that reads five integers and determines and prints the largest and the smallest integers in the group. Use only the programming techniques you learned in this chapter.

ANS:

```
1  // Exercise 2.26 Solution: LargeSmall.java
2  // Program calculates the largest and smallest
3  // of a group of integers entered one at a time.
4
5  import javax.swing.JOptionPane;
6
7  public class LargeSmall {
8
```

```
 9     public static void main( String args[] )
10     {
11        String input;  // string entered by user
12
13        // numbers to be entered
14        int firstNumber, secondNumber, thirdNumber,
15           fourthNumber, fifthNumber;
16
17        // largest and smallest
18        int largest, smallest;
19
20        // read from user as a string,
21        // and convert type String to type int
22        input = JOptionPane.showInputDialog( "Enter first number :" );
23        firstNumber = Integer.parseInt( input );
24
25        // initialize the smallest and largest
26        smallest = firstNumber;
27        largest = firstNumber;
28
29        // read the next four numbers, and check to see if
30        // each is the largest or smallest
31        input = JOptionPane.showInputDialog( "Enter second number :" );
32        secondNumber = Integer.parseInt( input );
33
34        if (smallest > secondNumber)
35           smallest = secondNumber;
36
37        if (largest < secondNumber)
38           largest = secondNumber;
39
40        input = JOptionPane.showInputDialog( "Enter third number :" );
41        thirdNumber = Integer.parseInt( input );
42        if (smallest > thirdNumber)
43           smallest = thirdNumber;
44
45        if (largest < thirdNumber)
46           largest = thirdNumber;
47
48        input = JOptionPane.showInputDialog( "Enter fourth number :" );
49        fourthNumber = Integer.parseInt( input );
50        if (smallest > fourthNumber)
51           smallest = fourthNumber;
52
53        if (largest < fourthNumber)
54           largest = fourthNumber;
55
56        input = JOptionPane.showInputDialog( "Enter fifth number :" );
57        fifthNumber = Integer.parseInt( input );
58        if (smallest > fifthNumber)
59           smallest = fifthNumber;
60
61        if (largest < fifthNumber)
62           largest = fifthNumber;
```

```
63
64          // build the result string:
65          // echo the input, then show the largest and smallest
66          String result;   // results string to build and display
67          result = "Numbers input: " + firstNumber + " " + secondNumber +
68             " " + thirdNumber + " " + fourthNumber + " " + fifthNumber +
69             "\n" + "The smallest number is: " + smallest + "\n"
70             + "The largest number is: " + largest + "\n";
71
72          // show the results
73          JOptionPane.showMessageDialog(null, result,
74             "Results", JOptionPane.INFORMATION_MESSAGE );
75
76          System.exit( 0 );
77
78       } // end method main
79
80    }  // end class LargeSmall
```

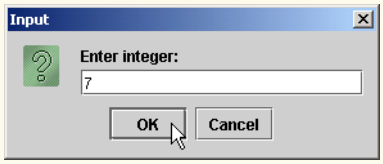


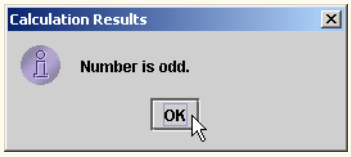
2.27 Write an application that reads an integer and determines and prints whether it is odd or even. [*Hint*: Use the remainder operator. An even number is a multiple of 2. Any multiple of 2 leaves a remainder of 0 when divided by 2.]

ANS:

```
1  // Exercise 2.27 Solution: OddEven.java
2  // Program that determines if a number is odd or even.
3
4  import javax.swing.JOptionPane;
5
6  public class OddEven {
```

```
7
8     // main method begins execution of Java application
9     public static void main( String args[] )
10    {
11       String input;    // string entered by user
12       String result;   // output display string
13       int number;      // number
14
15       // read from user as a string
16       input = JOptionPane.showInputDialog( "Enter integer:" );
17
18       // convert number from type String to type int
19       number = Integer.parseInt( input );
20
21       // initialize result to empty String
22       result = "";
23
24       if ( number % 2 == 0 )
25          result = "Number is even.";
26
27       if ( number % 2 != 0 )
28          result = "Number is odd.";
29
30       // Display results
31       JOptionPane.showMessageDialog( null, result, "Calculation Results",
32          JOptionPane.INFORMATION_MESSAGE );
33
34       System.exit( 0 );  // terminate application
35
36    }  // end method main
37
38 }  // end class OddEven
```



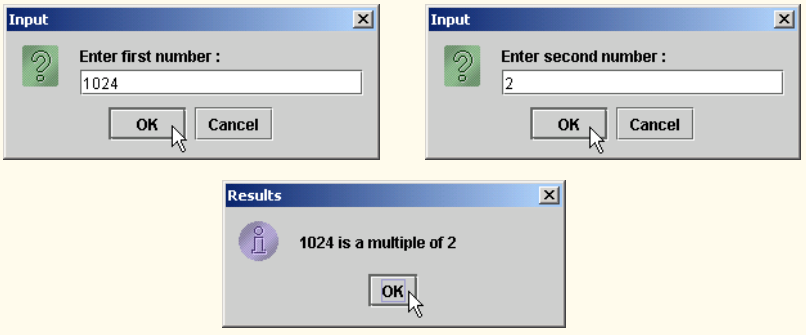


2.28 Write an application that reads two integers and determines whether the first is a multiple of the second and prints the result. [*Hint*: Use the remainder operator.]

ANS:

```
1  // Exercise 2.28 Solution: Multiple.java
2  // Program determines if the first number
3  // entered is a multiple of the second number entered
4  // ( i.e., 10 (first #) is a multiple of 2 (second #) )
5
6  import javax.swing.JOptionPane;
7
8  public class Multiple {
9
```

```
10      public static void main( String args[] )
11      {
12         int firstNumber, secondNumber;
13         String input;
14
15         // read from user as a string, and convert type String
16         // to type int for both numbers ...
17         input = JOptionPane.showInputDialog( "Enter first number :" );
18         firstNumber = Integer.parseInt( input );
19
20         input = JOptionPane.showInputDialog( "Enter second number :" );
21         secondNumber = Integer.parseInt( input );
22
23         // create the appropriate output string
24         String result;
25         if ( firstNumber % secondNumber == 0 )
26            result = firstNumber + " is a multiple of " +
27               secondNumber + "\n";
28         else
29            result = firstNumber + " is not a multiple of " +
30               secondNumber + "\n";
31
32         // show the results
33         JOptionPane.showMessageDialog( null, result,
34            "Results", JOptionPane.INFORMATION_MESSAGE );
35
36         System.exit( 0 );
37
38      } // end method main
39
40   }  // end class Multiple
```



2.29 Write an application that displays in the command window a checkerboard pattern as follows:

```
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
```

ANS:

```
1  // Exercise 2.29 Solution: Checker.java
2  // Program that draws a checkerboard.
3
4  public class Checker {
5
6     // main method begins execution of Java application
7     public static void main( String args[] )
8     {
9        System.out.println( "* * * * * * * * *" );
10       System.out.println( " * * * * * * * * *" );
11       System.out.println( "* * * * * * * * *" );
12       System.out.println( " * * * * * * * * *" );
13       System.out.println( "* * * * * * * * *" );
14       System.out.println( " * * * * * * * * *" );
15       System.out.println( "* * * * * * * * *" );
16       System.out.println( " * * * * * * * * *" );
17    }
18 }
```

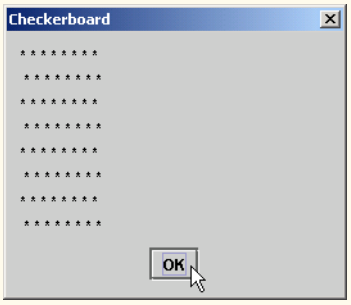
```
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
* * * * * * * * *
 * * * * * * * * *
```

2.30 Modify the program you wrote in Exercise 2.29 to display the checkerboard pattern in a `JOptionPane.PLAIN_MESSAGE` dialog. Does the program display the shapes exactly as in Exercise 2.29?

ANS:

```
1  // Exercise 2.30 Solution: Checker.java
2  // Program draws a checkerboard in a plain message dialog
3
4  import javax.swing.JOptionPane;
5
6  public class Checker {
```

```
 7
 8     public static void main( String args[] )
 9     {
10        // create the checkerboard as a string
11        String checkerboardString =
12           "* * * * * * * *\n" +
13           " * * * * * * * *\n" +
14           "* * * * * * * *\n" +
15           " * * * * * * * *\n" +
16           "* * * * * * * *\n" +
17           " * * * * * * * *\n" +
18           "* * * * * * * *\n" +
19           " * * * * * * * *\n";
20
21        JOptionPane.showMessageDialog( null, checkerboardString,
22           "Checkerboard", JOptionPane.PLAIN_MESSAGE );
23        System.exit( 0 );
24     }
25
26  }  // end class Checker
```



2.31 Here's a peek ahead. In this chapter, you have learned about integers and the type `int`. Java can also represent uppercase letters, lowercase letters and a considerable variety of special symbols. Every character has a corresponding integer representation. The set of characters a computer uses and the corresponding integer representations for those characters is called that computer's *character set*. You can indicate a character value in a program simply by enclosing that character in single quotes, as in `'A'`.

You can determine the integer equivalent of a character by preceding that character with `(int)`, as in

```
(int) 'A'
```

This form is called a *cast operator*. (We will say more about these in Chapter 4.) The following statement outputs a character and its integer equivalent:

```
System.out.println( "The character " + 'A' +
   " has the value " + ( int ) 'A' );
```

When the preceding statement executes, it displays the character `A` and the value `65` (from the so-called Unicode character set) as part of the string.

Using statements similar to the one shown eariler in this exercise, write an application that displays the integer equivalents of some uppercase letters, lowercase letters, digits and special symbols. Display the integer equivalents of the following: `A B C a b c 0 1 2 $ * + /` and the blank character.

ANS:

```
1  // Exercise 2.31 Solution: Display.java
2  // Program that prints a unicode character
3  // and its integer equivalent.
4
5  public class Display {
6
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10       System.out.println( "The character " + 'A' +
11          " has the value " + ( int ) 'A' );
12       System.out.println( "The character " + 'B' +
13          " has the value " + ( int ) 'B' );
14       System.out.println( "The character " + 'C' +
15          " has the value " + ( int ) 'C' );
16       System.out.println( "The character " + 'a' +
17          " has the value " + ( int ) 'a' );
18       System.out.println( "The character " + 'b' +
19          " has the value " + ( int ) 'b' );
20       System.out.println( "The character " + 'c' +
21          " has the value " + ( int ) 'c' );
22       System.out.println( "The character " + '0' +
23          " has the value " + ( int ) '0' );
24       System.out.println( "The character " + '1' +
25          " has the value " + ( int ) '1' );
26       System.out.println( "The character " + '2' +
27          " has the value " + ( int ) '2' );
28       System.out.println( "The character " + '$' +
29          " has the value " + ( int ) '$' );
30       System.out.println( "The character " + '*' +
31          " has the value " + ( int ) '*' );
32       System.out.println( "The character " + '+' +
33          " has the value " + ( int ) '+' );
34       System.out.println( "The character " + '/' +
35          " has the value " + ( int ) '/' );
36       System.out.println( "The character " + ' ' +
37          " has the value " + ( int ) ' ' );
38
39    } // end method main
40
41 } // end class Display
```

```
The character A has the value 65
The character B has the value 66
The character C has the value 67
The character a has the value 97
The character b has the value 98
The character c has the value 99
The character 0 has the value 48
The character 1 has the value 49
The character 2 has the value 50
The character $ has the value 36
The character * has the value 42
The character + has the value 43
The character / has the value 47
The character   has the value 32
```

2.32 Write an application that inputs one number consisting of five digits from the user, separates the number into its individual digits and prints the digits separated from one another by three spaces each. For example, if the user types in the number 42339, the program should print

```
4   2   3   3   9
```

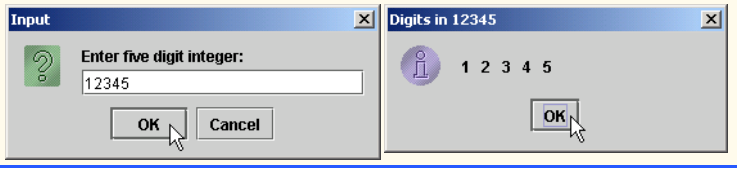
[*Hint*: It is possible to do this exercise with the techniques you learned in this chapter. You will need to use both division and remainder operations to "pick off" each digit.]

Assume that the user enters the correct number of digits. What happens when you execute the program and type a number with more than five digits? What happens when you execute the program and type a number with fewer than five digits?

ANS:

```
1  // Exercise 2.32 Solution: Five.java
2  // Program breaks apart a five-digit number
3
4  import javax.swing.JOptionPane;
5
6  public class Five {
7
8     public static void main( String args[] )
9     {
10       int number;
11       String inputString;
12
13       // read the five digit number from user as a string
14       inputString = JOptionPane.showInputDialog(
15          "Enter five digit integer:" );
16
17       number = Integer.parseInt( inputString );
18
19       // determine the 5 digits
20       int digit1, digit2, digit3, digit4, digit5;
```

```
21         digit1 = number / 10000;
22         digit2 = number % 10000 / 1000;
23         digit3 = number % 10000 % 1000 / 100;
24         digit4 = number % 10000 % 1000 % 100 / 10;
25         digit5 = number % 10000 % 1000 % 100 % 10;
26
27         // create the result string
28         String resultString = digit1 + "   " + digit2 + "   " +
29            digit3 + "   " + digit4 + "   " + digit5 ;
30
31         // display results
32         JOptionPane.showMessageDialog( null, resultString,
33            "Digits in " + number, JOptionPane.INFORMATION_MESSAGE );
34
35         System.exit( 0 );
36
37      } // end method main
38
39   }  // end class Five
```



2.33 Using only the programming techniques you learned in this chapter, write an application that calculates the squares and cubes of the numbers from 0 to 10 and prints the resulting values in table format as shown below. [*Note*: This program does not require any input from the user.]

```
number  square  cube
0       0       0
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

[*Note*: This program does not require any input from the user.]

ANS:

```
1  // Exercise 2.33 Solution: Numbers.java
2  // Program prints a table of squares and cubes of
3  // numbers from 0 to 10.
```

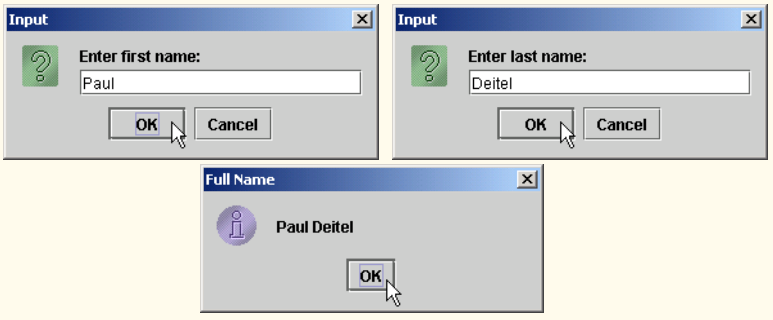
```
 4
 5  public class Numbers {
 6
 7     public static void main( String args[] )
 8     {
 9        // print a header for the table
10        System.out.println( "number square cube" );
11
12        // print x, x squared and x cubed for each value
13        int x = 0;
14        System.out.println ( x + "      " + x * x + "      " + x * x * x );
15        x = 1;
16        System.out.println ( x + "      " + x * x + "      " + x * x * x );
17        x = 2;
18        System.out.println ( x + "      " + x * x + "      " + x * x * x );
19        x = 3;
20        System.out.println ( x + "      " + x * x + "      " + x * x * x );
21        x = 4;
22        System.out.println ( x + "      " + x * x + "     " + x * x * x );
23        x = 5;
24        System.out.println ( x + "      " + x * x + "     " + x * x * x );
25        x = 6;
26        System.out.println ( x + "      " + x * x + "     " + x * x * x );
27        x = 7;
28        System.out.println ( x + "      " + x * x + "     " + x * x *x  );
29        x = 8;
30        System.out.println ( x + "      " + x * x + "     " + x * x * x );
31        x = 9;
32        System.out.println ( x + "      " + x * x + "     " + x * x * x );
33        x = 10;
34        System.out.println ( x + "     " + x * x + "    " + x * x * x );
35
36        System.exit( 0 );
37
38     } // end method main
39
40  }  // end class Numbers
```

```
number square cube
0      0      0
1      1      1
2      4      8
3      9      27
4      16     64
5      25     125
6      36     216
7      49     343
8      64     512
9      81     729
10     100    1000
```

2.34 Write a program that reads a first name and a last name from the user as two separate inputs and concatenates the first name and last name, separating them by a space. Display the concatenated name in a message dialog.

ANS:

```
1  // Exercise 2.34 Solution: Name.java
2  // Program displays a full name after receiving separate
3  // first and last names from the user.
4
5  import javax.swing.JOptionPane;
6
7  public class Name {
8
9     public static void main( String args[] )
10    {
11       String firstName,    // first string entered by user
12              lastName;     // last string entered by user
13
14       // read first name from user
15       firstName = JOptionPane.showInputDialog( "Enter first name: " );
16
17       // read last name from user
18       lastName = JOptionPane.showInputDialog( "Enter last name: " );
19
20       // display results
21       JOptionPane.showMessageDialog( null, firstName + " " +
22          lastName, "Full Name", JOptionPane.INFORMATION_MESSAGE );
23
24       System.exit( 0 );
25
26    } // end method main
27
28 }  // end class Name
```

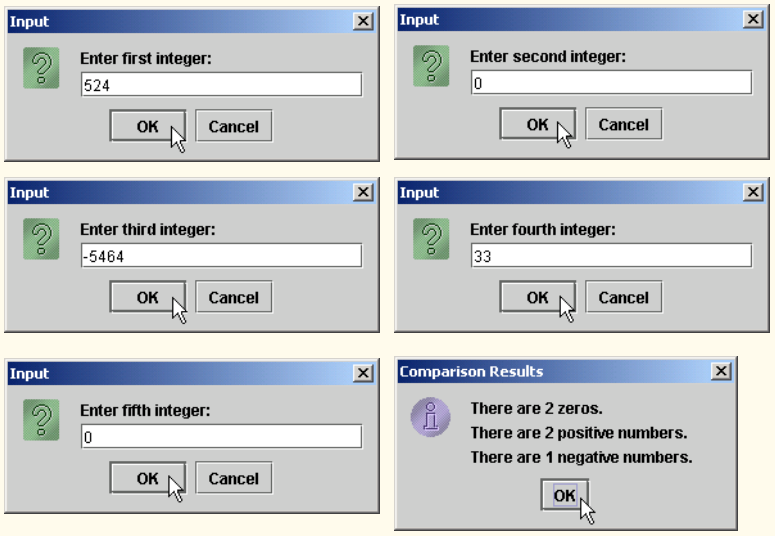


2.35 Write a program that inputs five numbers and determines and prints the number of negative numbers input, the number of positive numbers input and the number of zeros input.

ANS:

```
1  // Exercise 2.35 Solution: Tally.java
2  // Program accepts five numbers as input,and prints
3  // a tally of the number of negatives, positives
4  // and zeros
5
6  import javax.swing.JOptionPane;
7
8  public class Tally {
9
10    public static void main( String args[] )
11    {
12       String inputString;
13       int inputNumber, zeroTally, positiveTally, negativeTally;
14
15       // assign values to the counters
16       zeroTally = 0;
17       positiveTally = 0;
18       negativeTally = 0;
19
20       // read first number from user as a string
21       inputString = JOptionPane.showInputDialog( "Enter first integer:" );
22       inputNumber = Integer.parseInt( inputString );
23
24       if ( inputNumber == 0 )
25          zeroTally = zeroTally + 1;
26
27       if ( inputNumber < 0 )
28          negativeTally = negativeTally + 1;
29
30       if ( inputNumber > 0 )
31          positiveTally = positiveTally + 1;
32
33       // read second number from user as a string
34       inputString = JOptionPane.showInputDialog(
35          "Enter second integer:" );
36       inputNumber = Integer.parseInt( inputString );
37
38       if ( inputNumber == 0 )
39          zeroTally = zeroTally + 1;
40
41       if ( inputNumber < 0 )
42          negativeTally = negativeTally + 1;
43
44       if ( inputNumber > 0 )
45          positiveTally = positiveTally + 1;
46
47       // read third number from user as a string
48       inputString = JOptionPane.showInputDialog( "Enter third integer:" );
49       inputNumber = Integer.parseInt( inputString );
50
```

```
51          if ( inputNumber == 0 )
52             zeroTally = zeroTally + 1;
53
54          if ( inputNumber < 0 )
55             negativeTally = negativeTally + 1;
56
57          if ( inputNumber > 0 )
58             positiveTally = positiveTally + 1;
59
60          // read fouth number from user as a string
61          inputString = JOptionPane.showInputDialog(
62             "Enter fourth integer:" );
63          inputNumber = Integer.parseInt( inputString );
64
65          if ( inputNumber == 0 )
66             zeroTally = zeroTally + 1;
67
68          if ( inputNumber < 0 )
69             negativeTally = negativeTally + 1;
70
71          if ( inputNumber > 0 )
72             positiveTally = positiveTally + 1;
73
74          // read fifth number from user as a string
75          inputString = JOptionPane.showInputDialog( "Enter fifth integer:" );
76          inputNumber = Integer.parseInt( inputString );
77
78          if ( inputNumber == 0 )
79             zeroTally = zeroTally + 1;
80
81          if ( inputNumber < 0 )
82             negativeTally = negativeTally + 1;
83
84          if ( inputNumber > 0 )
85             positiveTally = positiveTally + 1;
86
87          // create a string describing the results
88          String resultString = "There are " + zeroTally + " zeros.\n" +
89             "There are " + positiveTally + " positive numbers.\n" +
90             "There are " + negativeTally + " negative numbers.\n";
91
92          // display results
93          JOptionPane.showMessageDialog( null, resultString,
94             "Comparison Results", JOptionPane.INFORMATION_MESSAGE );
95
96          System.exit( 0 );
97
98       } // end method main
99
100 }  // end class Tally
```



3

Introduction to Java Applets

Objectives

- To differentiate between applets and applications.
- To observe some of Java's exciting capabilities through the Java 2 Software Development Kit's demonstration applets.
- To be able to write simple Java applets.
- To be able to write a simple HyperText Markup Language (HTML) document to load an applet into the `appletviewer` or a Web browser and execute the applet.
- To understand the difference between variables and references.

He would answer to "Hi!" or to any loud cry,
Such as "Fry me!" or "Fritter my wig!"
To "What-you-may-call-um!" or "What-was-his-name!"
But especially "Thing-um-a-jig!"
Lewis Carroll

Painting is only a bridge linking the painter's mind with that of the viewer.
Eugène Delacroix

My method is to take the utmost trouble to find the right thing to say, and then to say it with the utmost levity.
George Bernard Shaw

Though this be madness, yet there is method in 't.
William Shakespeare

SELF-REVIEW EXERCISES

3.1 Fill in the blanks in each of the following.

a) Class _____ provides methods for drawing.
ANS: `Graphics`

b) Java applets begin execution with a series of three method calls: _____, _____ and _____.
ANS: `init`, `start`, `paint`

c) Methods _____ and _____ display lines and rectangles.
ANS: `drawLine`, `drawRect`

d) Keyword _____ indicates that a new class is a subclass of an existing class.
ANS: `extends`

e) Every Java 2 applet should extend class _____.
ANS: `JApplet`

f) Java's eight primitive types are _____, _____, _____, _____, _____, _____, _____ and _____.
ANS: `char`, `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`.

3.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) To draw a rectangle, method `drawRect` requires four arguments that specify two points on the applet.
ANS: False. Method `drawRect` requires four arguments—two that specify the upper-left corner of the rectangle and two that specify the width and height of the rectangle.

b) Method `drawLine` requires four arguments that specify two points on the applet to draw a line.
ANS: True

c) Type `Double` is a primitive type.
ANS: False. Type `Double` is a class in the `java.lang` package; `double` is a primitive date type. Remember that names that start with a capital letter are normally class names.

d) Type `int` is used to declare a floating-point number.
ANS: False. Type `double` or type `float` can be used to declare a floating-point number. Type `int` is used to declare integers.

e) Method `Double.parseDouble` converts a `String` to a primitive `double` value.
ANS: True.

3.3 Write Java statements to accomplish each of the following:

a) Display a dialog asking the user to enter a floating-point number.
ANS:
```
stringValue = JOptionPane.showInputDialog(
   "Enter a floating-point number" );
```

b) Convert a string to a floating-point number and store the converted value in `double` variable `age`. Assume that the string is stored in `stringValue`.
ANS:
```
age = Double.parseDouble( stringValue );
```

c) Draw the message "`This is a Java program`" on one line at position *(10, 10)* on an applet (assume you are defining this statement in the applet's `paint` method).
ANS:
```
g.drawString( "This is a Java program", 10, 10 );
```

d) Draw the message "`This is a Java program`" on two lines starting at position *(10, 10)* on an applet (assume these statements are defined in applet method `paint`). Have the first line end with `Java`. Make the two lines start at the same *x*-coordinate.
ANS:
```
g.drawString( "This is a Java", 10, 10 );
g.drawString( "program", 10, 25 );
```

3.4 What is the difference between a local variable and a field?
ANS: A local variable is declared in the body of a method and can be used only from the point at which it is declared through the end of the method declaration. A field is declared in a class, but not in the body of any of that class's methods. Every object (instance) of a class has a separate copy of that class's fields. Also, the fields are accessible to all methods of the class. (We will see an exception to this in Chapter 8.)

EXERCISES

3.5 Fill in the blanks in each of the following:
a) Type _____ declares a single-precision floating-point variable.
ANS: `float`

b) If class `Double` provides method `parseDouble` to convert a string to a `double` and class `Integer` provides method `parseInt` to convert a string to an `int`, then class `Float` probably provides method _____ to convert a string to a `float`.
ANS: `parseFloat`

c) Type _____ is used to declare double-precision, floating-point variables.
ANS: `double`

d) The _____ or a browser can be used to execute a Java applet.
ANS: appletviewer

e) To load an applet into a browser, you must first define a(n) _____ file.
ANS: HTML

f) The _____ and _____ HTML tags specify that an applet should be loaded into an applet container and executed.
ANS: `<applet>, </applet>`.

3.6 State whether each of the following is *true* or *false*. If *false*, explain why.
a) The application that executes an applet is generically referred to as an applet container.
ANS: False. The browser that executes an applet is generically referred to as an applet container.
b) When using an `import` declaration of the form `javax.swing.*`, all classes in the package are imported.
ANS: False. The compiler searches the package only for those classes used in the program.

c) You do not need import declarations if the full package name and class name are specified each time you refer to a class in a program.

ANS: True.

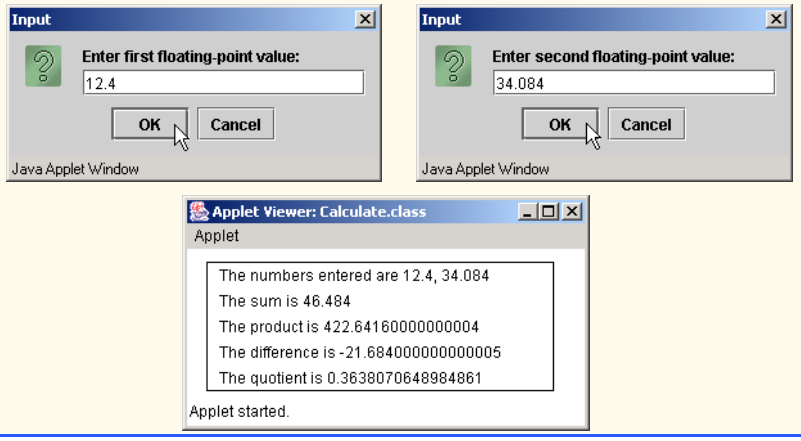
3.7 Write an applet that asks the user to enter two floating-point numbers, obtains the two numbers from the user and draws the sum, product (multiplication), difference and quotient (division) of the two numbers. Use the techniques shown in Fig. 3.13.

ANS:

```
<html>
<applet code = "Calculate.class" width = "310" height = "115" >
</applet>
</html>
```

```
// Exercise 3.7 Solution: Calculate.java
// Applet accepts two numbers as input, and computes
// the sum, product, difference and quotient

import java.awt.Graphics;
import javax.swing.*;

public class Calculate extends JApplet {

   String dataString;         // input string
   String sumString;          // sum string
   String productString;      // product string
   String differenceString;   // difference string
   String quotientString;     // quotient string

   // obtain numerical input and calculate results
   public void init()
   {
      // strings entered by user
      String firstNumber, secondNumber;

      // numerical values from input strings
      double number1, number2;

      // read first number from user as a string
      firstNumber = JOptionPane.showInputDialog(
         "Enter first floating-point value:" );

      // read second number from user as a string
      secondNumber = JOptionPane.showInputDialog(
         "Enter second floating-point value:" );

      // convert numbers from type String to type double
      number1 = Double.parseDouble( firstNumber );
      number2 = Double.parseDouble( secondNumber );

      // perform calculations on numbers
      double sum = number1 + number2;
      double product = number1 * number2;
```

```
40          double difference = number1 - number2;
41          double quotient = number1 / number2;
42
43          // assign numbers entered to a string
44          dataString = "The numbers entered are " + number1 + ", " + number2;
45
46          // assign calculated results to strings
47          sumString = "The sum is " + sum;
48          productString = "The product is " + product;
49          differenceString = "The difference is " + difference;
50          quotientString = "The quotient is " + quotient;
51
52       }  // end method init
53
54       // draw results on applet's background
55       public void paint( Graphics g )
56       {
57          // draw results
58          g.drawRect( 15, 10, 270, 100 );
59          g.drawString( dataString, 25, 25 );
60          g.drawString( sumString, 25, 45 );
61          g.drawString( productString, 25, 65 );
62          g.drawString( differenceString, 25, 85 );
63          g.drawString( quotientString, 25, 105 );
64
65       }  // end method paint
66
67    }  // end class Calculate
```



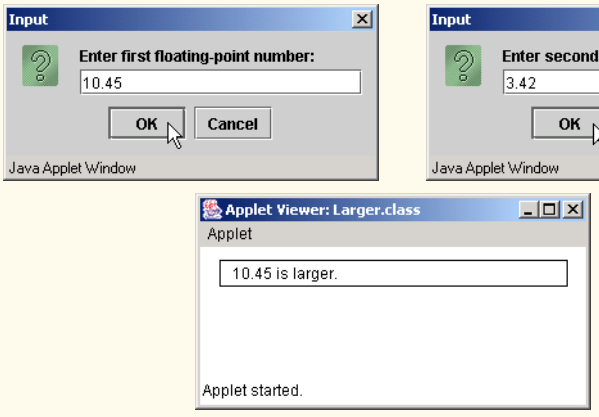
3.8 Write an applet that asks the user to enter two floating-point numbers, obtains the numbers from the user and displays the larger number followed by the words "`is larger`" as a string on the applet. If the numbers are equal, print the message "`These numbers are equal`." Use the techniques shown in Fig. 3.13.

ANS:

```
<html>
<applet code = "Larger.class" width = "300" height = "100" >
</applet>
</html>
```

```
// Exercise 3.8 Solution: Larger.java
// Program accepts two floating point numbers as input
// and determines which number is larger.

import java.awt.Graphics;   // import class Graphics
import javax.swing.*;       // import package javax.swing

public class Larger extends JApplet {
   String result;   // String containing the output

   // initialize applet by obtaining values from user
   public void init()
   {
      String firstNumber;    // first String entered by user
      String secondNumber;   // second String entered by user
      double number1;        // first number to compare
      double number2;        // second number to compare

      // read first number from user as a String
      firstNumber = JOptionPane.showInputDialog(
         "Enter first floating-point number:" );

      // read second number from user as a String
      secondNumber = JOptionPane.showInputDialog(
         "Enter second floating-point number:" );

      // convert numbers from type String to type double
      number1 = Double.parseDouble( firstNumber );
      number2 = Double.parseDouble( secondNumber );

      if ( number1 > number2 )
         result = number1 + " is larger.";

      if ( number1 < number2 )
         result = number2 + " is larger.";

      if ( number1 == number2 )
         result = "These numbers are equal.";

   }  // end method init

```

```
42      // draw results in a rectangle on applet's background
43      public void paint( Graphics g )
44      {
45         // draw rectangle starting from (15, 10) that is 270
46         // pixels wide and 20 pixels tall
47         g.drawRect( 15, 10, 270, 20 );
48
49         //draw result as a String at (25, 25)
50         g.drawString( result, 25, 25 );
51
52      }  // end method paint
53
54   }  // end class Larger
```



3.9 Write an applet that inputs three floating-point numbers from the user and displays the sum, average, product, smallest and largest of these numbers as strings on the applet. Use the techniques shown in Fig. 3.13.

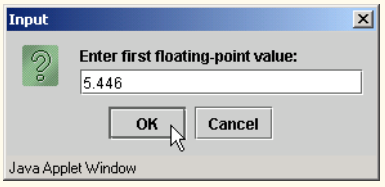
ANS:

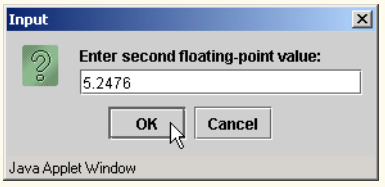
```
1  <html>
2  <applet code = "Calculate2.class" width = "310" height = "115" >
3  </applet>
4  </html>
```

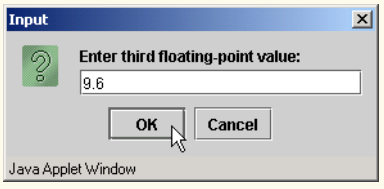
```
1  // Exercise 3.9 Solution: Calculate2.java
2  // Applet accepts three floating-point numbers as input,
3  // and computes the sum, average, product, smallest and largest
4  // numbers and displays the results in the applet window.
5
6  import java.awt.Graphics;
7  import javax.swing.*;
8
9  public class Calculate2 extends JApplet {
10
```

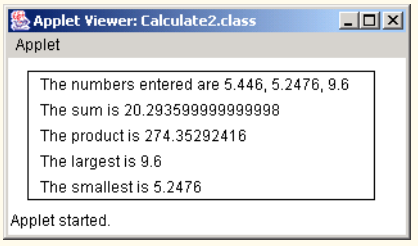
```
11       String dataString;          // output string
12       String sumString;           // sum string
13       String averageString;       // average string
14       String productString;       // product string
15       String smallestString;      // smallest number string
16       String largestString;       // largest number string
17
18       // obtain numerical input and determine results
19       public void init()
20       {
21          String inputString;   // string entered by user
22
23          // first, second and third number
24          double number1, number2, number3;
25
26          // read first number from user as a string
27          inputString = JOptionPane.showInputDialog(
28             "Enter first floating-point value:" );
29
30          // convert number from type String to type double
31          number1 = Double.parseDouble( inputString );
32
33          // smallest and largest numbers initialized to first number
34          double smallest = number1;
35          double largest = number1;
36
37          // read first number from user as a string
38          inputString = JOptionPane.showInputDialog(
39             "Enter second floating-point value:" );
40
41          // convert number from type String to type double
42          number2 = Double.parseDouble( inputString );
43
44          if ( smallest > number2 )
45             smallest = number2;
46
47          if ( largest < number2 )
48             largest = number2;
49
50          // read third number from user as a string
51          inputString = JOptionPane.showInputDialog(
52             "Enter third floating-point value:" );
53
54          // convert number from type String to type double
55          number3 = Double.parseDouble( inputString );
56
57          if ( smallest > number3 )
58             smallest = number3;
59
60          if ( largest < number3 )
61             largest = number3;
62
63          // calculated results
64          double sum, average, product;
```

```
65
66          // perform calculations on numbers
67          sum = number1 + number2 + number3;
68          average = sum / 3.0;
69          product = number1 * number2 * number3;
70
71          // assign numbers entered to a string
72          dataString = "The numbers entered are " + number1 + ", " +
73             number2 + ", " + number3;
74
75          // assign calculated results to strings
76          sumString = "The sum is " + sum;
77          productString = "The product is " + product;
78          smallestString = "The smallest is " + smallest;
79          largestString = "The largest is " + largest;
80
81       }  // end method init
82
83       // draw results on applet's background
84       public void paint( Graphics g )
85       {
86          // draw results
87          g.drawRect( 15, 10, 270, 100 );
88          g.drawString( dataString, 25, 25 );
89          g.drawString( sumString, 25, 45 );
90          g.drawString( productString, 25, 65 );
91          g.drawString( largestString, 25, 85 );
92          g.drawString( smallestString, 25, 105 );
93
94       }  // end method paint
95
96    }  // end class Calculate2
```









3.10 Write an applet that asks the user to input the radius of a circle as a floating-point number and draws the circle's diameter, circumference and area. Use the value 3.14159 for π. Use the techniques shown in Fig. 3.13. [*Note*: You may also use the predefined constant `Math.PI` for the value of π. This constant is more precise than the value 3.14159. Class `Math` is defined in the `java.lang` package, so you do not need to `import` it.] Use the following formulas (*r* is the radius):

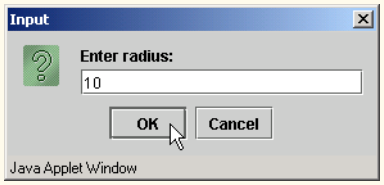
diameter = 2*r*
circumference = 2π*r*
area = πr^2

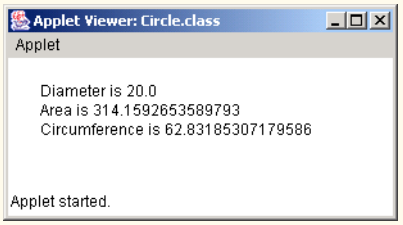
ANS:

```
1  <html>
2  <applet code = "Circle.class" width = "300" height = "100" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.10 Solution: Circle.java
2  // Program calculates the area, circumference
3  // and diameter for a circle
4
5  import java.awt.Graphics;   // import class Graphics
6  import javax.swing.*;       // import package javax.swing
7
8  public class Circle extends JApplet {
9
10    // Strings for output
11    String line1;
12    String line2;
13    String line3;
14
15    // initialize applet by obtaining values from user
16    public void init()
17    {
18       String input;   // String entered by user
19       double radius;     // radius of circle
20
21       // read from user as String
22       input = JOptionPane.showInputDialog( "Enter radius:" );
23
24       // convert number from type String to type int
25       radius = Double.parseDouble( input );
26
27       line1 = "Diameter is " + ( 2 * radius );
28       line2 = "Area is " + ( Math.PI * radius * radius );
29       line3 = "Circumference is " + ( 2 * Math.PI * radius );
30
31    }  // end method init
32
33    // draw results on applet's background
34    public void paint( Graphics g )
35    {
```

```
36         //draw line1 as a String at (25, 30)
37         g.drawString( line1, 25, 30 );
38
39         //draw line2 as a String at (25, 45)
40         g.drawString( line2, 25, 45 );
41
42         //draw line3 as a String at (25, 60)
43         g.drawString( line3, 25, 60 );
44
45      }  // end method paint
46
47   }  // end class Circle
```





3.11 Write an applet that reads five integers and determines and prints the largest and smallest integers in the group. Use only the programming techniques you learned in this chapter and Chapter 2. Draw the results on the applet.

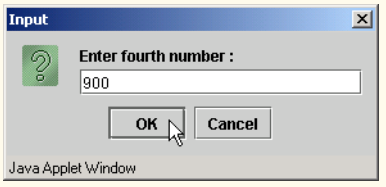
ANS:

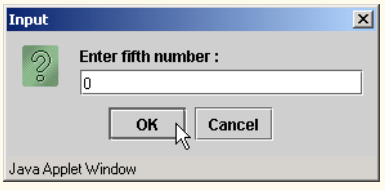
```
1  <html>
2  <applet code = "LargeSmall.class" width = "310" height = "115" >
3  </applet>
4  </html>
```

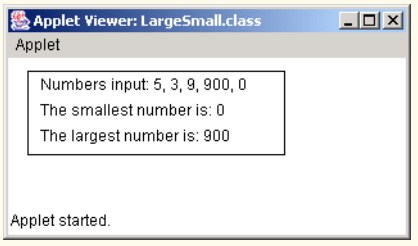
```
1  // Exercise 3.11 Solution: LargeSmall.java
2  // Calculates the largest and smallest
3  // of five integers entered one at a time.
4
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class LargeSmall extends JApplet {
9
10    String dataString;   // output string
11    String smallString;  // smallest number string
12    String largeString;  // largest number string
13
14    // obtain numerical input and determine results
15    public void init()
16    {
17       String input;     // string entered by user
18
```

```
19         // input integers
20         int firstNumber, secondNumber, thirdNumber,
21            fourthNumber, fifthNumber;
22
23         // largest and smallest integers
24         int largest, smallest;
25
26         // first value
27         input = JOptionPane.showInputDialog( "Enter first number :" );
28         firstNumber = Integer.parseInt( input );
29
30         // initialize the smallest and largest
31         smallest = largest = firstNumber;
32
33         // second value
34         input = JOptionPane.showInputDialog( "Enter second number :" );
35         secondNumber = Integer.parseInt( input );
36
37         if ( smallest > secondNumber )
38            smallest = secondNumber;
39
40         if ( largest < secondNumber )
41            largest = secondNumber;
42
43         // third value
44         input = JOptionPane.showInputDialog( "Enter third number :" );
45         thirdNumber = Integer.parseInt( input );
46
47         if ( smallest > thirdNumber )
48            smallest = thirdNumber;
49
50         if ( largest < thirdNumber )
51            largest = thirdNumber;
52
53         // fourth value
54         input = JOptionPane.showInputDialog( "Enter fourth number :" );
55         fourthNumber = Integer.parseInt( input );
56
57         if ( smallest > fourthNumber )
58            smallest = fourthNumber;
59
60         if ( largest < fourthNumber )
61            largest = fourthNumber;
62
63         // fifth value
64         input = JOptionPane.showInputDialog( "Enter fifth number :" );
65         fifthNumber = Integer.parseInt( input );
66
67         if ( smallest > fifthNumber )
68            smallest = fifthNumber;
69
70         if ( largest < fifthNumber )
71            largest = fifthNumber;
72
```

```
73          // create output Strings
74          dataString = "Numbers input: " + firstNumber + ", " + secondNumber +
75             ", " + thirdNumber + ", " + fourthNumber + ", " + fifthNumber;
76
77          smallString = "The smallest number is: " + smallest;
78          largeString = "The largest number is: " + largest;
79
80       }  // end method init
81
82       // draw results on applet window
83       public void paint( Graphics g )
84       {
85          g.drawRect( 15, 10, 200, 65);
86          g.drawString( dataString, 25, 25 );
87          g.drawString( smallString, 25, 45 );
88          g.drawString( largeString, 25, 65 );
89       }
90
91    }  // end class LargeSmall
```







3.12 What does the following code print?

```
g.drawString( "*", 25, 25 );
g.drawString( "***", 25, 55 );
g.drawString( "*****", 25, 85 );
g.drawString( "****", 25, 70 );
g.drawString( "**", 25, 40 );
```

ANS:

```
*
**
***
****
*****
```

3.13 Write an applet that draws a checkerboard pattern as follows:

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

ANS:

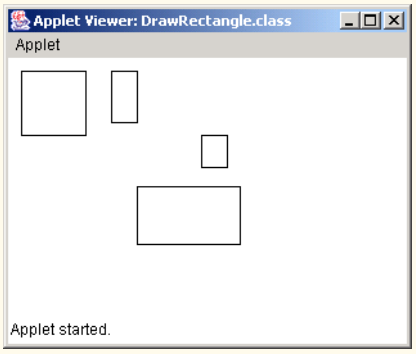
```
1  <html>
2  <applet code = "Checker.class" width = "50" height = "50" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.13 Solution: Checker.java
2  // Program draws a checkerboard.
3
4  import java.awt.Graphics;   // import class Graphics
5  import javax.swing.*;       // import package javax.swing
6
7  public class Checker extends JApplet {
8
9     // draw Strings on applet's background
10    public void paint( Graphics g )
11    {
12       g.drawString( "* * * * * * * *", 25, 20 );
13       g.drawString( " * * * * * * * *", 25, 28 );
14       g.drawString( "* * * * * * * *", 25, 36 );
15       g.drawString( " * * * * * * * *", 25, 44 );
16       g.drawString( "* * * * * * * *", 25, 52 );
17       g.drawString( " * * * * * * * *", 25, 60 );
18       g.drawString( "* * * * * * * *", 25, 68 );
19       g.drawString( " * * * * * * * *", 25, 76 );
20    }
21
22 } // end class Checker
```

3.14 Write an applet that draws rectangles of different sizes and locations.
ANS:

```
1  <html>
2  <applet code = "DrawRectangle.class" width = "310" height = "200" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.14 Solution: DrawRectangle.java
2  // Applet draws several rectangles on the applet window
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class DrawRectangle extends JApplet {
8
9     // draws four rectangles at different locations
10    public void paint( Graphics g )
11    {
12       g.drawRect( 10, 10, 50, 50 );
13       g.drawRect( 80, 10, 20, 40 );
14       g.drawRect( 100, 100, 80, 45 );
15       g.drawRect( 150, 60, 20, 25 );
16    }
17
18 }  // end class DrawRectangle
```

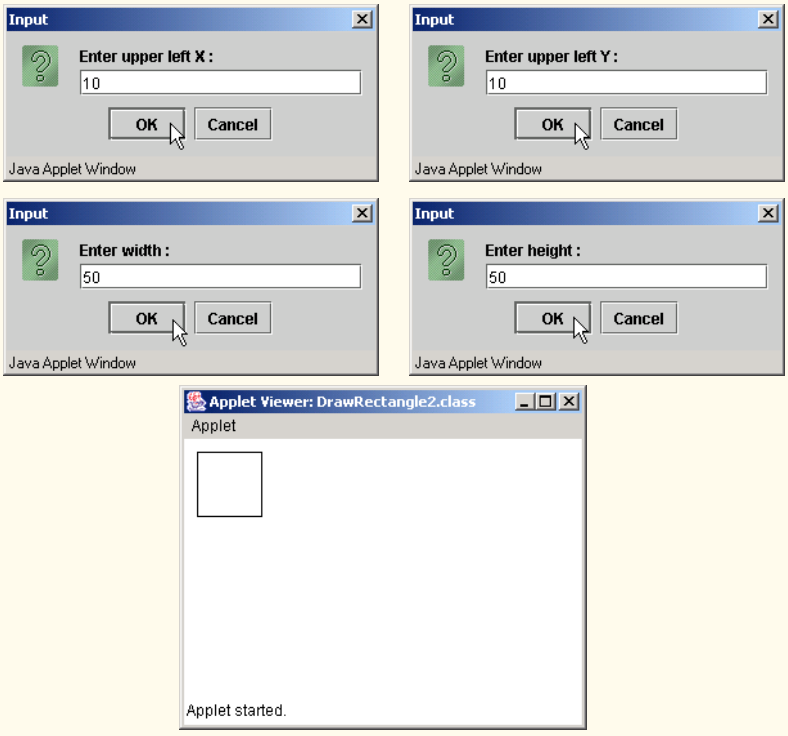


3.15 Write an applet that allows the user to input values for the arguments required by method `drawRect`, then draws a rectangle using the four input values.

ANS:

```
<html>
<applet code = "DrawRectangle2.class" width = "310" height = "200" >
</applet>
</html>
```

```
// Exercise 3.15 Solution: DrawRectangle2.java
// Draws a rectangle on the applet window whose
// dimension and location are specified by the user

import java.awt.*;
import javax.swing.*;

public class DrawRectangle2 extends JApplet {

   int upperLeftX, upperLeftY;  // rectangle coordinates
   int width, height;           // rectangle width, height

   // obtain rectangle dimensions and coordinates from user
   public void init()
   {
      // string entered by user
      String inputString;

      // read upper left x-value from user
      inputString = JOptionPane.showInputDialog( "Enter upper left X :" );
      upperLeftX = Integer.parseInt( inputString );

      // read upper right y-value from user
      inputString = JOptionPane.showInputDialog( "Enter upper left Y :" );
      upperLeftY = Integer.parseInt( inputString );

      // read width from user
      inputString = JOptionPane.showInputDialog( "Enter width :" );
      width = Integer.parseInt( inputString );

      // read height from user
      inputString = JOptionPane.showInputDialog( "Enter height :" );
      height = Integer.parseInt( inputString );

   }  // end method init

   // draw user-specified rectangle
   public void paint( Graphics g )
   {
      g.drawRect( upperLeftX, upperLeftY, width, height );
   }

}  // end class DrawRectangle2
```



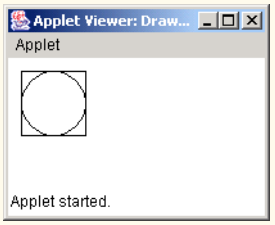
3.16 Class `Graphics` contains method `drawOval`, which takes as arguments the same four arguments as method `drawRect`. The arguments for method `drawOval` specify the "bounding box" for the oval—the sides of the bounding box are the boundaries of the oval. Write a Java applet that draws an oval and a rectangle with the same four arguments. The oval will touch the rectangle at the center of each side.

ANS:

```
1  <html>
2  <applet code = "Draw3.class" width = "200" height = "100" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.16 Solution: Draw3.java
2  // Program draws an oval inside a rectangle on the applet.
3
4  import java.awt.Graphics;   // import class Graphics
5  import javax.swing.*;       // import package javax.swing
6
7  public class Draw3 extends JApplet {
8
9     // draw shapes on applet's background
10    public void paint( Graphics g )
11    {
```

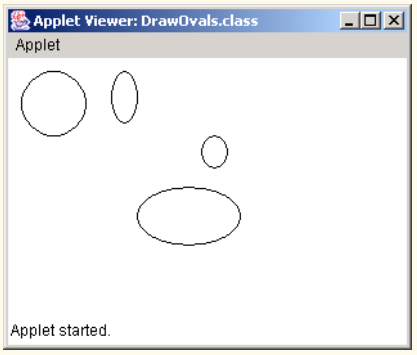
```
12         // draw rectangle starting at (10, 10) that is 50
13         // pixels wide and 50 pixels tall
14         g.drawRect( 10, 10, 50, 50 );
15
16         // draw oval starting at (10, 10) that is 50 pixels
17         // wide and 50 pixels tall
18         g.drawOval( 10, 10, 50, 50 );
19      }
20
21   } // end class Draw3
```



3.17 Modify the solution to Exercise 3.16 to output ovals of different shapes and sizes.
ANS:

```
1  <html>
2  <applet code = "DrawOvals.class" width = "310" height = "200" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.17 Solution: DrawOvals.java
2  // Applet draws several ovals on the applet window
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class DrawOvals extends JApplet {
8
9     // draws four ovals on the Applet
10    public void paint( Graphics g )
11    {
12       g.drawOval( 10, 10, 50, 50 );
13       g.drawOval( 80, 10, 20, 40 );
14       g.drawOval( 100, 100, 80, 45 );
15       g.drawOval( 150, 60, 20, 25 );
16    }
17
18 }  // end class DrawOvals
```



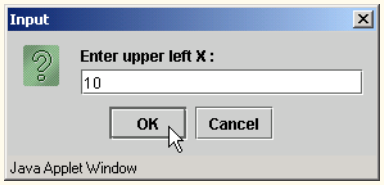
3.18 Write an applet that allows the user to input the four arguments required by method `drawOval`, then draws an oval using the four input values.

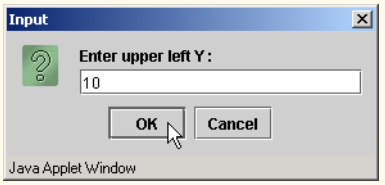
ANS:

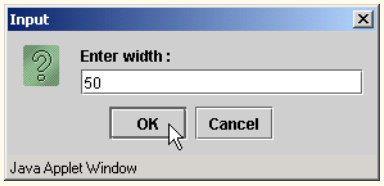
```
1  <html>
2  <applet code = "DrawOval.class" width = "310" height = "200" >
3  </applet>
4  </html>
```

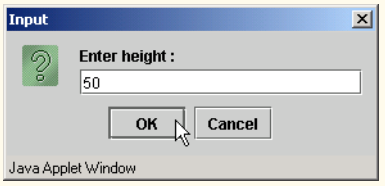
```
1  // Exercise 3.18 Solution: DrawOval.java
2  // Draws an oval on the applet window
3  // whose dimension and location are specified by the user
4
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class DrawOval extends JApplet {
9
10    int upperLeftX, upperLeftY;  // rectangle coordinates
11    int width, height;           // rectangle width, height
12
13    // obtain rectangle dimensions and coordinates from user
14    public void init()
15    {
16       // string entered by user
17       String inputString;
18
19       // read upper left coordinate from user as a String
20       inputString = JOptionPane.showInputDialog( "Enter upper left X :" );
21       upperLeftX = Integer.parseInt( inputString );
22
23       // read upper left coordinate from user as a String
24       inputString = JOptionPane.showInputDialog( "Enter upper left Y :" );
25       upperLeftY = Integer.parseInt( inputString );
26
27       // read width from user as a string
28       inputString = JOptionPane.showInputDialog( "Enter width :" );
```

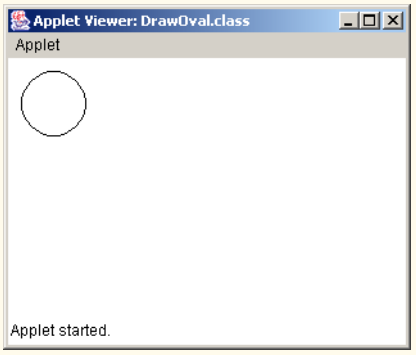
```
29          width = Integer.parseInt( inputString );
30
31          // read height from user as a string
32          inputString = JOptionPane.showInputDialog( "Enter height :" );
33          height = Integer.parseInt( inputString );
34
35       }  // end method init
36
37       // draw the new oval
38       public void paint( Graphics g )
39       {
40          g.drawOval( upperLeftX, upperLeftY, width, height );
41       }
42
43    }  // end class DrawOval
```











3.19 Using only programming techniques from this chapter and Chapter 2, write an applet that calculates the squares and cubes of the numbers from 0 to 10 and draws the resulting values in table format as shown below. [*Note*: This program does not require any input from the user.]

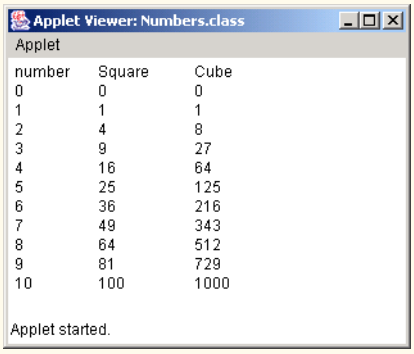
```
number  square  cube
0       0       0
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
6       36      216
7       49      343
8       64      512
9       81      729
10      100     1000
```

ANS:

```
1  <html>
2  <applet code = "Numbers.class" width = "310" height = "200" >
3  </applet>
4  </html>
```

```
1  // Exercise 3.19 Solution: Numbers.java
2  // Prints a table of the cubics and squares
3  // of the integers from 0 to 10
4
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class Numbers extends JApplet {
9
10    // draw calculated squares and cubics table
11    public void paint( Graphics g )
12    {
13
14       // draw a title row
15       g.drawString( "number", 5, 15 );
16       g.drawString( "Square", 70, 15 );
17       g.drawString( "Cube", 145, 15 );
18
19       // draw row with x, x squared, and x cubed for each
20       // value of x from 0 to 10
21       int x = 0;
22       g.drawString( "" + x, 5, 30 );
23       g.drawString( "" + ( x * x ), 70, 30 );
24       g.drawString( "" + ( x * x * x ), 145, 30 );
25
26       x = 1;
27       g.drawString( "" + x, 5, 45 );
```

```
28          g.drawString( "" + ( x * x ), 70, 45 );
29          g.drawString( "" + ( x * x * x ), 145, 45 );
30
31          x = 2;
32          g.drawString( "" + x, 5, 60 );
33          g.drawString( "" + ( x * x ), 70, 60 );
34          g.drawString( "" + ( x * x * x ), 145, 60 );
35
36          x = 3;
37          g.drawString( "" + x, 5, 75 );
38          g.drawString( "" + ( x * x ), 70, 75 );
39          g.drawString( "" + ( x * x * x ), 145, 75 );
40
41          x = 4;
42          g.drawString( "" + x, 5, 90 );
43          g.drawString( "" + ( x * x ), 70, 90 );
44          g.drawString( "" + ( x * x * x ), 145, 90 );
45
46          x = 5;
47          g.drawString( "" + x, 5, 105 );
48          g.drawString( "" + ( x * x ), 70, 105 );
49          g.drawString( "" + ( x * x * x ), 145, 105 );
50
51          x = 6;
52          g.drawString( "" + x, 5, 120 );
53          g.drawString( "" + ( x * x ), 70, 120 );
54          g.drawString( "" + ( x * x * x ), 145, 120 );
55
56          x = 7;
57          g.drawString( "" + x, 5, 135 );
58          g.drawString( "" + ( x * x ), 70, 135 );
59          g.drawString( "" + ( x * x * x ), 145, 135 );
60
61          x = 8;
62          g.drawString( "" + x, 5, 150 );
63          g.drawString( "" + ( x * x ), 70, 150 );
64          g.drawString( "" + ( x * x * x ), 145, 150 );
65
66          x = 9;
67          g.drawString( "" + x, 5, 165 );
68          g.drawString( "" + ( x * x ), 70, 165 );
69          g.drawString( "" + ( x * x * x ), 145, 165 );
70
71          x = 10;
72          g.drawString( "" + x, 5, 180 );
73          g.drawString( "" + ( x * x ), 70, 180 );
74          g.drawString( "" + ( x * x * x ), 145, 180 );
75
76       }  // end method paint
77
78    }  // end class Numbers
```



4

Control Statements: Part 1

Objectives

- To understand basic problem-solving techniques.
- To be able to develop algorithms through the process of top-down, stepwise refinement.
- To be able to use the `if` and `if...else` selection statements to choose among alternative actions.
- To be able to use the `while` repetition statement to execute statements in a program repeatedly.
- To understand counter-controlled repetition and sentinel-controlled repetition.
- To be able to use the assignment, increment and decrement operators.

Let's all move one place on.
Lewis Carroll

The wheel is come full circle.
William Shakespeare

How many apples fell on Newton's head before he took the hint!
Robert Frost

SELF-REVIEW EXERCISES

4.1 Fill in the blanks in each of the following statements:

a) All programs can be written in terms of three types of control structures: _____, _____ and _____.

ANS: sequence, selection, repetition

b) The _____ statement is used to execute one action when a condition is true and another action when that condition is false.

ANS: `if...else`

c) Repeating a set of instructions a specific number of times is called _____ repetition.

ANS: counter-controlled (or definite)

d) When it is not known in advance how many times a set of statements will be repeated, a(n) _____ value can be used to terminate the repetition.

ANS: sentinel, signal, flag or dummy

4.2 Write four different Java statements that each add 1 to integer variable `x`.

ANS:
```
x = x + 1;
x += 1;
++x;
x++;
```

4.3 Write Java statements to accomplish each of the following tasks:

a) Assign the sum of `x` and `y` to `z`, and increment `x` by 1 after the calculation. Use only one statement.

ANS:
```
z = x++ + y;
```

b) Test whether variable `count` is greater than 10. If it is, print `"Count is greater than 10"`.

ANS:
```
if ( count > 10 )
   System.out.println( "Count is greater than 10" );
```

c) Decrement the variable `x` by 1, then subtract it from the variable `total`. Use only one statement.

ANS:
```
total -= --x;
```

d) Calculate the remainder after `q` is divided by `divisor`, and assign the result to `q`. Write this statement in two different ways.

ANS:
```
q %= divisor;
q = q % divisor;
```

4.4 Write a Java statement to accomplish each of the following tasks:

a) Declare variables `sum` and `x` to be of type `int`.

ANS:
```
int sum, x;
```

b) Assign 1 to variable `x`.

ANS:
```
x = 1;
```

c) Assign 0 to variable `sum`.
ANS: `sum = 0;`

d) Add variable `x` to variable `sum`, and assign the result to variable `sum`.
ANS: `sum += x;` or `sum = sum + x;`

e) Print `"The sum is: "`, followed by the value of variable `sum`.
ANS: `System.out.println( "The sum is: " + sum );`

4.5 Combine the statements that you wrote in Exercise 4.4 into a Java application that calculates and prints the sum of the integers from 1 to 10. Use a `while` statement to loop through the calculation and increment statements. The loop should terminate when the value of `x` becomes 11.
ANS:

```
1  // Calculate the sum of the integers from 1 to 10
2  public class Calculate {
3
4     public static void main( String args[] )
5     {
6        int sum, x;
7
8        x = 1;
9        sum = 0;
10
11       while ( x <= 10 ) {
12          sum += x;
13          ++x;
14       }
15
16       System.out.println( "The sum is: " + sum );
17
18    } // end main
19
20 } // end class Calculate
```

4.6 Determine the value of each of the following variables after the calculation is performed. Assume that when each statement begins executing, all variables are type `int` and have the value 5.
a) `product *= x++;`
ANS: `product` = 25, `x` = 6

b) `quotient /= ++x;`
ANS: `quotient = 0, x = 6`

4.7 Identify and correct the errors in each of the following sets of code:
a)
```
while ( c <= 5 ) {
      product *= c;
      ++c;
```
ANS: Error: The closing right brace of the `while` statement's body is missing.
Correction: Add a closing right brace after the statement `++c;`.

b)
```
if ( gender == 1 )
   System.out.println( "Woman" );
else;
   System.out.println( "Man" );
```
ANS: Error: Semicolon after `else` results in a logic error. The second output statement will always be executed.
Correction: Remove the semicolon after `else`.

4.8 What is wrong with the following `while` statement?

```
while ( z >= 0 )
   sum += z;
```

ANS: The value of the variable `z` is never changed in the `while` statement. Therefore, if the loop-continuation condition `( z >= 0 )` is true, an infinite loop is created. To prevent an infinite loop from occurring, `z` must be decremented so that it eventually becomes less than 0.

EXERCISES

4.9 Identify and correct the errors in each of the following pieces of code. [*Note*: There may be more than one error in each piece of code.]

a)
```
if ( age >= 65 );
   System.out.println( "Age greater than or equal to 65" );
else
   System.out.println( "Age is less than 65 )";
```
ANS: Semicolon at the end of the `if` condition should be removed. The closing double quote of the second `System.out.println` should be inside of the closing parenthesis.

b)
```
int x = 1, total;
while ( x <= 10 ) {
   total += x;
   ++x;
}
```
ANS: The variable total should be initialized to zero.

c)
```
While ( x <= 100 )
   total += x;
   ++x;
```
ANS: The W in `While` should be lowercase. The two statements should be enclosed in curly braces to properly group them into the body of the `while`; otherwise the loop will be an infinite loop.

d)
```
while ( y > 0 ) {
   System.out.println( y );
   ++y;
```
ANS: The ++ operator should be changed to `--`. The closing curly brace for the `while` loop is missing.

4.10 What does the following program print?

```
1  public class Mystery {
2
3     public static void main( String args[] )
4     {
5        int y, x = 1, total = 0;
6
7        while ( x <= 10 ) {
8           y = x * x;
9           System.out.println( y );
10          total += y;
11          ++x;
12       }
13
14       System.out.println( "Total is " + total );
15
16    } // end main
17
18 } // end class Mystery
```

ANS:

```
1
4
9
16
25
36
49
64
81
100
Total is 385
```

For Exercise 4.11 through Exercise 4.14, perform each of the following steps:

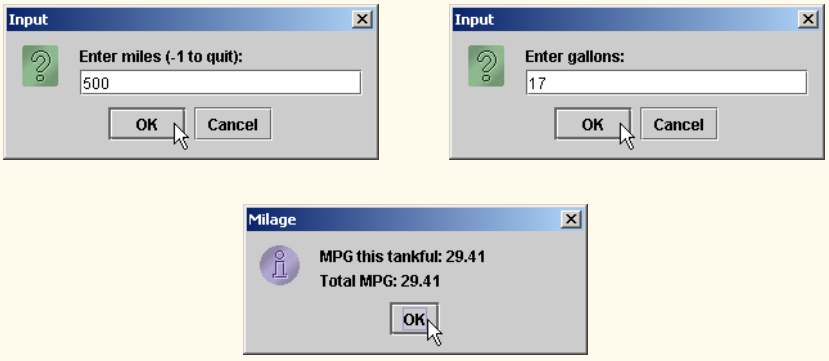
a) Read the problem statement.
b) Formulate the algorithm using pseudocode and top-down, stepwise refinement.
c) Write a Java program.
d) Test, debug and execute the Java program.
e) Process three complete sets of data.

4.11 Drivers are concerned with the mileage their automobiles get. One driver has kept track of several tankfuls of gasoline by recording miles driven and gallons used for each tankful. Develop a Java application that will input the miles driven and gallons used (both as integers) for each tankful. The program should calculate and display the miles per gallon obtained for each tankful and print the combined miles per gallon obtained for all tankfuls up to this point. All averaging calculations should produce floating-point results. Use input dialogs to obtain the data from the user.

ANS:

```
// Exercise 4.11 Solution: Gas.java
// Program calculates average mpg
import java.text.DecimalFormat;
import javax.swing.JOptionPane;

public class Gas {

   public static void main( String args[] )
   {
      int miles, gallons, totalMiles = 0, totalGallons = 0;
      float milesPerGallon, totalMilesPerGallon;
      String inputMiles, inputGallons, result = "";

      // read first number from user as a string
      inputMiles = JOptionPane.showInputDialog(
         "Enter miles (-1 to quit):" );

      // convert miles from type String to type int
      miles = Integer.parseInt( inputMiles );

      // exit if the input is -1 otherwise, proceed with the program
      while ( miles != -1 ) {
         // read second number from user as String
         inputGallons = JOptionPane.showInputDialog( "Enter gallons:" );

         // convert gallons from type String to type int
         gallons = Integer.parseInt( inputGallons );

         totalMiles += miles;
         totalGallons += gallons;

         DecimalFormat twoDigits = new DecimalFormat( "0.00" );

         if ( gallons != 0 ) {
            milesPerGallon = (float) miles / gallons;
            result = "MPG this tankful: " +
               twoDigits.format( milesPerGallon ) + "\n";
         }

         totalMilesPerGallon = (float) totalMiles / totalGallons;
         result += "Total MPG: " +
            twoDigits.format( totalMilesPerGallon ) + "\n";

         JOptionPane.showMessageDialog( null, result, "Milage",
            JOptionPane.INFORMATION_MESSAGE );

         // input new value for miles and convert from String to int
         inputMiles = JOptionPane.showInputDialog(
            "Enter miles (-1 to quit):" );
         miles = Integer.parseInt( inputMiles );

      }  // end while loop
```

```
53
54          System.exit( 0 );
55
56       }  // end method main
57
58    }  // end class Gas
```



4.12 Develop a Java application that will determine whether a department-store customer has exceeded the credit limit on a charge account. For each customer, the following facts are available:

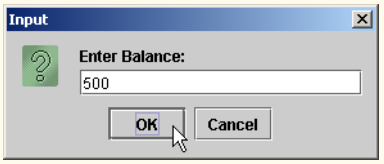
a) account number,
b) balance at the beginning of the month,
c) total of all items charged by the customer this month,
d) total of all credits applied to the customer's account this month and
e) allowed credit limit.

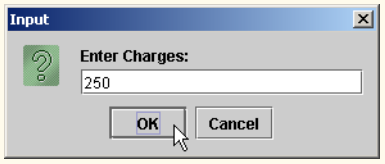
The program should input each of these facts from input dialogs as integers, calculate the new balance (= *beginning balance* + *charges* – *credits*), display the new balance and determine whether the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the message "Credit limit exceeded."

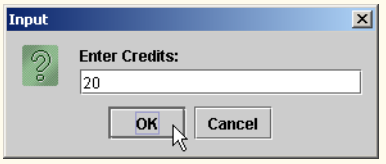
ANS:

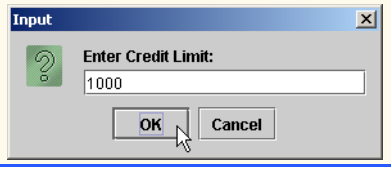
```
1   // Exercise 4.12 Solution: Credit.java
2   // Program monitors accounts.
3   import java.awt.*;
4   import javax.swing.JOptionPane;
5
6   public class Credit {
7
8      public static void main( String args[] )
9      {
10        String inputString,          // user input
11               resultsString,        // result String
12               creditStatusString;   // credit status
13        int account,                 // account number
14            oldBalance,              // starting balance
15            charges,                 // total charges
16            credits,                 // total credits
```

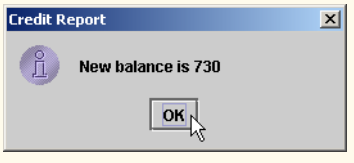
```
17             creditLimit,             // allowed credit limit
18             newBalance;              // new balance
19
20          inputString = JOptionPane.showInputDialog(
21             "Enter Account Number:" );
22          account = Integer.parseInt( inputString );
23
24          inputString = JOptionPane.showInputDialog( "Enter Balance: " );
25          oldBalance = Integer.parseInt( inputString );
26
27          inputString = JOptionPane.showInputDialog( "Enter Charges: " );
28          charges = Integer.parseInt( inputString );
29
30          inputString = JOptionPane.showInputDialog( "Enter Credits: " );
31          credits = Integer.parseInt( inputString );
32
33          inputString = JOptionPane.showInputDialog( "Enter Credit Limit: " );
34          creditLimit = Integer.parseInt( inputString );
35
36          newBalance = oldBalance + charges - credits;
37
38          resultsString = "New balance is " + newBalance;
39
40          if ( newBalance > creditLimit )
41             creditStatusString = "CREDIT LIMIT EXCEEDED";
42          else
43             creditStatusString = "Credit Report";
44
45          JOptionPane.showMessageDialog( null, resultsString,
46             creditStatusString, JOptionPane.INFORMATION_MESSAGE );
47
48          System.exit( 0 );
49
50       }  // end method main
51
52    }  // end class Credit
```











4.13 A large company pays its salespeople on a commission basis. The salespeople receive $200 per week, plus 9% of their gross sales for that week. For example, a salesperson who sells $5000 worth of merchandise in a week receives $200 plus 9% of $5000, or a total of $650. You have been supplied with a list of items sold by each salesperson. The values of these items are as follows:

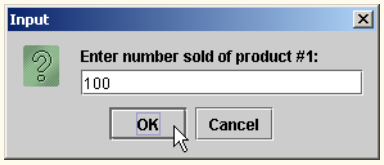
```
Item    Value
1       239.99
2       129.75
3        99.95
4       350.89
```

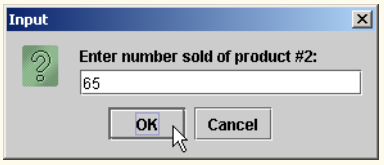
Develop a Java application that inputs one salesperson's items sold for last week and calculates and displays that salesperson's earnings. There is no limit to the number of items that can be sold by a salesperson.

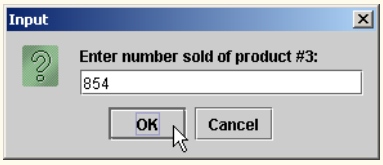
ANS:

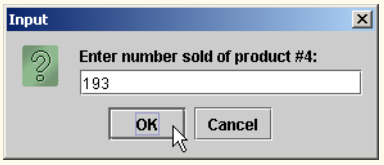
```
1  // Exercise 4.13 Solution: Sales.java
2  // Program calculates commissions based on sales.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class Sales {
7
8     public static void main( String args[] )
9     {
10       double gross = 0.0, earnings;
11       int product = 0, numberSold;
12       String input;
13
14       while ( product < 4 ) {
15          product++;
16
17          // read number from user as String
18          input = JOptionPane.showInputDialog(
19             "Enter number sold of product #" + product + ":" );
20
21          // convert numbers from type String to type int
22          numberSold = Integer.parseInt( input );
23
24          // determine gross of each individual product and add to total
25          if ( product == 1 )
26             gross += numberSold * 239.99;
27
28          else if ( product == 2 )
29             gross += numberSold * 129.75;
30
31          else if ( product == 3 )
32             gross += numberSold * 99.95;
33
34          else if ( product == 4 )
35             gross += numberSold * 350.89;
36
37       }  // end while
38
```

```
39          DecimalFormat twoDigits = new DecimalFormat( "0.00" );
40
41          earnings = 0.09 * gross + 200;
42          String result = "Earnings this week: " +
43             twoDigits.format( earnings );
44
45          JOptionPane.showMessageDialog( null, result, "Sales",
46             JOptionPane.INFORMATION_MESSAGE );
47
48          System.exit( 0 );
49
50       }  // end method main
51
52    }  // end class Sales
```







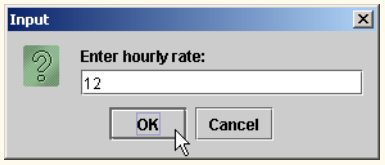


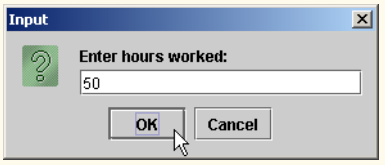
4.14 Develop a Java application that will determine the gross pay for each of three employees. The company pays "straight time" for the first 40 hours worked by each employee and pays "time and a half" for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee and should determine and display the employee's gross pay. Use input dialogs to input the data.

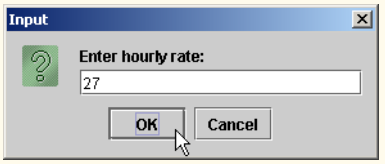
ANS:

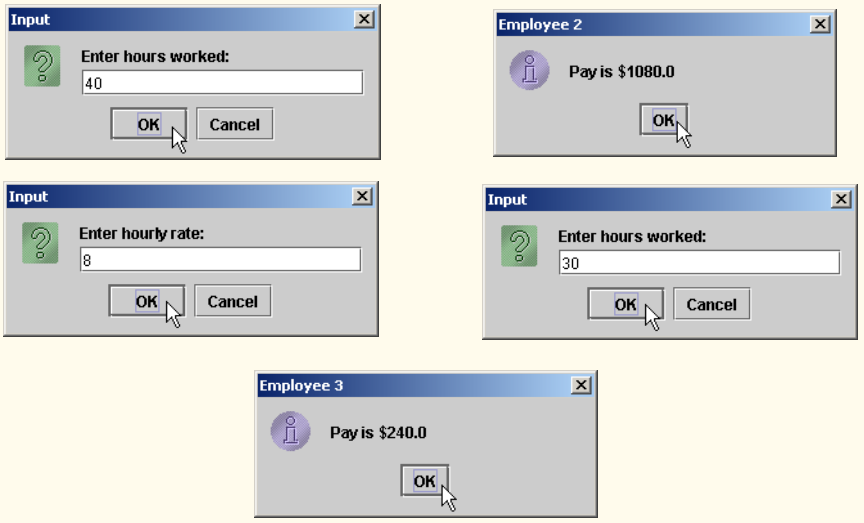
```
1  // Exercise 4.14 Solution: Wages.java
2  // Program calculates wages.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class Wages {
7
8     public static void main( String args[] )
9     {
```

```
10          String inputString,   // user input
11                 resultsString;  // result String
12          double pay;           // gross pay
13          int hours,            // hours worked
14              rate,             // hourly rate
15              count = 1;        // number of employees
16
17          // repeat calculation for 3 employees
18          while ( count <= 3 ) {
19
20             inputString = JOptionPane.showInputDialog(
21                "Enter hourly rate: " );
22             rate = Integer.parseInt( inputString );
23
24             inputString = JOptionPane.showInputDialog(
25                "Enter hours worked: " );
26             hours = Integer.parseInt( inputString );
27
28             // with overtime
29             if ( hours > 40 )
30                pay = ( 40.0 * rate ) + ( hours - 40 ) * ( rate * 1.5 );
31
32             else // straight time
33                pay = hours * rate;
34
35             resultsString = "Pay is $" + pay;
36
37             JOptionPane.showMessageDialog( null, resultsString,
38                "Employee " + count, JOptionPane.INFORMATION_MESSAGE );
39
40             count++;
41          }
42
43          System.exit( 0 );
44
45       }  // end method main
46
47    }  // end class Wages
```









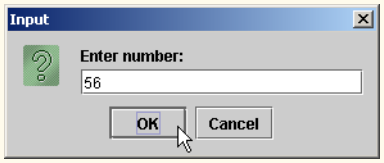
4.15 The process of finding the largest value (i.e., the maximum of a group of values) is used frequently in computer applications. For example, a program that determines the winner of a sales contest would input the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a pseudocode program and then a Java application that inputs a series of 10 single-digit numbers as characters and determines and prints the largest of the numbers. Your program should use at least the following three variables:

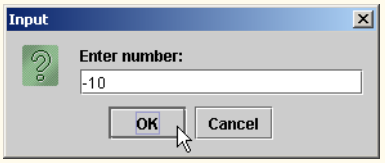
a) `counter`: A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed);
b) `number`: The current digit input to the program;
c) `largest`: The largest number found so far.

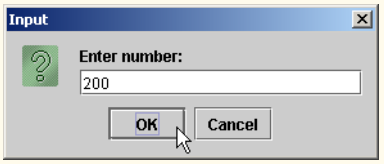
ANS: Pseudocode: Input the first number directly into the variable largest.

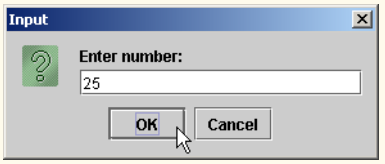
```
1  // Exercise 4.15 Solution: Largest.java
2  // Program determines and prints the largest of ten numbers.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Largest {
7
8     public static void main( String args[] )
9     {
10       int largest,   // largest number
11           number,    // user input
12           counter;   // number of inputted values
13
14       // get first number and assign it to variable largest
15       largest = Integer.parseInt(
16          JOptionPane.showInputDialog( "Enter number: " ) );
17
18       counter = 2;
19
```

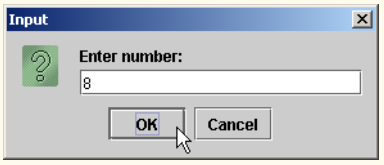
```
20         // get rest numbers and find the largest
21         while ( counter <= 10 ) {
22            number = Integer.parseInt(
23               JOptionPane.showInputDialog( "Enter number: " ) );
24
25            if ( number > largest )
26               largest = number;
27
28            counter++;
29         }
30
31         JOptionPane.showMessageDialog( null, "Largest number is " + largest,
32            "Largest", JOptionPane.INFORMATION_MESSAGE );
33
34         System.exit( 0 );
35
36      }  // end method main
37
38   }  // end class Largest
```

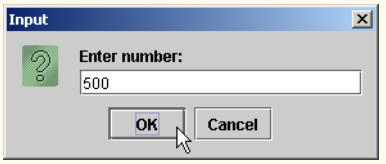


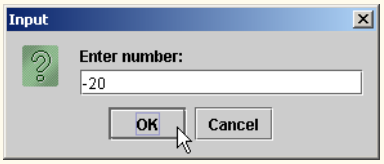


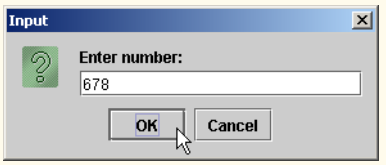


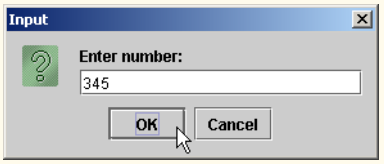


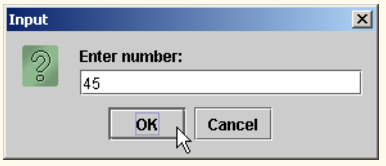












4.16 Write a Java application that uses looping to print the following table of values:

```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

ANS:

```
1  // Exercise 4.16 Solution: Table.java
2  // Program prints a table of values using a while loop.
3
4  public class Table {
5
6     public static void main( String args[] )
7     {
8        int n = 1;
9
10       System.out.println( "N\t10*N\t100*N\t1000*N\n" );
11
12       while ( n <= 5 ) {
13          System.out.println( n + "\t" + ( 10 * n ) +
14             "\t" + ( 100 * n ) + "\t" + ( 1000 * n ) );
15          n++;
16       }
17    }
18
19 } // end class Table
```

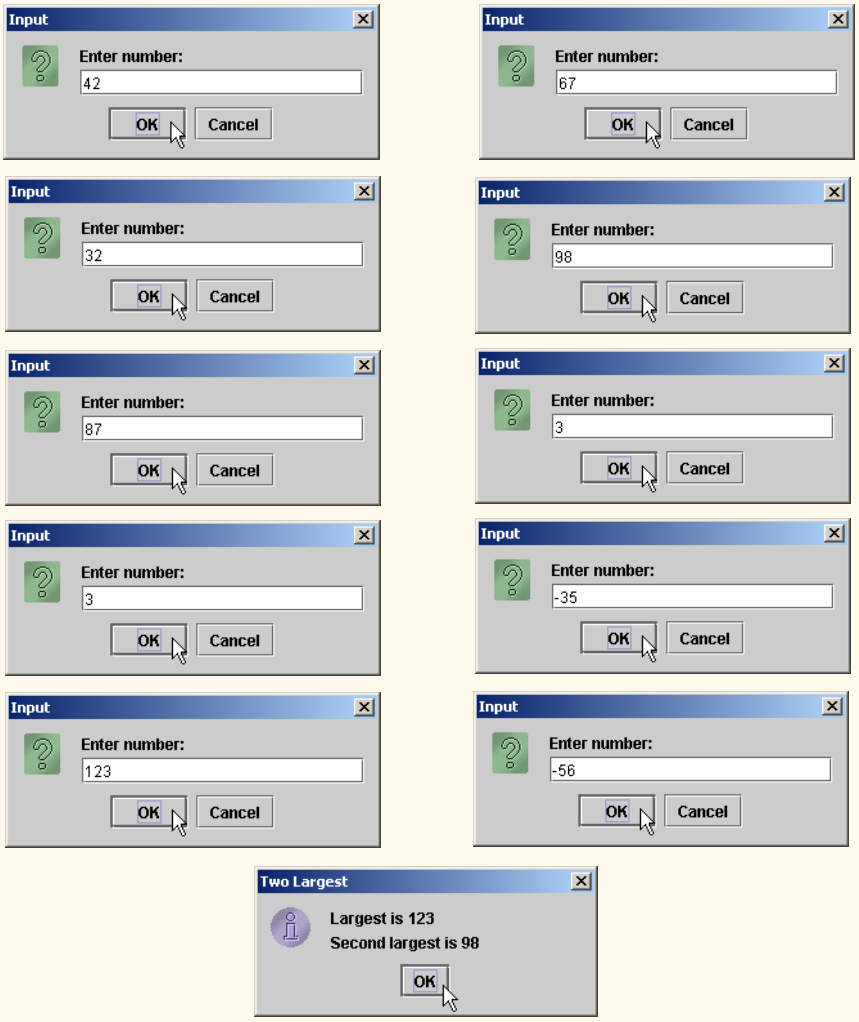
```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

4.17 Using an approach similar to that for Exercise 4.15, find the *two* largest values of the 10 digits entered. [*Note*: You may input each number only once.]

ANS:

```
1  // Exercise 4.17 Solution: TwoLargest.java
2  // Program determines and prints the two largest of ten numbers.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class TwoLargest {
7
8     public static void main( String args[] )
9     {
10       int largest,         // largest number
11           nextLargest,     // second largest number
12           number,          // user input
13           counter;         // number of values inputted
14       String resultsString;   // result String
15
16       // get first number and assign it to variable largest
17       largest = Integer.parseInt(
18          JOptionPane.showInputDialog( "Enter number: " ) );
19
20       // get second number and compare it with first number
21       number = Integer.parseInt(
22          JOptionPane.showInputDialog( "Enter number: " ) );
23
24       if ( number > largest ) {
25          nextLargest = largest;
26          largest = number;
27       }
28       else
29          nextLargest = number;
30
31       counter = 3;
32
33       // get rest numbers and find the largest and nextLargest
34       while ( counter <= 10 ) {
35          number = Integer.parseInt(
36             JOptionPane.showInputDialog( "Enter number: " ) );
37
38          if ( number > largest ) {
39             nextLargest = largest;
40             largest = number;
41          }
42
43          else if ( number > nextLargest )
44             nextLargest = number;
45
46          counter++;
47       }
48
49       resultsString = "Largest is " + largest +
50          "\nSecond largest is " + nextLargest;
51
52       JOptionPane.showMessageDialog( null, resultsString,
53          "Two Largest", JOptionPane.INFORMATION_MESSAGE );
```

```
54
55          System.exit( 0 );
56
57       }  // end method main
58
59    }  // end class TwoLargest
```

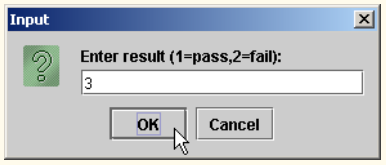


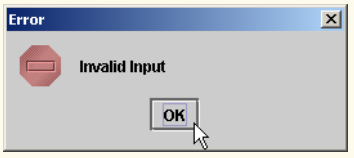
4.18 Modify the program in Fig. 4.11 to validate its inputs. For any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value.

ANS:

```
1  // Exercise 4.18 Solution: Analysis.java
2  // Program performs analysis of examination results.
```

```
3  import javax.swing.JOptionPane;
4
5  public class Analysis {
6
7     public static void main( String args[] )
8     {
9        // initializing variables in declarations
10       int passes = 0, failures = 0, student = 1, result;
11       String input, output;
12
13       // process 10 students; counter-controlled loop
14       while ( student <= 10 ) {
15          input = JOptionPane.showInputDialog(
16             "Enter result (1=pass,2=fail): " );
17
18          result = Integer.parseInt( input );
19
20          if ( result == 1 ) {     // if...else nested in while
21             passes++;
22             student++;
23          }
24
25          else if ( result == 2 ) {
26             failures++;
27             student++;
28          }
29
30          else
31             JOptionPane.showMessageDialog( null, "Invalid Input",
32                "Error", JOptionPane.ERROR_MESSAGE );
33       }
34
35       output = "Passed: " + passes + "\nFailed: " + failures;
36
37       if ( passes > 8 )
38          output += "\nRaise tuition ";
39
40       JOptionPane.showMessageDialog( null, output,
41          "Results", JOptionPane.INFORMATION_MESSAGE );
42
43       System.exit( 0 );
44
45    }  // end method main
46
47 }  // end class Analysis
```





4.19 What does the following program print?

```
1  public class Mystery2 {
2
3     public static void main( String args[] )
4     {
5        int count = 1;
6
7        while ( count <= 10 ) {
8           System.out.println( count % 2 == 1 ? "****" : "++++++++" );
9
10          ++count;
11       }
12
13    } // end main
14
15 } // end class Mystery2
```

ANS:

```
****
++++++++
****
++++++++
****
++++++++
****
++++++++
****
++++++++
```

4.20 What does the following program print?

```
1  public class Mystery3 {
2
3     public static void main( String args[] )
4     {
5        int row = 10, column;
6
7        while ( row >= 1 ) {
8           column = 1;
9
10          while ( column <= 10 ) {
11             System.out.print( row % 2 == 1 ? "<" : ">" );
12             ++column;
13          }
14
15          --row;
16          System.out.println();
17       }

18
```

```
19      } // end main
20
21   } // end class Mystery3
```

ANS:

```
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
>>>>>>>>>>
<<<<<<<<<<
```

4.21 *(Dangling-`else` Problem)* Determine the output for each of the given sets of code when `x` is 9 and `y` is 11 and when `x` is 11 and `y` is 9. Note that the compiler ignores the indentation in a Java program. Also, the Java compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{}`). On first glance, the programmer may not be sure which `if` an `else` matches; this situation is referred to as the "dangling-`else` problem." We have eliminated the indentation from the following code to make the problem more challenging. [*Hint*: Apply indentation conventions you have learned.]

a)
```
if ( x < 10 )
   if ( y > 10 )
   System.out.println( "*****" );
   else
   System.out.println( "#####" );
   System.out.println( "$$$$$" );
```

ANS:

```
When:  x = 9, y = 11
*****
$$$$$

When:  x = 11, y = 9
$$$$$
```

b)
```
if ( x < 10 ) {
   if ( y > 10 )
   System.out.println( "*****" );
   }
   else {
   System.out.println( "#####" );
   System.out.println( "$$$$$" );
   }
```

ANS:

```
When:  x = 9, y = 11
*****

When:  x = 11, y = 9
#####
$$$$$
```

4.22 *(Another Dangling-`else` Problem)* Modify the given code to produce the output shown in each part of the problem. Use proper indentation techniques. Make no changes other than inserting braces and changing the indentation of the code. The compiler ignores indentation in a Java program. We have eliminated the indentation from the given code to make the problem more challenging. [*Note*: It is possible that no modification is necessary for some of the parts.]

```
if ( y == 8 )
if ( x == 5 )
System.out.println( "@@@@@" );
else
System.out.println( "#####" );
System.out.println( "$$$$$" );
System.out.println( "&&&&&" );
```

a) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else
      System.out.println( "#####" );
System.out.println( "$$$$$" );
System.out.println( "&&&&&" );
```

b) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else {
      System.out.println( "#####" );
      System.out.println( "$$$$$" );
      System.out.println( "&&&&&" );
   }
```

c) Assuming that x = 5 and y = 8, the following output is produced:

```
@@@@@
&&&&&
```

ANS:

```
if ( y == 8 )
   if ( x == 5 )
      System.out.println( "@@@@@" );
   else {
      System.out.println( "#####" );
      System.out.println( "$$$$$" );
   }
System.out.println( "&&&&&" );
```

d) Assuming that x = 5 and y = 7, the following output is produced. [*Note*: The last three output statements after the `else` are all part of a block.]

```
#####
$$$$$
&&&&&
```

ANS:

```
if ( y == 8 ) {
   if ( x == 5 )
      System.out.println( "@@@@@" );
}
else
   System.out.println( "#####" );

System.out.println( "$$$$$" );
System.out.println( "&&&&&" );
```

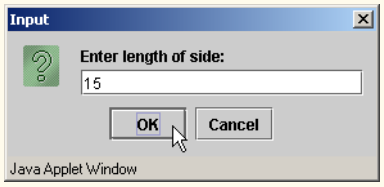
4.23 Write an applet that reads in the size of the side of a square and displays a hollow square of that size out of asterisks, by using the `drawString` method inside your applet's `paint` method. Use an input dialog to read the size from the user. Your program should work for squares of all side lengths between 1 and 20.

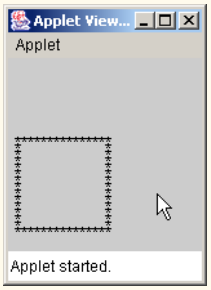
ANS:

```
1  // Exercise 4.23 Solution: Hollow.java
2  // Program prints a hollow square.
3  import java.awt.Graphics;
4  import javax.swing.*;
5
6  public class Hollow extends JApplet {
7     int stars;
8
9     // initializes applet by obtaining value from user
10    public void init()
11    {
12       String input;   // String entered by user
13
```

```
14          // read number from user as String
15          input = JOptionPane.showInputDialog( "Enter length of side:" );
16
17          // convert numbers from type String to type int
18          stars = Integer.parseInt( input );
19
20          if ( stars < 1 ) {
21             stars = 1;
22             JOptionPane.showMessageDialog( null,
23                "Invalid Input\nUsing default value 1",
24                "Error", JOptionPane.ERROR_MESSAGE );
25          }
26          if ( stars > 20 ) {
27             stars = 20;
28             JOptionPane.showMessageDialog( null,
29                "Invalid Input\nUsing default value 20",
30                "Error", JOptionPane.ERROR_MESSAGE );
31          }
32
33       }  // end method init
34
35       // draw Strings on applet's background
36       public void paint( Graphics g )
37       {
38          super.paint( g ); // call inherited version of method paint
39
40          int xCoordinate = 5, yCoordinate = 70, row = 1, column = 1;
41
42          // repeat for as many rows as the user entered
43          while ( row <= stars ) {
44
45             // and for as many columns as rows
46             while ( column <= stars ) {
47                if ( row == 1 )
48                   g.drawString( "*", xCoordinate, yCoordinate );
49
50                else if ( row == stars )
51                   g.drawString( "*", xCoordinate, yCoordinate );
52
53                else if ( column == 1 )
54                   g.drawString( "*", xCoordinate, yCoordinate );
55
56                else if ( column == stars )
57                   g.drawString( "*", xCoordinate, yCoordinate );
58
59                else
60                   g.drawString( " ", xCoordinate, yCoordinate );
61
62                xCoordinate += 5;
63                column++;
64             }
65
66             column = 1;
67             row++;
```

```
            yCoordinate += 5;
            xCoordinate = 5;

         }  // end while

      }  // end method paint

}  // end class Hollow
```



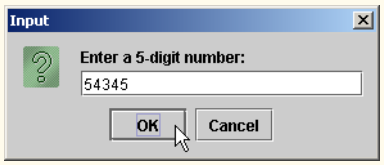


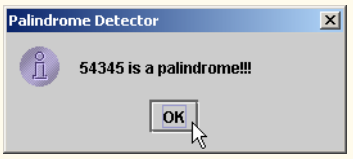
4.24 A palindrome is a sequence of characters that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message dialog indicating the problem to the user. When the user dismisses the error dialog, allow the user to enter a new value.

ANS:

```
// Exercise 4.24 Solution: Palindrome.java
// Program tests for a palindrome
import java.awt.*;
import javax.swing.JOptionPane;

public class Palindrome {

   public static void main( String args[] )
   {
      String resultString;    // result String
      int number,             // user input number
          digit1,             // first digit
          digit2,             // second digit
          digit4,             // fourth digit
          digit5,             // fifth digit
          digits;             // number of digits in input

      number = 0;
      digits = 0;

      while ( digits != 5 ) {
         number = Integer.parseInt( JOptionPane.showInputDialog(
            "Enter a 5-digit number: " ) );

```

```
25            if ( number < 100000 ) {
26
27               if ( number > 9999 )
28                  digits = 5;
29               else
30                  JOptionPane.showMessageDialog( null,
31                     "Number must be 5 digits", "Error",
32                     JOptionPane.INFORMATION_MESSAGE );
33            }
34            else
35               JOptionPane.showMessageDialog( null,
36                  "Number must be 5 digits", "Error",
37                  JOptionPane.INFORMATION_MESSAGE );
38         }
39
40         digit1 = number / 10000;
41         digit2 = number % 10000 / 1000;
42         digit4 = number % 10000 % 1000 % 100 / 10;
43         digit5 = number % 10000 % 1000 % 100 % 10;
44
45         if ( digit1 == digit5 ) {
46
47            if ( digit2 == digit4 )
48               resultString = number + " is a palindrome!!!";
49            else
50               resultString = number + " is not a palindrome.";
51         }
52         else
53            resultString = number + " is not a palindrome.";
54
55         JOptionPane.showMessageDialog( null, resultString,
56            "Palindrome Detector", JOptionPane.INFORMATION_MESSAGE );
57
58         System.exit( 0 );
59
60      }  // end method main
61
62   }  // end class Palindrome
```

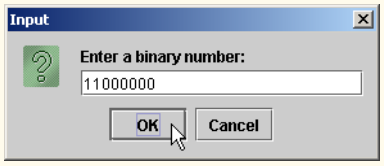


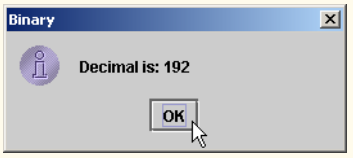


4.25 Write an application that inputs an integer containing only 0s and 1s (i.e., a "binary" integer) and prints its decimal equivalent. [*Hint*: Use the remainder and division operators to pick off the "binary number's" digits one at a time, from right to left. In the decimal number system, the rightmost digit has a positional value of 1 and the next digit to the left has a positional value of 10, then 100, then 1000, etc. The decimal number 234 can be interpreted as 4 * 1 + 3 * 10 + 2 * 100. In the binary number system, the rightmost digit has a positional value of 1, the next digit to the left has a positional value of 2, then 4, then 8, etc. The decimal equivalent of binary 1101 is 1 * 1 + 0 * 2 + 1 * 4 + 1 * 8, or 1 + 0 + 4 + 8 or, 13.]

ANS:

```
1  // Exercise 4.25 Solution: Binary.java
2  // Program prints the decimal equivalent of a binary number.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class Binary  {
7
8     public static void main( String args[] )
9     {
10       int binary,    // binary value
11           bit,       // bit positional value
12           decimal;   // decimal value
13
14       bit = 1;
15       decimal = 0;
16
17       binary = Integer.parseInt( JOptionPane.showInputDialog(
18          "Enter a binary number: " ) );
19
20       // convert to decimal equivalent
21       while ( binary != 0 ) {
22          decimal += binary % 10 * bit;
23          binary /= 10;
24          bit *= 2;
25       }
26
27       JOptionPane.showMessageDialog( null, "Decimal is: " + decimal,
28          "Binary", JOptionPane.INFORMATION_MESSAGE );
29
30       System.exit( 0 );
31
32    }  // end method main
33
34 }  // end class Binary
```





4.26 Write an application that uses only the output statements

```
System.out.print( "* " );
System.out.print( "  " );
System.out.println();
```

to display the checkerboard pattern that follows. Note that a `System.out.println` method call with no arguments causes the program to output a single newline character. [*Hint*: Repetition structures are required.]

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

ANS:

```
1  // Exercise 4.26 Solution: Stars.java
2  // Program prints a checkerboard pattern.
3  public class Stars {
4
5     public static void main( String args[] )
6     {
7        int row = 1;
8
9        while ( row <= 8 ) {
10          int column = 1;
11
12          if ( row % 2 == 0 )
13             System.out.print( " " );
14
15          while ( column <= 8 ) {
16             System.out.print( "* " );
17             column++;
18          }
19
20          System.out.println();
21          row++;
22       }
23
24    }  // end method main
25
26 }  // end class Stars
```

```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

4.27 Write an application that keeps displaying in the command window the multiples of the integer 2—namely, 2, 4, 8, 16, 32, 64, etc. Your loop should not terminate (i.e., create an infinite loop). What happens when you run this program?

ANS:

```
1  // Exercise 4.27 Solution: Infinite.java
2  // Program creates an infinite loop.
3
4  public class Infinite {
5
6     public static void main( String args[] )
7     {
8        int x = 1;
9
10       while ( true ) {
11          x *= 2;
12          System.out.println( x );
13       }
14    }
15
16 } // end class Infinite
```

```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
16384
32768
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
67108864
134217728
268435456
536870912
1073741824
-2147483648
0
0
0
```

4.28 What is wrong with the following statement? Provide the correct statement to add one to the sum of x and y.

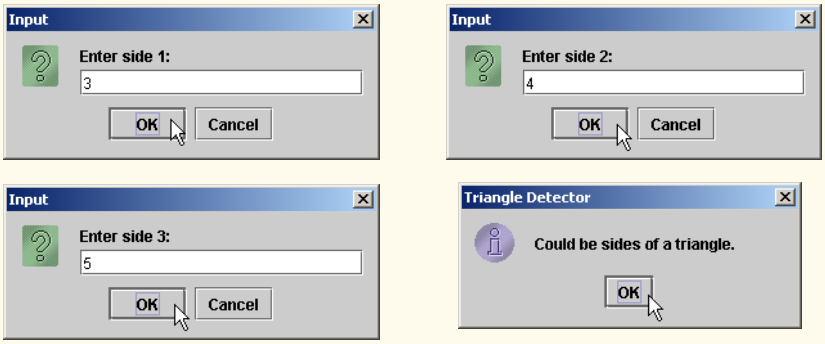
```
System.out.println( ++(x + y) );
```

ANS: ++ can only be applied to a variable—not a constant expression. The correct statement is `System.out.println( x + y + 1 );`

4.29 Write an application that reads three nonzero values entered by the user in input dialogs and determines whether and prints they could represent the sides of a triangle.

ANS:

```
1  // Exercise 4.29 Solution: Triangle1.java
2  // Program takes three values and determines if
3  // they form the sides of a triangle.
4  import java.awt.*;
5  import javax.swing.JOptionPane;
6
7  public class Triangle1  {
8
9     public static void main( String args[] )
10    {
11       String resultString;          // result String
12       double side1, side2, side3;   // three sides
13
14       // get values of three sides
15       side1 = Double.parseDouble(
16          JOptionPane.showInputDialog( "Enter side 1: " ) );
17
18       side2 = Double.parseDouble(
19          JOptionPane.showInputDialog( "Enter side 2: " ) );
20
21       side3 = Double.parseDouble(
22          JOptionPane.showInputDialog( "Enter side 3: " ) );
23
24       resultString = "These do not form a triangle.";
25
26       // triangle testing
27       if ( side1 + side2 > side3 ) {
28
29          if ( side2 + side3 > side1 ) {
30
31             if ( side3 + side1 > side2 )
32                resultString = "Could be sides of a triangle.";
33          }
34       }
35
36       JOptionPane.showMessageDialog( null, resultString,
37          "Triangle Detector", JOptionPane.INFORMATION_MESSAGE );
38
39       System.exit( 0 );
40
41    }  // end method main
42
43 }  // end class Triangle1
```

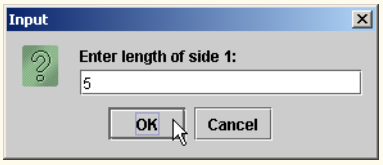


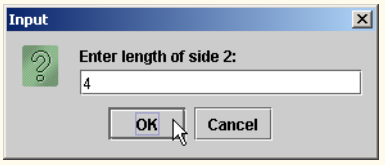
4.30 Write an application that reads three nonzero integers and determines whether and prints if they could represent the sides of a right triangle.

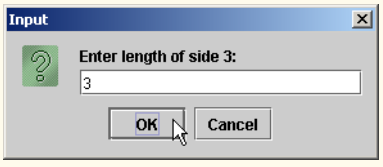
ANS:

```
1  // Exercise 4.30 Solution: Triangle2.java
2  // Program takes three integers and determines if they
3  // form the sides of a right triangle.
4  import javax.swing.JOptionPane;
5
6  public class Triangle2 {
7
8     public static void main( String args[] )
9     {
10       int side1,        // length of side 1
11           side2,        // length of side 2
12           side3;        // length of side 3
13       String input,     // user input
14              result;    // output String
15
16       // read number from user as String
17       input = JOptionPane.showInputDialog( "Enter length of side 1:" );
18
19       side1 = Integer.parseInt( input );
20
21       // read number from user as String
22       input = JOptionPane.showInputDialog( "Enter length of side 2:" );
23
24       side2 = Integer.parseInt( input );
25
26       // read number from user as String
27       input = JOptionPane.showInputDialog( "Enter length of side 3:" );
28
29       side3 = Integer.parseInt( input );
30
31       int side1Square = side1 * side1;
32       int side2Square = side2 * side2;
33       int side3Square = side3 * side3;
34
35       result = "These do not form a right triangle.";
```

```
36
37          if ( ( side1Square + side2Square ) == side3Square )
38             result = "These are the sides of a right triangle.";
39
40          if ( ( side1Square + side3Square ) == side2Square )
41             result = "These are the sides of a right triangle.";
42
43          if ( ( side2Square + side3Square ) == side1Square )
44             result = "These are the sides of a right triangle.";
45
46          JOptionPane.showMessageDialog( null, result, "Result",
47             JOptionPane.INFORMATION_MESSAGE );
48
49          System.exit( 0 );
50
51       }  // end method main
52
53    }  // end class Triangle2
```





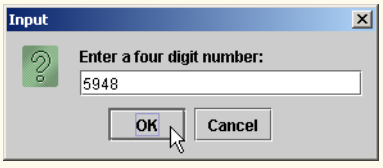


4.31 A company wants to transmit data over the telephone, but is concerned that its phones may be tapped. It has asked you to write a program that will encrypt its data so that the data may be transmitted more securely. All of its data is transmitted as four-digit integers. Your application should read a four-digit integer entered by the user in an input dialog and encrypt it as follows: Replace each digit with the result of adding 7 to the digit and getting the remainder after dividing the new value by 10. Then swap the first digit with the third, and swap the second digit with the fourth. Then print the encrypted integer. Write a separate application that inputs an encrypted four-digit integer and decrypts it to form the original number.

ANS:

```
1  // Exercise 4.31 Part A Solution: Encrypt.java
2  // Program encrypts a four digit number.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class Encrypt {
7
8     public static void main( String args[] )
9     {
10       int number,            // original number
```

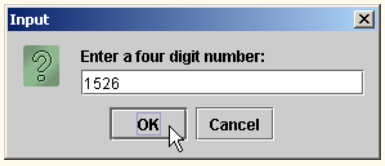
```
11            digit1,            // first digit
12            digit2,            // second digit
13            digit3,            // third digit
14            digit4,            // fourth digit
15            encryptedNumber;   // encrypted number
16
17         // enter four digit number to be encrypted
18         number = Integer.parseInt( JOptionPane.showInputDialog(
19            "Enter a four digit number: " ) );
20
21         // encrypt
22         digit1 = ( number / 1000 + 7 ) % 10;
23         digit2 = ( number % 1000 / 100 + 7 ) % 10;
24         digit3 = ( number % 100 / 10 + 7 ) % 10;
25         digit4 = ( number % 10 + 7 ) % 10;
26
27         encryptedNumber = digit1 * 10 + digit2 +
28            digit3 * 1000 + digit4 * 100;
29
30         JOptionPane.showMessageDialog( null,
31            "Encrypted number is " + encryptedNumber,
32            "Encryptor", JOptionPane.INFORMATION_MESSAGE );
33
34         System.exit( 0 );
35
36      }  // end method main
37
38   }  // end class Encrypt
```



ANS:

```
1  // Exercise 4.31 Part B Solution: Decrypt.java
2  // Program decrypts a four digit number.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class Decrypt {
7
8     public static void main( String args[] )
9     {
10       int number,    // encrypted number
11           digit1,            // first digit
12           digit2,            // second digit
13           digit3,            // third digit
14           digit4,            // fourth digit
15           decryptedNumber;   // decrypted number
```

```
16
17          // enter four digit number to be decrypted
18          number = Integer.parseInt( JOptionPane.showInputDialog(
19             "Enter a four digit number: " ) );
20
21          // decrypt
22          digit1 = ( number / 1000 + 3 ) % 10;
23          digit2 = ( number % 1000 / 100 + 3 ) % 10;
24          digit3 = ( number % 100 / 10 + 3 ) % 10;
25          digit4 = ( number % 10 + 3 ) % 10;
26
27          decryptedNumber = digit1 * 10 + digit2 +
28             digit3 * 1000 + digit4  * 100;
29
30          JOptionPane.showMessageDialog( null,
31             "Decrypted number is " + decryptedNumber,
32             "Decryptor", JOptionPane.INFORMATION_MESSAGE );
33
34          System.exit( 0 );
35
36       }  // end method main
37
38    }  // end class Decrypt
```



4.32 The factorial of a nonnegative integer n is written as $n!$ (pronounced "n factorial") and is defined as follows:

$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$ (for values of n greater than or equal to 1)

and

$n! = 1$ (for $n = 0$).

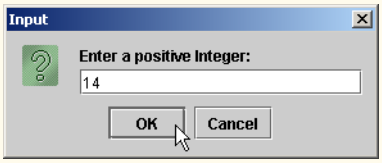
For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120.

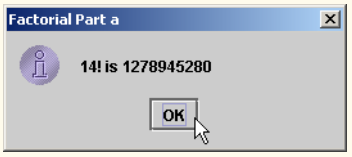
a) Write an application that reads a nonnegative integer from an input dialog and computes and prints its factorial.

ANS:

```
1  // Exercise 4.32 Part A Solution: Factorial.java
2  // Program calculates a factorial.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class Factorial {
7
8     public static void main( String args[] )
9     {
10       String input;
```

```
11         int number,      // user input
12             factorial;   // factorial of input value
13
14         factorial = 1;
15
16         input = JOptionPane.showInputDialog( "Enter a positive Integer: " );
17         number = Integer.parseInt( input );
18
19         // calculate factorial
20         while ( number > 0 ) {
21            factorial *= number;
22            number--;
23         }
24
25         JOptionPane.showMessageDialog( null, input + "! is " + factorial,
26            "Factorial Part a", JOptionPane.INFORMATION_MESSAGE );
27
28         System.exit( 0 );
29
30      }  // end method main
31
32   }  // end class Factorial
```





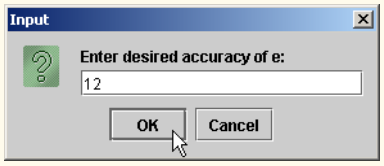
b) Write an application that estimates the value of the mathematical constant *e* by using the formula

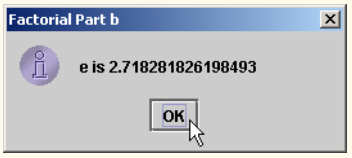
$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

ANS:

```
1  // Exercise 4.32 Part B Solution: E.java
2  // Program calculates estimate value of e.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class E {
7
8     public static void main( String args[] )
9     {
10       int number,      // counter
11           accuracy,    // accuracy of estimate
12           factorial;   // value of factorial
13       double e;        // estimate value of e
14
```

```
15          number = 1;
16          factorial = 1;
17          e = 1.0;
18
19          accuracy = Integer.parseInt( JOptionPane.showInputDialog(
20             "Enter desired accuracy of e: " ) );
21
22          // calculate estimation
23          while ( number < accuracy ) {
24             factorial *= number;
25             e += 1.0 / factorial;
26             number++;
27          }
28
29          JOptionPane.showMessageDialog( null, "e is " + e,
30             "Factorial Part b", JOptionPane.INFORMATION_MESSAGE );
31
32          System.exit( 0 );
33
34       }  // end method main
35
36    }  // end class E
```





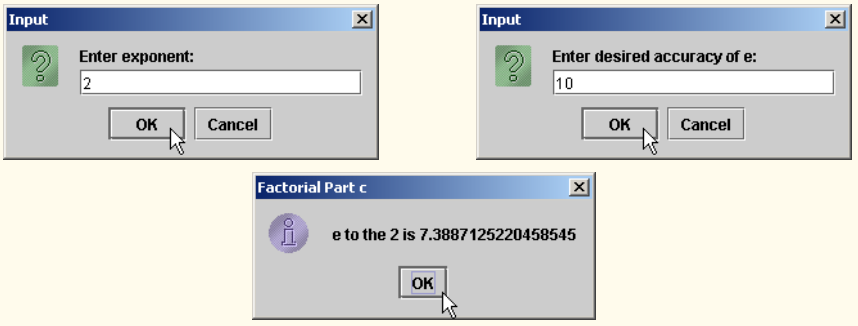
c) Write an application that computes the value of e^x by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

ANS:

```
1  // Exercise 4.32 Part C Solution: EtoX.java
2  // Program calculates e raised to x.
3  import java.awt.*;
4  import javax.swing.JOptionPane;
5
6  public class EtoX {
7
8     public static void main( String args[] )
9     {
10       int number,        // counter
11           accuracy,      // accuracy of estimate
12           factorial,     // value of factorial
13           x;             // x value
```

```
14          double e,           // estimate value of e
15                 exponent;    // exponent value
16
17          number = 1;
18          factorial = 1;
19          e = 1.0;
20          exponent = 1.0;
21
22          x = Integer.parseInt( JOptionPane.showInputDialog(
23             "Enter exponent: " ) );
24
25          accuracy = Integer.parseInt( JOptionPane.showInputDialog(
26             "Enter desired accuracy of e: " ) );
27
28          // calculate estimation
29          while ( number < accuracy ) {
30             exponent *= x;
31             factorial *= number;
32             e += exponent / factorial;
33             number++;
34          }
35
36          JOptionPane.showMessageDialog( null, " e to the " + x + " is " + e,
37             "Factorial Part c", JOptionPane.INFORMATION_MESSAGE );
38
39          System.exit( 0 );
40
41       }  // end method main
42
43    }  // end class EtoX
```



Control Structures: Part 2

Objectives

- To be able to use the `for` and `do…while` repetition statements to execute statements in a program repeatedly.
- To understand multiple selection using the `switch` selection statement.
- To be able to use the `break` and `continue` program control statements.
- To be able to use the logical operators.

Who can control his fate?
William Shakespeare

The used key is always bright.
Benjamin Franklin

Man is a tool-making animal.
Benjamin Franklin

Intelligence … is the faculty of making artificial objects, especially tools to make tools.
Henri Bergson

SELF-REVIEW EXERCISES

5.1 State whether each of the following is true or false. If false, explain why.

a) The `default` case is required in the `switch` selection statement.

ANS: False. The `default` case is optional. If no default action is needed, then there is no need for a `default` case.

b) The `break` statement is required in the last case of a `switch` selection statement.

ANS: False. The `break` statement is used to exit the `switch` statement. The `break` statement is not required for the last case in a `switch` statement.

c) The expression `( x > y && a < b )` is true if either `x > y` is true or `a < b` is true.

ANS: False. Both of the relational expressions must be true for the entire expression to be true when using the && operator.

d) An expression containing the || operator is true if either or both of its operands is true.

ANS: True.

5.2 Write a Java statement or a set of Java statements to accomplish each of the following tasks:

a) Sum the odd integers between 1 and 99, using a `for` statement. Assume that the integer variables `sum` and `count` have been declared.

ANS:
```
sum = 0;
for ( count = 1; count <= 99; count += 2 )
   sum += count;
```

b) Calculate the value of 2.5 raised to the power of 3, using the pow method.

ANS: `Math.pow( 2.5, 3 )`

c) Print the integers from 1 to 20, using a `while` loop and the counter variable `i`. Assume that the variable `i` has been declared, but not initialized. Print only five integers per line. [*Hint*: Use the calculation `i % 5`. When the value of this expression is 0, print a newline character; otherwise, print a tab character. Assume that this code is an application; use the `System.out.println()` method to output the newline character, and use the `System.out.print( '\t' )` method to output the tab character.]

ANS:
```
i = 1;

while ( i <= 20 ) {
System.out.print( i );

if ( i % 5 == 0 )
   System.out.println();
else
   System.out.print( '\t' );

++i;
}
```

d) Repeat Exercise 5.2 (c), using a `for` statement.

ANS:
```
for ( i = 1; i <= 20; i++ ) {
     System.out.print( i );

     if ( i % 5 == 0 )
        System.out.println();
     else
        System.out.print( '\t' );
 }
```

or

```
for ( i = 1; i <= 20; i++ )

   if ( i % 5 == 0 )
      System.out.println( i );
   else
      System.out.print( i + "\t" );
```

5.3 Find the error in each of the following code segments, and explain how to correct it:

a)
```
i = 1;

while ( i <= 10 );
   i++;
}
```

ANS: Error: The semicolon after the `while` header causes an infinite loop, and there is a missing left brace.
Correction: Replace the semicolon by a `{`, or remove both the `;` and the `}`.

b)
```
for ( k = 0.1; k != 1.0; k += 0.1 )
   System.out.println( k );
```

ANS: Error: Using a floating-point number to control a `for` statement may not work, because floating-point numbers are represented only approximately by most computers.
Correction: Use an integer, and perform the proper calculation in order to get the values you desire:

```
for ( k = 1; k != 10; k++ )
   System.out.println( ( float ) k / 10 );
```

c)
```
switch ( n ) {
   case 1:
      System.out.println( "The number is 1" );
   case 2:
      System.out.println( "The number is 2" );
      break;
   default:
      System.out.println( "The number is not 1 or 2" );
      break;
}
```

ANS: Error: The missing code is the `break` statement in the statements for the first `case`.
Correction: Add a `break` statement at the end of the statements for the first `case`. Note that this omission is not necessarily an error if the programmer wants the statement of `case 2:` to execute every time the `case 1:` statement executes.

d) The following code should print the values 1 to 10:

```
n = 1;

while ( n < 10 )
   System.out.println( n++ );
```

ANS: Error: An improper relational operator is used in the `while` repetition-continuation condition.
Correction: Use <= rather than <, or change 10 to 11.

EXERCISES

5.4 Find the error(s) in each of the following segments of code:

a)
```
For ( i = 100, i >= 1, i++ )
   System.out.println( i );
```

ANS: The F in `for` should be lowercase. Semicolons should be used in the `for` header instead of commas. ++ should be --.

b) The following code should print whether integer `value` is odd or even:

```
switch ( value % 2 ) {

   case 0:
      System.out.println( "Even integer" );

   case 1:
      System.out.println( "Odd integer" );
}
```

ANS: A `break` statement should be placed in `case 0`.

c) The following code should output the odd integers from 19 to 1:

```
for ( i = 19; i >= 1; i += 2 )
   System.out.println( i );
```

ANS: += should be -=.

d) The following code should output the even integers from 2 to 100:

```
counter = 2;

do {
   System.out.println( counter );
   counter += 2;
} While ( counter < 100 );
```

ANS: The W in `while` should be lowercase. < should be <=.

5.5 What does the following program do?

```
1  public class Printing {
2
3     public static void main( String args[] )
4     {
5        for ( int i = 1; i <= 10; i++ ) {
6
7           for ( int j = 1; j <= 5; j++ )
8              System.out.print( '@' );
9
10          System.out.println();
11
12       } // end outer for
13
14    } // end method main
15
16 } // end class Printing
```

ANS:

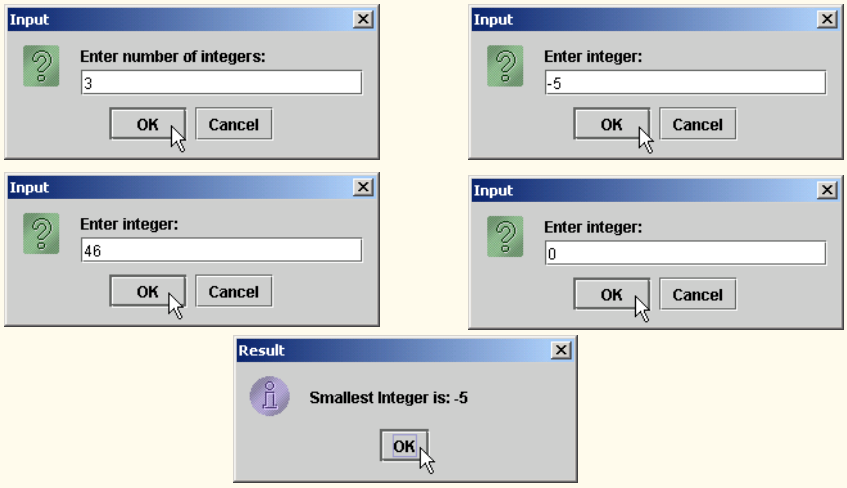
```
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
@@@@@
```

5.6 Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

ANS:

```
1  // Exercise 5.6 Solution: Small.java
2  // Program finds the smallest of several integers.
3
4  import javax.swing.JOptionPane;
5
6  public class Small {
7
8     // main method begins execution of Java application
9     public static void main( String args[] )
10    {
11       int smallest = 0,   // smallest number
12           number = 0,     // number entered by user
13           integers;       // number of integers
14       String input;       // user input
15
16       input = JOptionPane.showInputDialog( "Enter number of integers:" );
17       integers = Integer.parseInt( input );
18
19       for ( int counter = 1; counter <= integers; counter++ ) {
20          input = JOptionPane.showInputDialog( "Enter integer:" );
21          number = Integer.parseInt( input );
22
23          if ( counter == 1 )
24             smallest = number;
25
26          else if ( number < smallest )
27             smallest = number;
28
29       } // end for loop
30
31       JOptionPane.showMessageDialog( null, "Smallest Integer is: " +
32          smallest, "Result", JOptionPane.INFORMATION_MESSAGE );
33
34       System.exit( 0 );
```

```
35
36       }  // end of main method
37
38    }  // end of class Small
```



5.7 Write an application that calculates the product of the odd integers from 1 to 15 and displays the results in a message dialog.

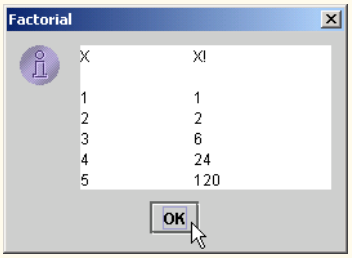
ANS:

```
1  // Exercise 5.7 Solution: Odd.java
2  // Program prints the product of the odd integers from 1 to 15
3
4  import javax.swing.JOptionPane;
5
6  public class Odd {
7
8     public static void main( String args[] )
9     {
10       int product = 1;
11
12       // loop through all odd numbers from 3 to 15
13       for ( int x = 3; x <= 15; x += 2 )
14          product *= x;
15
16       // show results
17       JOptionPane.showMessageDialog( null, "Product is " + product,
18          "Odd Product", JOptionPane.INFORMATION_MESSAGE );
19
20       System.exit( 0 );
21    }
22
23 }  // end class Odd
```

5.8 *Factorials* are used frequently in probability problems. The factorial of a positive integer *n* (written *n!* and pronounced "*n* factorial") is equal to the product of the positive integers from 1 to *n*. Write an application that evaluates the factorials of the integers from 1 to 5. Display the results in tabular format in a `JTextArea` that is displayed on a message dialog. What difficulty might prevent you from calculating the factorial of 20?

ANS:

```
1  // Exercise 5.8 Solution: Factorial.java
2  // Program calculates factorials.
3
4  import javax.swing.*;
5
6  public class Factorial {
7
8     // main method begins execution of Java program
9     public static void main( String args[] )
10    {
11       JTextArea outputArea = new JTextArea( 5, 10 );
12       String outputString = "X\tX!\n";
13
14       for ( int number = 1; number <= 5; number++ ) {
15          int factorial = 1;
16
17          for ( int smaller = 1; smaller <= number; smaller++ )
18             factorial *= smaller;
19
20          outputString += "\n" + number + "\t" + factorial;
21       }
22
23       outputArea.setText( outputString );
24       JOptionPane.showMessageDialog( null, outputArea,
25          "Factorial", JOptionPane.INFORMATION_MESSAGE );
26
27       System.exit( 0 );
28    }
29
30 } // end class Factorial
```

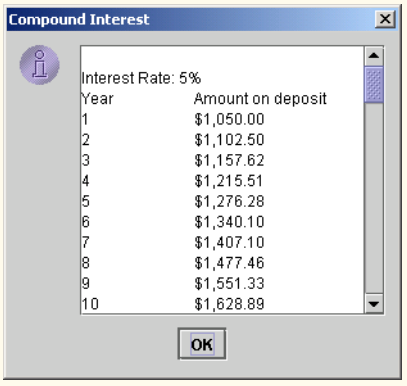


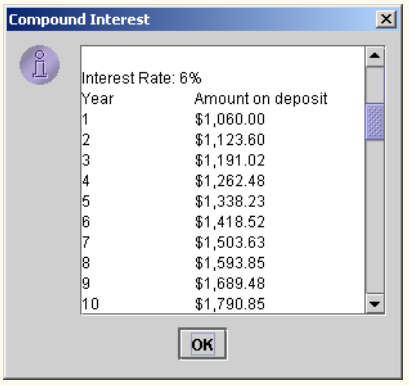
5.9 Modify the compound-interest application of Fig. 5.6 to repeat its steps for interest rates of 5, 6, 7, 8, 9 and 10%. Use a `for` loop to vary the interest rate. Add scrolling functionality to the `JTextArea`, so you can scroll through all the output.

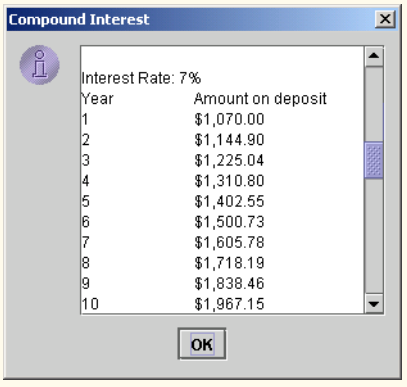
ANS:

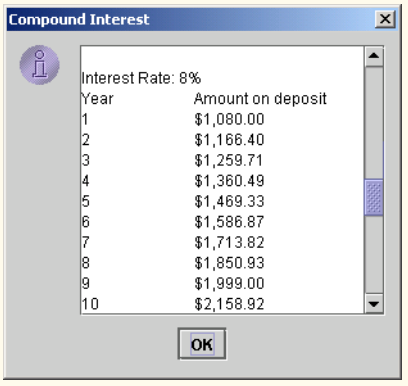
```
1  // Exercise 5.9 Solution: Interest.java
2  // Calculating compound interest
3
4  import java.text.NumberFormat;
5  import java.util.Locale;
6  import javax.swing.*;
7
8  public class Interest {
9
10    public static void main( String args[] )
11    {
12       // create decimal format to format floating point numbers
13       // with two digits to the right of the decimal point
14       NumberFormat moneyFormat =
15          NumberFormat.getCurrencyInstance( Locale.US );
16
17       // create a scrolled pane for the output
18       JTextArea outputTextArea = new JTextArea( 13, 20 );
19       JScrollPane scroller = new JScrollPane( outputTextArea );
20
21       //initial deposit.
22       double principal = 1000.0;
23
24       // print out statistics for each rate
25       for ( int interestRate = 5; interestRate <= 10; interestRate++ ) {
26          double rate = interestRate / 100.0;
27          outputTextArea.append( "\nInterest Rate: " +
28             interestRate + "%\n" );
29          outputTextArea.append( "Year\tAmount on deposit\n" );
30
31          // for each rate, print a ten year forcast
32          for ( int year = 1; year <= 10; year++ ) {
33             double amount = principal * ( 1.0 + rate );
34
35             // raise the amount to the power of the year
36             for ( int power = 2; power <= year; power++ )
37                amount *= ( 1.0 + rate );
```

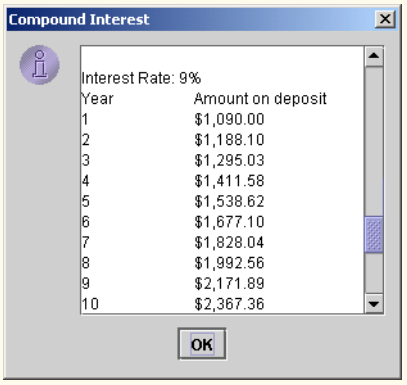
```
38
39             outputTextArea.append( year + "\t" +
40                moneyFormat.format( amount ) + "\n" );
41          }
42       }
43
44       // show result
45       JOptionPane.showMessageDialog( null, scroller,
46          "Compound Interest", JOptionPane.INFORMATION_MESSAGE );
47
48       // terminate the application
49       System.exit( 0 );
50    }
51
52 }  // end class Interest
```

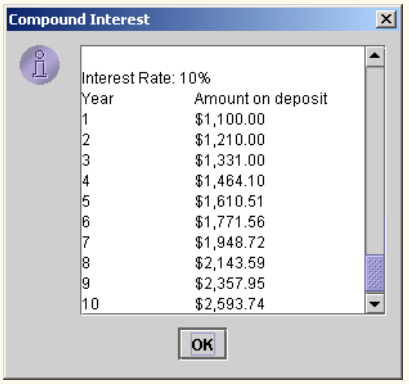












5.10 Write an application that displays the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (*) should be printed by a single statement of the form `System.out.print( '*' );` which causes the asterisks to print side by side. A statement of the form `System.out.println();` can be used to position to the next line. A statement of the form `System.out.print( ' ' );` can be used to display a space for the last two patterns. There should be no other output statements in the program. [*Hint*: The last two patterns require that each line begin with an appropriate number of blank spaces.]

(a)

```
*
**
***
****
*****
******
*******
********
*********
**********
```

(b)

```
**********
*********
********
*******
******
*****
****
***
**
*
```

(c)

```
**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *
```

(d)

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

ANS:

```
1  // Exercise 5.10 Solution: Triangles.java
2  // Program prints four triangles, one below the other
3
4  public class Triangles {
5
6     public static void main( String args[] )
7     {
8        int row, column, space;
9
10       // first triangle
11       for ( row = 1; row <= 10; row++ ) {
12
```

```
13            for ( column = 1; column <= row; column++ )
14               System.out.print( '*' );
15
16            System.out.println();
17         }
18
19         System.out.println();
20
21         // second triangle
22         for ( row = 10; row >= 1; row-- ) {
23
24            for ( column = 1; column <= row; column++ )
25               System.out.print( '*' );
26
27            System.out.println();
28         }
29
30         System.out.println();
31
32         // third triangle
33         for ( row = 10; row >= 1; row-- ) {
34
35            for ( space = 10; space > row; space-- )
36               System.out.print( ' ' );
37
38            for ( column = 1; column <= row; column++ )
39               System.out.print( '*' );
40
41            System.out.println();
42         }
43
44         System.out.println();
45
46         // fourth triangle
47         for ( row = 10; row >= 1; row-- ) {
48
49            for ( space = 1; space < row; space++ )
50               System.out.print( ' ' );
51
52            for ( column = 10; column >= row; column-- )
53               System.out.print( '*' );
54
55            System.out.println();
56         }
57
58      } // end method main
59
60   }  // end class Triangles
```

```
*
**
***
****
*****
******
*******
********
*********
**********

**********
*********
********
*******
******
*****
****
***
**
*

**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *

         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

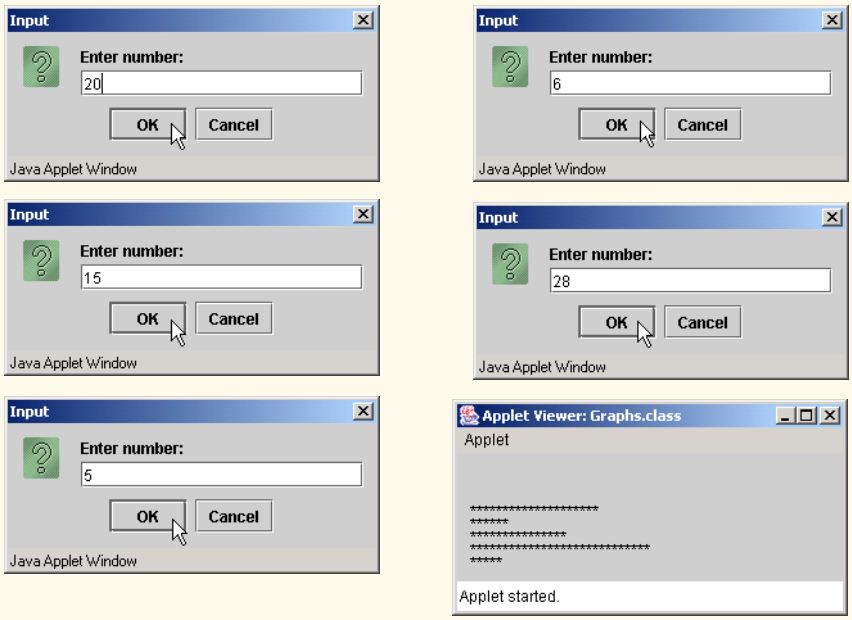
5.11 One interesting application of computers is drawing graphs and bar charts (sometimes called "histograms"). Write an applet that reads five numbers, each between 1 and 30. For each number read, your program should draw that number of adjacent asterisks. For example, if your program reads the number 7, it should display *******. [*Hint:* All asterisks on one line should be drawn with the same *y*-coordinate.]

ANS:

```
1  // Exercise 5.11 Solution: Graphs.java
2  // Program prints 5 histograms with lengths determined by user.
3
4  import java.awt.Graphics;
5  import javax.swing.*;
6
7  public class Graphs extends JApplet {
8     int number1, number2, number3, number4, number5;
9
10    // initialize applet by obtaining values from user
11    public void init()
12    {
13       String inputString;
14       int inputNumber, counter = 1;
15
16       while ( counter <= 5) {
17          inputString = JOptionPane.showInputDialog( "Enter number:" );
18          inputNumber = Integer.parseInt( inputString );
19
20          // define appropriate num if input is between 1-30
21          if ( inputNumber >= 1 && inputNumber <= 30 ) {
22
23             switch ( counter ) {
24
25                case 1:
26                   number1 = inputNumber;
27                   break;   // done processing case
28
29                case 2:
30                   number2 = inputNumber;
31                   break;   // done processing case
32
33                case 3:
34                   number3 = inputNumber;
35                   break;   // done processing case
36
37                case 4:
38                   number4 = inputNumber;
39                   break;   // done processing case
40
41                case 5:
42                   number5 = inputNumber;
43                   break;   // done processing case
44             }
45
46             counter++;
```

```
47
48          }  // end if
49
50          else
51             JOptionPane.showMessageDialog( null,
52                "Invalid Input\nNumber should be between 1 and 30",
53                "Error", JOptionPane.ERROR_MESSAGE );
54
55       }  // end while
56
57    }  // end method init
58
59    // draw histograms on applet's background
60    public void paint( Graphics g )
61    {
62       // call inherited version of method paint
63       super.paint( g );
64
65       int xCoordinate, yCoordinate = 0, value = 0;
66
67       // print histograms
68       for ( int counter = 1; counter <= 5; counter++ ) {
69
70          switch ( counter ) {
71
72             case 1:
73                value = number1;
74                break;   // done processing case
75
76             case 2:
77                value = number2;
78                break;   // done processing case
79
80             case 3:
81                value = number3;
82                break;   // done processing case
83
84             case 4:
85                value = number4;
86                break;   // done processing case
87
88             case 5:
89                value = number5;
90                break;   // done processing case
91          }
92
93          xCoordinate = 5;
94          yCoordinate = counter * 10 + 40;
95
96          for ( int j = 1; j <= value; j++ )
97             g.drawString( "*", xCoordinate += 5, yCoordinate );
98       }
99
100   }  // end method paint
```

```
101
102 } // end class Graphs
```



5.12 A mail-order house sells five products whose retail prices are as follows: Product 1, $2.98; product 2, $4.50; product 3, $9.98; product 4, $4.49 and product 5, $6.87. Write an application that reads a series of pairs of numbers as follows:

a) product number;
b) quantity sold.

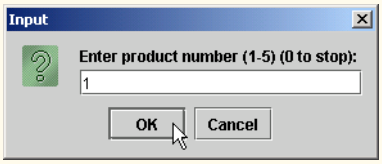
Your program should use a `switch` statement to determine the retail price for each product. It should calculate and display the total retail value of all products sold. Use a sentinel-controlled loop to determine when the program should stop looping and display the final results

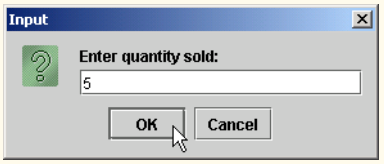
ANS:

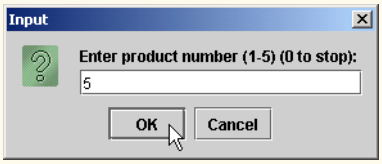
```
1  // Exercise 5.12 Solution: Sales.java
2  // Program calculates sales, based on an input of product
3  // number and quantity sold
4
5  import java.awt.*;
6  import java.text.NumberFormat;
7  import java.util.Locale;
8  import javax.swing.*;
9
10 public class Sales {
11
12    public static void main( String args[] )
13    {
14       float product1 = 0, product2 = 0, product3 = 0;
```

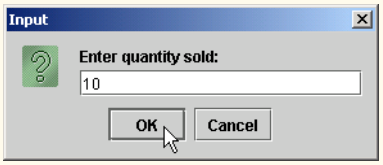
```
15         float product4 = 0, product5 = 0;
16
17         String inputString;
18
19         int productId = 1;
20
21         // ask user for product number until flag value entered
22         while (productId != 0) {
23
24            // determine the product chosen
25            inputString = JOptionPane.showInputDialog(
26               "Enter product number (1-5) (0 to stop):" );
27            productId = Integer.parseInt( inputString );
28
29            if ( productId >= 1 && productId <= 5 ) {
30
31               // determine the number sold of the item
32               inputString = JOptionPane.showInputDialog(
33                  "Enter quantity sold:" );
34               int quantity = Integer.parseInt( inputString );
35
36               // increment the total for the item by the
37               // price times the quantity sold
38               switch ( productId ) {
39
40                  case 1:
41                     product1 += quantity * 2.98;
42                     break;
43
44                  case 2:
45                     product2 += quantity * 4.50;
46                     break;
47
48                  case 3:
49                     product3 += quantity * 9.98;
50                     break;
51
52                  case 4:
53                     product4 += quantity * 4.49;
54                     break;
55
56                  case 5:
57                     product5 += quantity * 6.87;
58                     break;
59               }
60
61            }
62
63            else if ( productId != 0 ) {
64               JOptionPane.showMessageDialog( null,
65                  "Product number must be between 1 and 5 or 0 to stop",
66                  "Input Error", JOptionPane.INFORMATION_MESSAGE );
67            }
68         }
```

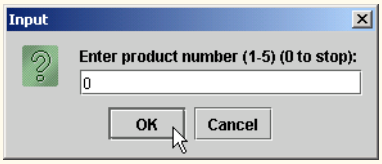
```
69
70          // create decimal format to format floating point numbers
71          // with two digits to the right of the decimal point
72          NumberFormat moneyFormat =
73             NumberFormat.getCurrencyInstance( Locale.US );
74
75          // create a summary message
76          String output  = "Product 1: " + moneyFormat.format( product1 );
77          output += "\nProduct 2: "+ moneyFormat.format( product2 );
78          output += "\nProduct 3: " + moneyFormat.format( product3 );
79          output += "\nProduct 4: " + moneyFormat.format( product4 );
80          output += "\nProduct 5: " + moneyFormat.format( product5 ) + "\n";
81
82          JTextArea outputArea = new JTextArea( 6, 20 );
83          outputArea.setText( output );
84
85          // show results
86          JOptionPane.showMessageDialog( null, outputArea,
87             "Totals", JOptionPane.INFORMATION_MESSAGE );
88
89          System.exit( 0 );
90       }
91
92    }  // end class Sales
```

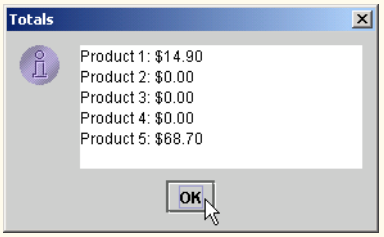








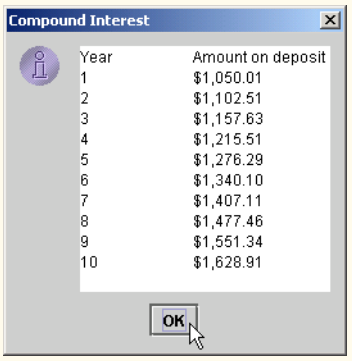




5.13 Modify the application in Fig. 5.6 to use only integers to calculate the compound interest. [*Hint*: Treat all monetary amounts as integral numbers of pennies. Then "break" the result into its dollar portion and cents portion by using the division and remainder operations, respectively. Insert a period between the dollar and the cents portions.]

ANS:

```
1  // Exercise 5.13: Interest.java
2  // Calculating compound interest.
3
4  import java.text.NumberFormat;
5  import java.util.Locale;
6  import javax.swing.JOptionPane;
7  import javax.swing.JTextArea;
8
9  public class Interest {
10
11    public static void main( String args[] )
12    {
13       int amount;       // amount on deposit at end of each year
14       int principal = 1000;  // initial amount before interest
15       double rate = 0.05;    // interest rate
16
17       // create NumberFormat for currency in US dollar format
18       NumberFormat moneyFormat =
19          NumberFormat.getCurrencyInstance( Locale.US );
20
21       // create JTextArea to display output
22       JTextArea outputTextArea = new JTextArea();
23
24       // set first line of text in outputTextArea
25       outputTextArea.setText( "Year\tAmount on deposit\n" );
26
27       // calculate amount on deposit for each of ten years
28       for ( int year = 1; year <= 10; year++ ) {
29
30          // calculate new amount for specified year
31          amount =
32             ( int ) ( principal * 100 * Math.pow( 1.0 + rate, year ) );
33
34          // append one line of text to outputTextArea
35          outputTextArea.append( year + "\t" + moneyFormat.format(
36             ( amount + Math.pow( 1.0 + rate, year) ) / 100 ) + "\n" );
37
38       } // end for
39
40       // display results
41       JOptionPane.showMessageDialog( null, outputTextArea,
42          "Compound Interest", JOptionPane.INFORMATION_MESSAGE );
43
44       System.exit( 0 );  // terminate the application
45
46    } // end main
47
48 } // end class Interest
```



5.14 Assume that `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `System.out.println( i == 1 );`
ANS: True.

b) `System.out.println( j == 3 );`
ANS: False.

c) `System.out.println( i >= 1 && j < 4 );`
ANS: True.

d) `System.out.println( m <= 99 & k < m );`
ANS: False.

e) `System.out.println( j >= i || k == m );`
ANS: True.

f) `System.out.println( k + m < j | 3 - j >= k );`
ANS: False.

g) `System.out.println( !( k > m ) );`
ANS: False.

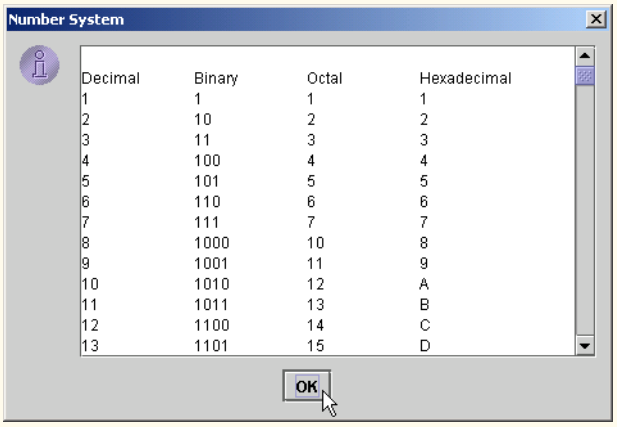
5.15 Write an application that prints a table of the binary, octal, and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you are not familiar with these number systems, read Appendix C first. Place the results in a `JTextArea` with scrolling functionality. Display the `JTextArea` in a message dialog.

ANS:

```
1  // Exercise 5.15 Solution: NumberSystem.java
2  // Converting a decimal number to binary, octal and hexadecimal.
3
4  import javax.swing.*;
5
```

```
6  public class NumberSystem {
7
8     public static void main( String args[] )
9     {
10       // create a scrolled pane for the output
11       JTextArea outputTextArea = new JTextArea( 15, 35 );
12       JScrollPane scroller = new JScrollPane( outputTextArea );
13
14       outputTextArea.append( "\nDecimal\tBinary\tOctal\tHexadecimal\n" );
15
16       // print out binary, octal and hexadecimal representation
17       // for each number
18       for ( int i = 1; i <= 256; i++ ) {
19          int decimal = i;
20          String binary = "", octal = "", hexadecimal = "";
21          int temp;
22
23          // get binary representation
24          while ( decimal >= 2 ) {
25             temp = decimal % 2;
26             binary = temp + binary;
27             decimal /= 2;
28          }
29          binary = decimal + binary;
30
31          decimal = i;
32
33          // get octal representation
34          while ( decimal >= 8 ) {
35             temp = decimal % 8;
36             octal = temp + octal;
37             decimal /= 8;
38          }
39          octal = decimal + octal;
40
41          decimal = i;
42
43          // get hexadecimal representation
44          while ( decimal >= 16 ) {
45             temp = decimal % 16;
46
47             switch ( temp ) {
48                case 10:
49                   hexadecimal = "A" + hexadecimal;
50                   break;
51                case 11:
52                   hexadecimal = "B" + hexadecimal;
53                   break;
54                case 12:
55                   hexadecimal = "C" + hexadecimal;
56                   break;
57                case 13:
58                   hexadecimal = "D" + hexadecimal;
59                   break;
```

```
60                case 14:
61                   hexadecimal = "E" + hexadecimal;
62                   break;
63                case 15:
64                   hexadecimal = "F" + hexadecimal;
65                   break;
66                default:
67                   hexadecimal = temp + hexadecimal;
68                   break;
69             }
70
71             decimal /= 16;
72          }
73
74          switch ( decimal ) {
75             case 10:
76                hexadecimal = "A" + hexadecimal;
77                break;
78             case 11:
79                hexadecimal = "B" + hexadecimal;
80                break;
81             case 12:
82                hexadecimal = "C" + hexadecimal;
83                break;
84             case 13:
85                hexadecimal = "D" + hexadecimal;
86                break;
87             case 14:
88                hexadecimal = "E" + hexadecimal;
89                break;
90             case 15:
91                hexadecimal = "F" + hexadecimal;
92                break;
93             default:
94                hexadecimal = decimal + hexadecimal;
95                break;
96          }
97
98          outputTextArea.append( i + "\t" + binary + "\t" +
99             octal + "\t" + hexadecimal + "\n" );
100       }
101
102       // show result
103       JOptionPane.showMessageDialog( null, scroller,
104          "Number System", JOptionPane.INFORMATION_MESSAGE );
105
106       // terminate the application
107       System.exit( 0 );
108    }
109
110 }  // end class NumberSystem
```



5.16 Calculate the value of π from the infinite series

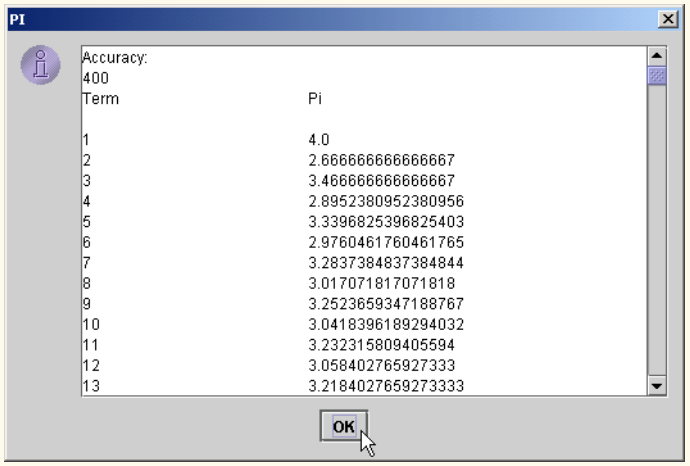
$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by computing one term of this series, by two terms, by three terms, etc. How many terms of this series do you have to use before you first get 3.14? 3.141? 3.1415? 3.14159?

ANS:

```
1  // Execise 5.16 Solution: Pi.java
2  // Program calculates Pi from the infinite series.
3
4  import javax.swing.*;
5
6  public class Pi {
7
8     //method main begins execution of Java application
9     public static void main( String args[] )
10    {
11       double piValue = 0, number = 4.0, denominator = 1.0;
12       int accuracy = 400;
13
14       JTextArea outputArea = new JTextArea( 17, 40 );
15       JScrollPane scroller = new JScrollPane( outputArea );
16
17       String output = "Accuracy: \n" + accuracy;
18       output += "\nTerm\t\tPi\n";
19
20       for ( int term = 1; term <= accuracy; term++ ) {
21
22          if ( term % 2 != 0 )
23             piValue += number / denominator;
24
25          else
26             piValue -= number / denominator;
```

```
27
28             output += "\n" + term + "\t\t" + piValue;
29             denominator += 2.0;
30          }
31
32          outputArea.setText( output );
33
34          JOptionPane.showMessageDialog( null, scroller, "PI",
35             JOptionPane.INFORMATION_MESSAGE );
36
37          System.exit( 0 );
38
39       }  // end method main
40
41    }  // end class Pi
```



5.17 (*Pythagorean Triples*) A right triangle can have sides whose lengths are all integers. The set of three integer values for the lengths of the sides of a right triangle is called a Pythagorean triple. The lengths of the three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Write an application to find all Pythagorean triples for `side1`, `side2` and the `hypotenuse`, all no larger than 500. Use a triple-nested `for` loop that tries all possibilities. This method is an example of "brute force" computing. You will learn in more advanced computer science courses that there are large numbers of interesting problems for which there is no known algorithmic approach other than using sheer brute force.

ANS:

```
1  // Exercise 5.17 Solution: Triples.java
2  // Program calculates Pythagorean triples
3
4  public class Triples {
5
6     public static void main( String args[] )
7     {
```

```
 8         // declare the three sides of a triangle
 9         int side1, side2, hypotenuse;
10
11         for ( side1 = 1; side1 <= 500; side1++ )
12
13            for ( side2 = 1; side2 <= 500; side2++ )
14
15               for ( hypotenuse = 1; hypotenuse <= 500; hypotenuse++ )
16
17                  // use Pythagorean Theorem to print right triangles
18                  if ( ( side1 * side1 ) + ( side2 * side2 ) ==
19                     ( hypotenuse * hypotenuse ) )
20
21                     System.out.println( "s1: " + side1 + " s2: "
22                        + side2 + "  h: " + hypotenuse );
23      }
24
25   }  // end class Triples
```

```
s1: 3 s2: 4  h: 5
s1: 4 s2: 3  h: 5
s1: 5 s2: 12  h: 13
s1: 6 s2: 8  h: 10
s1: 7 s2: 24  h: 25
s1: 8 s2: 6  h: 10
s1: 8 s2: 15  h: 17
s1: 9 s2: 12  h: 15
s1: 9 s2: 40  h: 41
s1: 10 s2: 24  h: 26

...

s1: 456 s2: 190  h: 494
s1: 468 s2: 155  h: 493
s1: 468 s2: 176  h: 500
s1: 475 s2: 132  h: 493
s1: 476 s2: 93  h: 485
s1: 480 s2: 31  h: 481
s1: 480 s2: 88  h: 488
s1: 480 s2: 108  h: 492
s1: 480 s2: 140  h: 500
s1: 483 s2: 44  h: 485
```

5.18 Modify Exercise 5.10 to combine your code from the four separate triangles of asterisks such that all four patterns print side by side. Make clever use of nested `for` loops.

```
*           **********  **********          *
**          *********    *********         **
***         ********      ********        ***
****        *******        *******       ****
*****       ******          ******      *****
******      *****            *****     ******
*******     ****              ****    *******
********    ***                ***   ********
*********   *                   **  *********
********** *                     * **********
```

ANS:

```
1  // Exercise 5.18 Solution: Triangles2.java
2  // Program prints four triangles side by side
3
4  public class Triangles2 {
5
6     public static void main( String args[] )
7     {
8        int row, column, space;
9
10       // print one row at a time, tabbing between triangles
11       for ( row = 1; row <= 10; row++ ) {
12
13          // triangle one
14          for ( column = 1; column <= row; column++ )
15             System.out.print( '*' );
16
17          for ( space = 1; space <= 10 - row; space++ )
18             System.out.print( ' ' );
19
20          System.out.print( "\t" );
21
22          // triangle two
23          for ( column = 10; column >= row; column-- )
24             System.out.print( '*' );
25
26          for ( space = 1; space < row; space++ )
27             System.out.print( ' ' );
28
29          System.out.print( "\t" );
30
31          // triangle four
32          for ( space = 1; space < row; space++ )
33             System.out.print( ' ' );
34
35          for ( column = 10; column >= row; column-- )
36             System.out.print( '*' );
```

```
37
38             System.out.print( "\t" );
39
40             // triangle three
41             for ( space = 10; space > row; space-- )
42                System.out.print( ' ' );
43
44             for ( column = 1; column <= row; column++ )
45                System.out.print( '*' );
46
47             System.out.println();
48
49          }
50       }
51
52    }  // end class Triangles2
```

```
*                  **********       **********                  *
**                 *********         *********                 **
***                ********           ********                ***
****               *******             *******               ****
*****              ******               ******              *****
******             *****                 *****             ******
*******            ****                   ****            *******
********           ***                     ***           ********
*********          **                       **          *********
**********         *                         *         **********
```

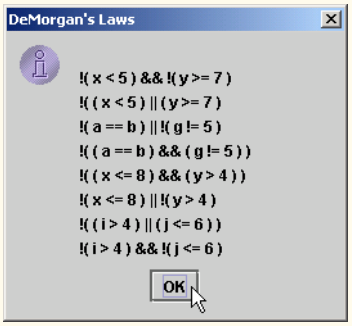
5.19 (*De Morgan's Laws*) In this chapter, we have discussed the logical operators &&, &, ||, |, ^ and !. De Morgan's Laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression !(*condition1* && *condition2*) is logically equivalent to the expression (!*condition1* || !*condition2*). Also, the expression !(*condition1* || *condition2*) is logically equivalent to the expression (!*condition1* && !*condition2*). Use De Morgan's Laws to write equivalent expressions for each of the following, then write an application to show that both the original expression and the new expression in each case produce the same value:

a) `!( x < 5 ) && !( y >= 7 )`
b) `!( a == b ) || !( g != 5 )`
c) `!( ( x <= 8 ) && ( y > 4 ) )`
d) `!( ( i > 4 ) || ( j <= 6 ) )`

ANS:

```
1  // Exercise 5.19 Solution: DeMorgan.java
2  // Program tests DeMorgan's laws.
3
4  import javax.swing.JOptionPane;
5
6  public class DeMorgan {
7
8     // method main begins execution of Java application
9     public static void main( String args[] )
10    {
```

```
11         String result = "";
12         int x = 6, y = 0;
13
14         // part a
15         if ( !( x < 5 ) && !( y >= 7 ) )
16            result += "\n!( x < 5 ) && !( y >= 7 )";
17
18         if ( !( ( x < 5 ) || ( y >= 7 ) ) )
19            result += "\n!( ( x < 5 ) || ( y >= 7 )";
20
21         int a = 8, b = 22, g = 88;
22
23         // part b
24         if ( !( a == b ) || !( g != 5 ) )
25            result += "\n!( a == b ) || !( g != 5 )";
26
27         if ( !( ( a == b ) && ( g != 5 ) ) )
28            result += "\n!( ( a == b ) && ( g != 5 ) )";
29
30         x = 8;
31         y = 2;
32
33         // part c
34         if ( !( ( x <= 8 ) && ( y > 4 ) ) )
35            result += "\n!( ( x <= 8 ) && ( y > 4 ) )";
36
37         if ( !( x <= 8 ) || !( y > 4 ) )
38            result += "\n!( x <= 8 ) || !( y > 4 )";
39
40         int i = 0, j = 7;
41
42         // part d
43         if ( !( ( i > 4 ) || ( j <= 6 ) ) )
44            result += "\n!( ( i > 4 ) || ( j <= 6 ) )";
45
46         if ( !( i > 4 ) && !( j <= 6 ) )
47            result += "\n!( i > 4 ) && !( j <= 6 )";
48
49         JOptionPane.showMessageDialog( null, result, "DeMorgan's Laws",
50            JOptionPane.INFORMATION_MESSAGE );
51
52         System.exit( 0 );
53
54      } // end method main
55
56   } // end class DeMorgan
```



5.20 Write an application that prints the following diamond shape. You may use output statements that print a single asterisk (*), a single space or a single newline character. Maximize your use of repetition (with nested `for` statements), and minimize the number of output statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

ANS:

```
1  // Exercise 5.20 Solution: Diamond.java
2  // Program prints a diamond with minimal output statements
3
4  public class Diamond {
5
6     public static void main( String args[] )
7     {
8        int row, stars, spaces;
9        String diamondString = "";
10
11       // top half (1st five lines)
12       for ( row = 1; row <= 5; row++ ) {
13
14          for ( spaces = 5; spaces > row; spaces-- )
15             diamondString += " " ;
16
17          for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
18             diamondString += "*" ;
19
20          diamondString += "\n" ;
21       }
```

```
22
23          // bottom half (last four rows)
24          for ( row = 4; row >= 1; row-- ) {
25
26             for ( spaces = 5; spaces > row; spaces-- )
27                diamondString += " " ;
28
29             for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
30                diamondString += "*" ;
31
32             diamondString += "\n" ;
33          }
34
35          // display result
36          System.out.println( diamondString );
37
38          System.exit( 0 );
39       }
40
41    }  // end class Diamond
```

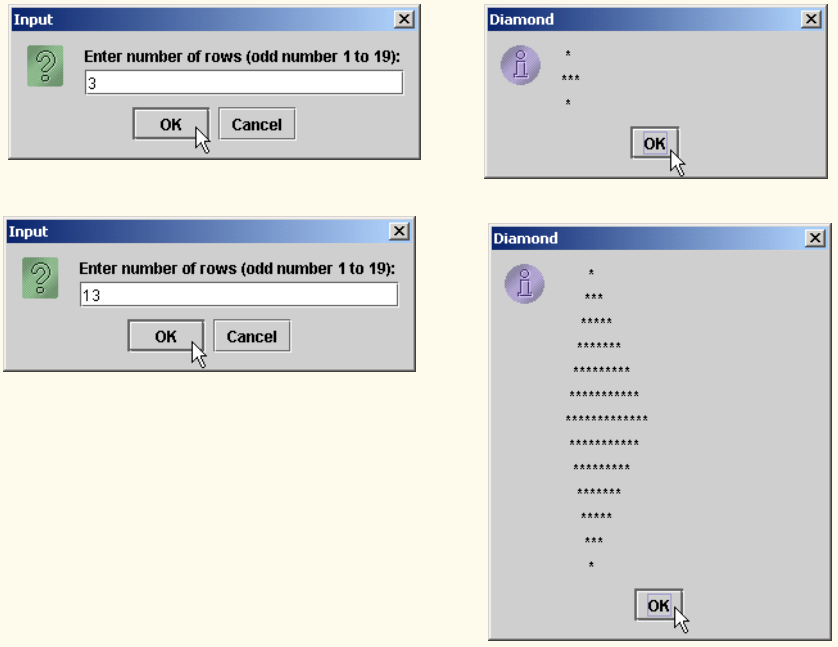
```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

5.21 Modify the application you wrote in Exercise 5.20 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

ANS:

```
1  // Exercise 5.21 Solution: Diamond2.java
2  // Program prints a diamond of a user-specified size
3
4  import javax.swing.*;
5
6  public class Diamond2  {
7
8     public static void main( String args[] )
9     {
10       int row, stars, spaces, numRows;
11       String inputString;
12
13       inputString = JOptionPane.showInputDialog(
14          "Enter number of rows (odd number 1 to 19): " );
15       numRows = Integer.parseInt( inputString );
```

```
16
17          // continue to ask user for entry until within range
18          while ( ( numRows > 19 ) || ( numRows < 0 ) ||
19             ( numRows % 2 == 0 ) ) {
20             JOptionPane.showMessageDialog( null,
21                "Number of rows = " + numRows, "Diamond Input Error",
22                JOptionPane.INFORMATION_MESSAGE );
23             inputString = JOptionPane.showInputDialog(
24                "Enter number of rows (odd number 1 to 19): " );
25             numRows = Integer.parseInt( inputString );
26          }
27
28          String diamondString = "";
29
30          // top half
31          for ( row = 1; row < ( numRows / 2 ) + 1; row++ ) {
32
33          for ( spaces = numRows / 2; spaces >= row; spaces-- )
34             diamondString += " ";
35
36             for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
37                diamondString += "*";
38
39             diamondString += "\n";
40          }
41
42          // bottom half, note that the first clause of the for
43          // loop isn't needed; row is already defined
44          for ( ; row >= 1; row-- ) {
45
46             for ( spaces = numRows / 2; spaces >= row; spaces-- )
47                diamondString += " ";
48
49             for ( stars = 1; stars <= ( 2 * row ) - 1; stars++ )
50                diamondString += "*";
51
52             diamondString += "\n";
53          }
54
55          // show results
56          JOptionPane.showMessageDialog( null, diamondString,
57             "Diamond", JOptionPane.INFORMATION_MESSAGE );
58
59          System.exit( 0 );
60       }
61
62    }  // end class Diamond2
```

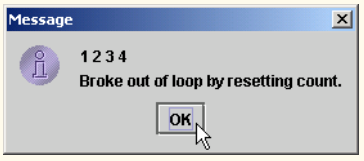


5.22 A criticism of the `break` statement and the `continue` statement is that each is unstructured. Actually, `break` statements and `continue` statements can always be replaced by structured statements, although doing so can be awkward. Describe in general how you would remove any `break` statement from a loop in a program and replace that statement with some structured equivalent. [*Hint*: The `break` statement exits a loop from the body of the loop. The other way to exit is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition."] Use the technique you developed here to remove the `break` statement from the application in Fig. 5.11.

ANS:

```
1  // Exercise. 5.22: WithoutBreak.java
2  // Terminating a loop without break.
3
4  import javax.swing.JOptionPane;
5
6  public class WithoutBreak {
7
8     public static void main( String args[] )
9     {
10       String output = "";
11       int count;
12
13       for ( count = 1; count <= 10; count++ ) {  // loop 10 times
14          output += count + " ";
15
16          if ( count == 4 )  // if count is 4,
17             count = 10;     // reset count to terminate loop
```

```
18
19         } // end for
20
21         output += "\nBroke out of loop by resetting count.";
22         JOptionPane.showMessageDialog( null, output );
23
24         System.exit( 0 );  // terminate application
25
26      } // end main
27
28   } // end class WithoutBreak
```



5.23 What does the following program segment do?

```
for ( i = 1; i <= 5; i++ ) {

   for ( j = 1; j <= 3; j++ ) {

      for ( k = 1; k <= 4; k++ )
         System.out.print( '*' );

      System.out.println();
   }

   System.out.println();
}
```

ANS:

```
****
****
****

****
****
****

****
****
****

****
****
****

****
****
****
```

5.24 Describe in general how you would remove any `continue` statement from a loop in a program and replace that statement with some structured equivalent. Use the technique you developed here to remove the `continue` statement from the program in Fig. 5.12.

ANS:

```
1  // Exercise 5.24 Solution: ContinueTest.java
2  // Alternative to the continue statement in a for structure
3
4  import javax.swing.JOptionPane;
5
6  public class ContinueTest {
7
8     public static void main( String args[] )
9     {
10       String output = "";
11
12       // loop through numbers without printing 5
13       for ( int count = 1; count <= 10; count++ )
14          if ( count != 5 )
15             output += count + " ";
16
17       output += "\nRemoved continue to skip printing 5";
18
19       // show result
20       JOptionPane.showMessageDialog( null, output );
21       System.exit( 0 );
22    }
23
24 }  // end class ContinueTest
```

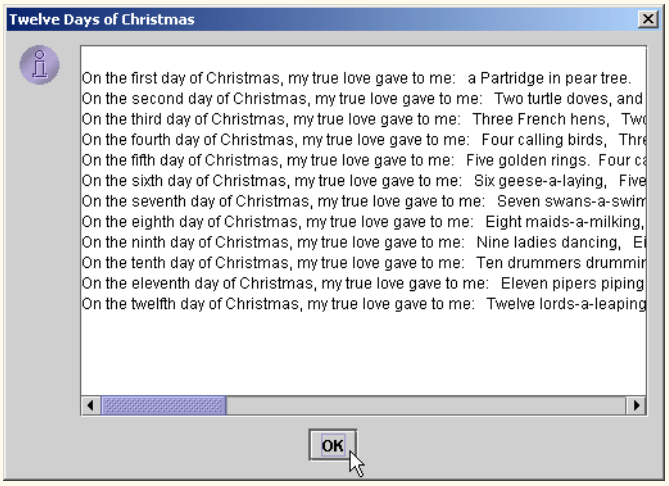
5.25 *("The Twelve Days of Christmas" Song)* Write an application that uses repetition and `switch` statements to print the song "The Twelve Days of Christmas." One `switch` statement should be used to print the day (i.e., "First," "Second," etc.). A separate `switch` statement should be used to print the remainder of each verse. Visit the Web site `www.12days.com/library/carols/12daysofxmas.htm` for the complete lyrics to the song.

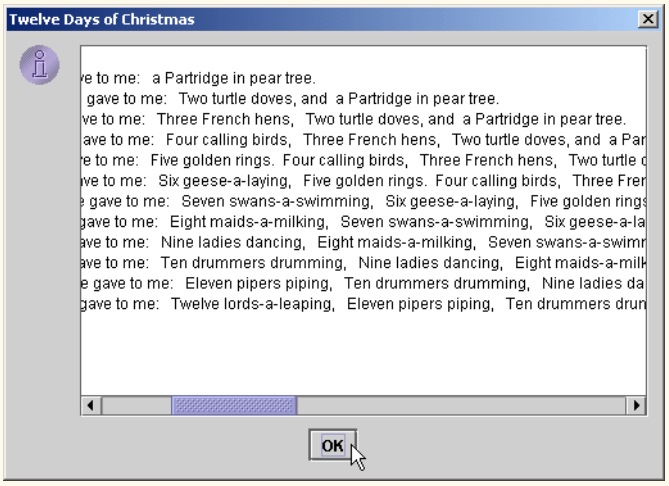
ANS:

```
1  // Exercise 5.25 Solution: Twelve.java
2  // Program prints the 12 days of Christmas song.
3
4  import javax.swing.*;
5
6  public class Twelve {
7
8     // method main begins execution of Java application
9     public static void main( String args[] )
10    {
11       JTextArea outputArea = new JTextArea( 17, 40 );
12       JScrollPane scroller = new JScrollPane( outputArea );
13       String result = "";
14
15       for ( int day = 1; day <= 12; day++ ) {
16          result += "\nOn the ";
17
18          // add correct day to String
19          switch( day ) {
20
21             case 1:
22                result += "first";
23                break;
24
25             case 2:
26                result += "second";
27                break;
28
29             case 3:
30                result += "third";
31                break;
32
33             case 4:
34                result += "fourth";
35                break;
36
37             case 5:
38                result += "fifth";
39                break;
```

```
40
41             case 6:
42                result += "sixth";
43                break;
44
45             case 7:
46                result += "seventh";
47                break;
48
49             case 8:
50                result += "eighth";
51                break;
52
53             case 9:
54                result += "ninth";
55                break;
56
57             case 10:
58                result += "tenth";
59                break;
60
61             case 11:
62                result += "eleventh";
63                break;
64
65             case 12:
66                result += "twelfth";
67                break;
68
69          }  // end switch
70
71          result += " day of Christmas, my true love gave to me: ";
72
73          // add remainder of verse to String
74          switch( day ) {
75
76             case 12:
77                result += "  Twelve lords-a-leaping, ";
78
79             case 11:
80                result += "  Eleven pipers piping, ";
81
82             case 10:
83                result += "  Ten drummers drumming, ";
84
85             case 9:
86                result += "  Nine ladies dancing, ";
87
88             case 8:
89                result += "  Eight maids-a-milking, ";
90
91             case 7:
92                result += "  Seven swans-a-swimming, ";
93
```

```
94                case 6:
95                   result += "  Six geese-a-laying, ";
96
97                case 5:
98                   result += "  Five golden rings.";
99
100               case 4:
101                  result += "  Four calling birds, ";
102
103               case 3:
104                  result += "  Three French hens, ";
105
106               case 2:
107                  result += "  Two turtle doves, and";
108
109               case 1:
110                  result += "  a Partridge in pear tree.";
111
112            }  // end switch
113
114         }  // end for
115
116         outputArea.setText( result );
117         JOptionPane.showMessageDialog( null, scroller,
118            "Twelve Days of Christmas", JOptionPane.INFORMATION_MESSAGE );
119
120         System.exit( 0 );
121
122      }  // end method main
123
124   }  // end class Twelve
```





6

Methods

Objectives

- To understand how to construct programs modularly from small pieces called *methods*.
- To introduce the common math methods available in the Java API.
- To be able to create new methods.
- To understand the mechanisms for passing information between methods.
- To introduce simulation techniques that use random-number generation.
- To understand how the visibility of declarations is limited to specific regions of programs.
- To understand how to wrthite and use methods that call themselves.

Form ever follows function.
Louis Henri Sullivan

E pluribus unum.
(One composed of many.)
Virgil (Publius Vergilius Maro)

O! call back yesterday, bid time return.
William Shakespeare

Call me Ishmael.
Herman Melville

When you call me that, smile.
Owen Wister

SELF-REVIEW EXERCISES

6.1 Fill in the blanks in each of the following statements:

a) A method is invoked with a(n) _____.

ANS: method call.

b) A variable known only within the method in which it is declared is called a(n) _____.

ANS: local variable.

c) The _____ statement in a called method can be used to pass the value of an expression back to the calling method.

ANS: `return`.

d) The keyword _____ indicates that a method does not return a value.

ANS: `void`.

e) The _____ of a declaration is the portion of a program that can refer to the entity in the declaration by name.

ANS: scope.

f) The three ways to return control from a called method to a caller are _____, _____ and _____.

ANS: `return;` or `return` *expression*; or encountering the closing right brace of a method.

g) The _____ method is called once when an applet begins execution.

ANS: `init`.

h) The _____ method produces random numbers.

ANS: `Math.random`.

i) The _____ method is called each time the user of a browser revisits the HTML page on which an applet resides.

ANS: `start`.

j) The _____ method is invoked to draw on an applet.

ANS: `paint`.

k) The _____ method invokes the applet's `update` method, which in turn invokes the applet's `paint` method.

ANS: `repaint`.

l) The _____ method is invoked for an applet each time the user of a browser leaves an HTML page on which the applet resides.

ANS: `stop`

m) A method that calls itself either directly or indirectly is a(n) _____ method.

ANS: recursive

n) A recursive method typically has two components: one that provides a means for the recursion to terminate by testing for a(n) _____ case and one that expresses the problem as a recursive call for a slightly simpler problem than does the original call.

ANS: base

o) In Java, it is possible to have various methods with the same name that each operate on different types or numbers of arguments. This feature is called method _____

ANS: overloading

p) The _____ modifier is used to declare constant variables.

ANS: `final`.

6.2 For the following program, state the scope of each of the following entities:

a) the variable `x`.
ANS: class body.

b) the variable `y`.
ANS: block that defines method `cube`'s body.

c) the method `cube`.
ANS: class body.

d) the method `paint`.
ANS: class body.

e) the variable `yPos`
ANS: block that defines method `paint`'s body.

```
1  public class CubeTest extends JApplet {
2     int x;
3
4     public void paint( Graphics g )
5     {
6        int yPos = 25;
7
8        for ( x = 1; x <= 10; x++ ) {
9           g.drawString( cube( x ), 25, yPos );
10          yPos += 15;
11       }
12    }
13
14    public int cube( int y )
15    {
16       return y * y * y;
17    }
18 }
```

6.3 Write an application that tests if the examples of the math-library method calls shown in Fig. 6.2 actually produce the indicated results.

```
// Exercise 6.3: MathTest.java
// Testing the Math class methods.

public class MathTest {

   public static void main( String args[] )
   {
      System.out.println( "Math.abs( 23.7 ) = " + Math.abs( 23.7 ) );
      System.out.println( "Math.abs( 0.0 ) = " + Math.abs( 0.0 ) );
      System.out.println( "Math.abs( -23.7 ) = " + Math.abs( -23.7 ) );
      System.out.println( "Math.ceil( 9.2 ) = " + Math.ceil( 9.2 ) );
      System.out.println( "Math.ceil( -9.8 ) = " + Math.ceil( -9.8 ) );
      System.out.println( "Math.cos( 0.0 ) = " + Math.cos( 0.0 ) );
      System.out.println( "Math.exp( 1.0 ) = " + Math.exp( 1.0 ) );
      System.out.println( "Math.exp( 2.0 ) = " + Math.exp( 2.0 ) );
      System.out.println( "Math.floor( 9.2 ) = " + Math.floor( 9.2 ) );
      System.out.println( "Math.floor( -9.8 ) = " + Math.floor( -9.8 ) );
      System.out.println( "Math.log( Math.E ) = " +
         Math.log( Math.E ) );
      System.out.println( "Math.log( Math.E * Math.E ) = " +
         Math.log( Math.E * Math.E ) );
      System.out.println( "Math.max( 2.3, 12.7 ) = " +
         Math.max( 2.3, 12.7 ) );
      System.out.println( "Math.max( -2.3, -12.7 ) = " +
         Math.max( -2.3, -12.7 ) );
      System.out.println( "Math.min( 2.3, 12.7 ) = " +
         Math.min( 2.3, 12.7 ) );
      System.out.println( "Math.min( -2.3, -12.7 ) = " +
         Math.min( -2.3, -12.7 ) );
      System.out.println( "Math.pow( 2.0, 7.0 ) = " +
         Math.pow( 2.0, 7.0 ) );
      System.out.println( "Math.pow( 9.0, 0.5 ) = " +
         Math.pow( 9.0, 0.5 ) );
      System.out.println( "M``ath.sin( 0.0 ) = " + Math.sin( 0.0 ) );
      System.out.println( "Math.sqrt( 900.0 ) = " + Math.sqrt( 900.0 ) );
      System.out.println( "Math.sqrt( 9.0 ) = " + Math.sqrt( 9.0 ) );
      System.out.println( "Math.tan( 0.0 ) = " + Math.tan( 0.0 ) );

   } // end main

} // end class MathTest
```

```
Math.abs( 23.7 ) = 23.7
Math.abs( 0.0 ) = 0.0
Math.abs( -23.7 ) = 23.7
Math.ceil( 9.2 ) = 10.0
Math.ceil( -9.8 ) = -9.0
Math.cos( 0.0 ) = 1.0
Math.exp( 1.0 ) = 2.7182818284590455
Math.exp( 2.0 ) = 7.38905609893065
Math.floor( 9.2 ) = 9.0
Math.floor( -9.8 ) = -10.0
Math.log( Math.E ) = 1.0
Math.log( Math.E * Math.E ) = 2.0
Math.max( 2.3, 12.7 ) = 12.7
Math.max( -2.3, -12.7 ) = -2.3
Math.min( 2.3, 12.7 ) = 2.3
Math.min( -2.3, -12.7 ) = -12.7
Math.pow( 2.0, 7.0 ) = 128.0
Math.pow( 9.0, 0.5 ) = 3.0
Math.sin( 0.0 ) = 0.0
Math.sqrt( 900.0 ) = 30.0
Math.sqrt( 9.0 ) = 3.0
Math.tan( 0.0 ) = 0.0
```

6.4 Give the method header for each of the following methods:

a) Method `hypotenuse`, which takes two double-precision, floating-point arguments `side1` and `side2` and returns a double-precision, floating-point result.

ANS:
```
double hypotenuse( double side1, double side2 )
```

b) Method `smallest`, which takes three integers `x`, `y` and `z` and returns an integer.

ANS:
```
int smallest( int x, int y, int z )
```

c) Method `instructions`, which does not take any arguments and does not return a value. [Note: Such methods are commonly used to display instructions to a user.]

ANS:
```
void instructions()
```

d) Method `intToFloat`, which takes an integer argument `number` and returns a floating-point result.

ANS:
```
float intToFloat( int number )
```

6.5 Find the error in each of the following program segments. Explain how to correct the error.

a)
```
int g() {
   System.out.println( "Inside method g" );
   int h() {
      System.out.println( "Inside method h" );
   }
}
```

ANS: Error: Method `h` is declared within method `g`.
Correction: Move the declaration of `h` outside the declaration of `g`.

b)
```
int sum( int x, int y ) {
   int result;
   result = x + y;
}
```
ANS: Error: The method is supposed to return an integer, but does not.
Correction: Delete the variable `result`, and place the statement
```
return x + y;
```
in the method, or add the following statement at the end of the method body:
```
return result;
```
c)
```
int sum( int n ) {
   if ( n == 0 )
      return 0;
   else
      n + sum( n - 1 );
}
```
ANS: Error: The result of `n + sum( n - 1 )` is not returned by this recursive method, resulting in a syntax error.
Correction: Rewrite the statement in the `else` clause as
```
return n + sum( n - 1 );
```
d)
```
void f( float a ); {
   float a;
   System.out.println( a );
}
```
ANS: Error: Both the semicolon after the right parenthesis of the parameter list is incorrect and the parameter a should not be redeclared in the method.
Correction: Delete the semicolon after the right parenthesis of the parameter list, and delete the declaration `float a;`.
e)
```
void product() {
   int a = 6, b = 5, c = 4, result;
   result = a * b * c;
   System.out.println( "Result is " + result );
   return result;
}
```
ANS: Error: The method returns a value when it is not supposed to.
Correction: Change the return type from `void` to `int`.

6.6 Write a complete Java applet to prompt the user for the `double` radius of a sphere, and call method `sphereVolume` to calculate and display the volume of that sphere, using the assignment

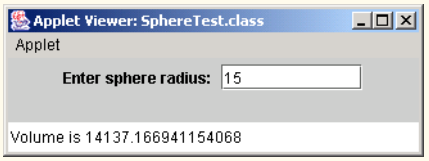
```
volume = ( 4.0 / 3.0 ) * Math.PI * Math.pow( radius, 3 )
```

The user should input the radius through a `JTextField`.

ANS:

```
1  // Exercise 6.6: SphereTest.java
2  // Calculate the volume of a sphere.
3  import java.awt.*;
4  import java.awt.event.*;
5
6  import javax.swing.*;
7
8  public class SphereTest extends JApplet implements ActionListener {
```

```
 9      JLabel promptLabel;
10      JTextField inputField;
11
12      // create GUI
13      public void init()
14      {
15         Container container = getContentPane();
16         container.setLayout( new FlowLayout() );
17
18         promptLabel = new JLabel( "Enter sphere radius: " );
19         inputField = new JTextField( 10 );
20         inputField.addActionListener( this );
21         container.add( promptLabel );
22         container.add( inputField );
23
24      } // end method init
25
26      // calculate sphere volume when user presses Enter in inputField
27      public void actionPerformed( ActionEvent actionEvent )
28      {
29         double radius =
30            Double.parseDouble( actionEvent.getActionCommand() );
31
32         showStatus( "Volume is " + sphereVolume( radius ) );
33
34      } // end method actionPerformed
35
36      // calculate and return sphere volume
37      public double sphereVolume( double radius )
38      {
39         double volume = ( 4.0 / 3.0 ) * Math.PI * Math.pow( radius, 3 );
40
41         return volume;
42
43      } // end method sphereVolume
44
45   } // end class SphereTest
```



EXERCISES

6.7 What is the value of x after each of the following statements is executed?

a) `x = Math.abs( 7.5 );`
ANS: 7.5

b) `x = Math.floor( 7.5 );`
ANS: 7.0

c) `x = Math.abs( 0.0 );`
ANS: 0.0

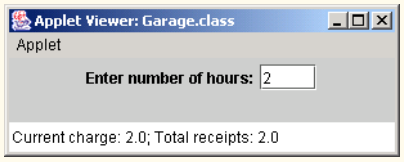
d) `x = Math.ceil( 0.0 );`
ANS: 0.0
e) `x = Math.abs( -6.4 );`
ANS: 6.4
f) `x = Math.ceil( -6.4 );`
ANS: -6.0
g) `x = Math.ceil( -Math.abs( -8 + Math.floor( -5.5 ) ) );`
ANS: -14.0

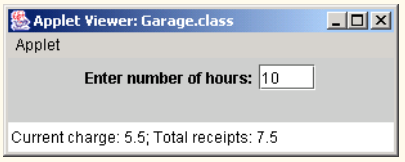
6.8 A parking garage charges a $2.00 minimum fee to park for up to three hours. The garage charges an additional $0.50 per hour for each hour *or part thereof* in excess of three hours. The maximum charge for any given 24-hour period is $10.00. Assume that no car parks for longer than 24 hours at a time. Write an applet that calculates and displays the parking charges for each customer who parked in the garage yesterday. You should enter in a `JTextField` the hours parked for each customer. The program should display the charge for the current customer and should calculate and display the running total of yesterday's receipts. The program should use the method `calculateCharges` to determine the charge for each customer.

ANS:

```
1  // Exercise 6.8 Solution: Garage.java
2  // Program calculates charges for parking
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class Garage extends JApplet implements ActionListener {
9     JTextField hoursInput;
10    JLabel hoursPrompt;
11    double totalReceipts, fee;
12
13    public void init()
14    {
15       // create a label and a text field
16       hoursPrompt = new JLabel( "Enter number of hours:" );
17       hoursInput = new JTextField( 4 );
18       hoursInput.addActionListener( this );
19
20       // add to applet
21       Container container = getContentPane();
22       container.setLayout( new FlowLayout() );
23       container.add( hoursPrompt);
24       container.add( hoursInput );
25    }
26
27    // adds newest fee to total charges
28    public void actionPerformed( ActionEvent e )
29    {
30       double hours = Double.parseDouble( e.getActionCommand() );
31
32       fee = calculateCharges( hours );
33       totalReceipts += fee;
34       showStatus( "Current charge: " + fee +
```

```
35            "; Total receipts: " + totalReceipts );
36      }
37
38      // determines fee based on time
39      public double calculateCharges( double hours )
40      {
41         // apply minimum charge
42         double charge = 2.0;
43
44         // add extra fees as applicable
45         if ( hours > 3.0 )
46            charge = 2.0 + 0.5 * Math.ceil( hours - 3.0 );
47
48         // apply maximum value if needed
49         if ( charge > 10.0 )
50            charge = 10.0;
51
52         return charge;
53      }
54
55   }  // end class Garage
```





6.9 An application of method `Math.floor` is rounding a value to the nearest integer. The statement

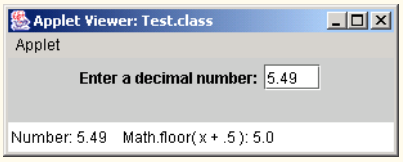
```
y = Math.floor( x + 0.5 );
```

will round the number `x` to the nearest integer and assign the result to `y`. Write an applet that reads `double` values and uses the preceding statement to round each of the numbers to the nearest integer. For each number processed, display both the original number and the rounded number.

ANS:

```
1   // Exercise 6.9 Solution: Test.java
2   // Program tests Math.floor.
3
4   import java.awt.event.*;
5   import java.awt.*;
6   import javax.swing.*;
7
8   public class Test extends JApplet implements ActionListener {
9      JTextField inputField;
10     JLabel inputLabel;
11
12     // set up GUI components
13     public void init()
14     {
```

```
15          inputLabel = new JLabel( "Enter a decimal number:" );
16          inputField = new JTextField( 4 );
17          inputField.addActionListener( this );
18
19          Container container = getContentPane();
20          container.setLayout( new FlowLayout() );
21          container.add( inputLabel );
22          container.add( inputField );
23       }
24
25       // execute floor function on input number
26       public void actionPerformed( ActionEvent e )
27       {
28          double x = Double.parseDouble( inputField.getText() );
29
30          showStatus( "Number: " + x + "    Math.floor( x + .5 ): "
31             + String.valueOf( Math.floor( x + .5 ) ) );
32       }
33
34    }  // end class Test
```



6.10 `Math.floor` may be used to round a number to a specific decimal place. The statement

```
y = Math.floor( x * 10 + 0.5 ) / 10;
```

rounds `x` to the tenths position (i.e., the first position to the right of the decimal point). The statement

```
y = Math.floor( x * 100 + 0.5 ) / 100;
```

rounds `x` to the hundredths position (i.e., the second position to the right of the decimal point). Write an applet that defines four methods to round a number **x** in various ways:

a) `roundToInteger( number )`
b) `roundToTenths( number )`
c) `roundToHundredths( number )`
d) `roundToThousandths( number )`

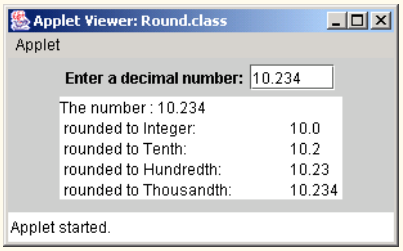
For each value read, your program should display the original value, the number rounded to the nearest integer, the number rounded to the nearest tenth, the number rounded to the nearest hundredth and the number rounded to the nearest thousandth.

ANS:

```
1  // Exercise 6.10 Solution: Round.java
2  // Program tests rounding with Math.floor
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
```

```
 7
 8  public class Round extends JApplet implements ActionListener {
 9     JTextField inputNumber;
10     JTextArea outputData;
11     JLabel inputLabel;
12
13     public void init()
14     {
15        // create facilities for input and output
16        inputLabel = new JLabel( "Enter a decimal number:" );
17        inputNumber = new JTextField( 6 );
18        inputNumber.addActionListener( this );
19        outputData = new JTextArea( 5,20 );
20
21        // add to applet
22        Container container = getContentPane();
23        container.setLayout( new FlowLayout() );
24        container.add( inputLabel );
25        container.add( inputNumber );
26        container.add( outputData );
27     }
28
29     // read in value and perform rounding
30     public void actionPerformed( ActionEvent e )
31     {
32        double x = Double.parseDouble( e.getActionCommand() );
33
34        //create an output String with appropriate rounding
35        outputData.setText( "The number : " + String.valueOf( x ) +
36           "\n rounded to Integer:\t " +
37           String.valueOf( roundToInteger( x ) ) +
38           "\n rounded to Tenth:\t " +
39           String.valueOf( roundToTenths( x ) ) +
40           "\n rounded to Hundredth:\t " +
41           String.valueOf( roundToHundredths( x ) ) +
42           "\n rounded to Thousandth:\t " +
43           String.valueOf( roundToThousandths( x ) ) );
44     }
45
46     // round to ones place
47     public double roundToInteger( double number )
48     {
49        return( Math.floor( number + .5 ) );
50     }
51
52     // round to tenths place
53     public double roundToTenths( double number )
54     {
55        return( Math.floor( number * 10 + .5 ) / 10 );
56     }
57
58     // round to hundredths place
59     public double roundToHundredths( double number )
60     {
```

```
61          return( Math.floor( number * 100 + .5 ) / 100 );
62       }
63
64       // round to thousandths place
65       public double roundToThousandths( double number )
66       {
67          return( Math.floor( number * 1000 + .5 ) / 1000 );
68       }
69
70    }  // end class Round
```



6.11 Answer each of the following questions:

a) What does it mean to choose numbers "at random?"

ANS: Every number has an equal chance of being chosen at any time.

b) Why is the `Math.random` method useful for simulating games of chance?

ANS: Because it produces a series of random numbers.

c) Why is it often necessary to scale or shift the values produced by `Math.random`?

ANS: To produce random numbers in a specific range.

d) Why is computerized simulation of real-world situations a useful technique?

ANS: It enables more accurate predictions of random events such as cars arriving at toll booths and people arriving in lines at a supermarket. The results of a simulation can help determine how many toll booths to have open or how many cashiers to have open at specified times.

6.12 Write statements that assign random integers to the variable *n* in the following ranges:

a) $1 \le n \le 2$

ANS: `n = ( int ) ( 1 + Math.random() * 2 );`

b) $1 \le n \le 100$

ANS: `n = ( int ) ( 1 + Math.random() * 100 );`

c) $0 \le n \le 9$

ANS: `n = ( int ) ( Math.random() * 10 );`

d) $1000 \le n \le 1112$

ANS: `n = ( int ) ( 1000 + Math.random() * 113 );`

e) $-1 \le n \le 1$

ANS: `n = ( int ) ( -1 + Math.random() * 3 );`

f) $-3 \le n \le 11$

ANS: `n = ( int ) ( -3 + Math.random() * 15 );`

6.13 For each of the following sets of integers, write a single statement that will print a number at random from the set:

a) 2, 4, 6, 8, 10.

ANS: `System.out.print( ( int ) ( 1 + Math.random() * 5 ) * 2 );`

b) 3, 5, 7, 9, 11.
ANS: `System.out.print( ( int ) ( 1 + Math.random() * 5 ) * 2 + 1 );`
c) 6, 10, 14, 18, 22.
ANS: `System.out.print( ( int ) ( Math.random() * 5 ) * 4 + 6 );`

6.14 Write a method `integerPower( base, exponent )` that returns the value of

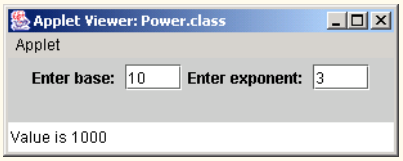
base ^{exponent}

For example, `integerPower( 3, 4 )` calculates 3^4 (or `3 * 3 * 3 * 3`). Assume that `exponent` is a positive, nonzero integer and that `base` is an integer. Method `integerPower` should use a `for` or `while` loop to control the calculation. Do not use any math-library methods. Incorporate this method into an applet that reads integer values for `base` and `exponent` from `JTextField` objects and performs the calculation with the `integerPower` method. [*Note*: Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextFields`, but pressing *Enter* only in the second `JTextField`.]

ANS:

```
1  // Exercise 6.14 Solution: Power.java
2  // Program calculates an exponent
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Power extends JApplet implements ActionListener {
9     JLabel basePrompt, exponentPrompt;
10    JTextField baseInput, exponentInput;
11
12    public void init()
13    {
14       // create input areas
15       basePrompt = new JLabel( "Enter base: " );
16       baseInput = new JTextField( 4 );
17
18       exponentPrompt  = new JLabel( "Enter exponent: " );
19       exponentInput  = new JTextField( 4 );
20       exponentInput.addActionListener( this );
21
22       // add components to applet
23       Container container = getContentPane();
24       container.setLayout( new FlowLayout() );
25       container.add( basePrompt );
26       container.add( baseInput );
27       container.add( exponentPrompt );
28       container.add( exponentInput );
29    }
30
31    // get user input and calculate number
32    public void actionPerformed( ActionEvent e )
33    {
34       // get values from user
35       int base = Integer.parseInt( baseInput.getText() );
36       int exponent = Integer.parseInt( exponentInput.getText() );
```

```
37
38          // raise to exponent if appropriate
39          if ( exponent > 0 ) {
40             int result = integerPower( base, exponent );
41             showStatus( "Value is " + result );
42          }
43
44          else
45             showStatus( "Invalid Exponent." );
46       }
47
48       // raise integer base to the exponent power
49       public int integerPower( int base, int exponent )
50       {
51          int product = 1;
52
53          for ( int x = 1; x <= exponent; x++ )
54             product *= base;
55
56          return product;
57       }
58
59    }  // end class Power
```



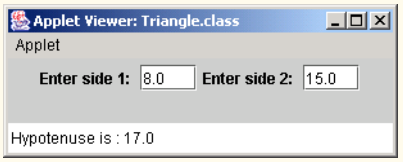
6.15 Define a method `hypotenuse` that calculates the length of the hypotenuse of a right triangle when the lengths of the other two sides are given. (Use the sample data in Fig. 6.21.) The method should take two arguments of type `double` and return the hypotenuse as a `double`. Incorporate this method into an applet that reads values for `side1` and `side2` from `JTextField` objects and performs the calculation with the `hypotenuse` method. Determine the length of the hypotenuse for each of the triangles in Fig. 6.21. [*Note*: Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextFields`, but pressing *Enter* only in the second `JTextField`.]

Triangle	Side 1	Side 2
1	3.0	4.0
2	5.0	12.0
3	8.0	15.0

Fig. 6.21 Values for the sides of triangles in Exercise 6.15.

ANS:

```
// Exercise 6.15 Solution: Triangle.java
// Program calculates the hypotenuse of a right triangle.

import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class Triangle extends JApplet implements ActionListener {
   JTextField side1Input, side2Input;
   JLabel side1Prompt, side2Prompt;

   // set up GUI components
   public void init()
   {
      side1Prompt = new JLabel( "Enter side 1: " );
      side2Prompt = new JLabel( "Enter side 2: " );
      side1Input = new JTextField( 4 );
      side2Input = new JTextField( 4 );
      side2Input.addActionListener( this );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      container.add( side1Prompt );
      container.add( side1Input );
      container.add( side2Prompt );
      container.add( side2Input );
   }

   // perform calculation of hypotenuse
   public void actionPerformed( ActionEvent actionEvent )
   {
      double side1, side2;

      side1 = Double.parseDouble( side1Input.getText() );
      side2 = Double.parseDouble( side2Input.getText() );

      double result = hypotenuse( side1, side2 );
      showStatus( "Hypotenuse is : " + result );
   }

   // calculate hypotenuse given lengths of two sides
   public double hypotenuse( double side1, double side2 )
   {
      double hypotenuseSquared = Math.pow( side1, 2 ) +
         Math.pow( side2, 2 );

      return Math.sqrt( hypotenuseSquared );
   }

}  // end class Triangle
```

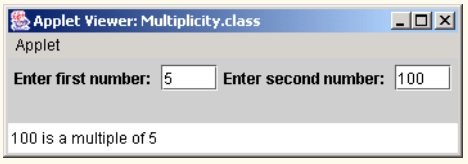


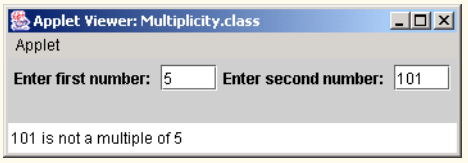
6.16 Write a method `multiple` that determines, for a pair of integers, whether the second integer is a multiple of the first. The method should take two integer arguments and return `true` if the second is a multiple of the first and `false` otherwise. Incorporate this method into an applet that inputs a series of pairs of integers (one pair at a time, using `JTextFields`). [*Note:* Register for event handling on only the second `JTextField`. The user should interact with the program by typing numbers in both `JTextFields`, but pressing *Enter* only in the second `JTextField`.]

ANS:

```
1  // Exercise 6.16 Solution: Multiplicity.java
2  // Determines if the second number entered is a multiple of the first.
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class Multiplicity extends JApplet implements ActionListener {
9     JTextField input, input2;
10    JLabel prompt, prompt2;
11
12    public void init()
13    {
14       // create text fields and labels
15       prompt = new JLabel( "Enter first number: " );
16       input = new JTextField( 4 );
17       prompt2 = new JLabel( "Enter second number: " );
18       input2 = new JTextField( 4 );
19       input2.addActionListener( this );
20
21       // add components to container
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24       container.add( prompt );
25       container.add( input );
26       container.add( prompt2 );
27       container.add( input2 );
28    }
29
30    // get input values from user and determine if a multiple
31    public void actionPerformed( ActionEvent e )
32    {
33       // get user input
34       int first = Integer.parseInt( input.getText() );
35       int second = Integer.parseInt( input2.getText() );
36
37       if ( multiple( first, second ) == true )
38          showStatus( second + " is a multiple of " + first );
```

```
39
40          else
41             showStatus( second + " is not a multiple of " + first );
42
43       }
44
45       // determine if one int is a multiple of the second
46       public boolean multiple( int one, int two )
47       {
48          if ( ( two % one == 0 ) && two != 0 )
49             return true;
50
51          return false;
52       }
53
54    }  // end class Multiplicity
```



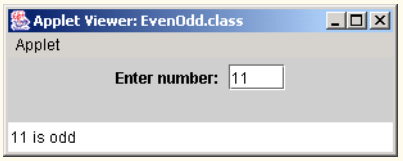


6.17 Write a method `isEven` that uses the remainder operator (%) to determine whether an integer is even. The method should take an integer argument and return `true` if the integer is even and `false` otherwise. Incorporate this method into an applet that inputs a sequence of integers (one at a time, using a `JTextField`).

ANS:

```
1  // Exercise 6.17 Solution: EvenOdd.java
2  // Program determines if a number is odd or even.
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class EvenOdd extends JApplet implements ActionListener {
9     JTextField inputField;
10    JLabel prompt;
11
12    // set up GUI components
13    public void init()
14    {
15       prompt = new JLabel( "Enter number: " );
16       inputField = new JTextField( 4 );
17       inputField.addActionListener( this );
18
19       Container container = getContentPane();
20       container.setLayout( new FlowLayout() );
21       container.add( prompt );
22       container.add( inputField );
```

```
23
24       }  // end method init
25
26       // determine if input is even or odd
27       public void actionPerformed( ActionEvent e )
28       {
29          int number = Integer.parseInt( inputField.getText() );
30          String result = "";
31
32          if ( isEven( number ) == true )
33             result = number + " is even";
34
35          else
36             result = number + " is odd ";
37
38          showStatus( result );
39
40       }  // end method actionPerformed
41
42       // return true if number is even
43       public boolean isEven( int number )
44       {
45          if ( number % 2 == 0 )
46             return true;
47
48          return false;
49
50       }  // end method isEven
51
52    }  // end class EvenOdd
```



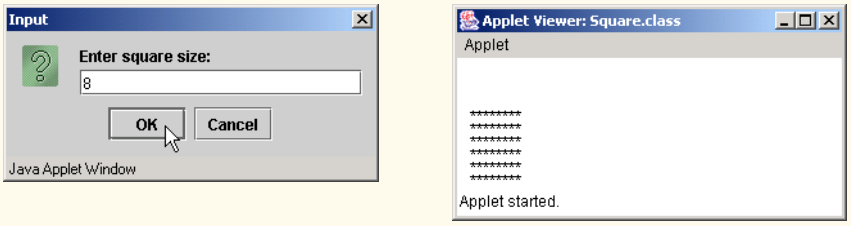
6.18 Write a method `squareOfAsterisks` that displays a solid square (the same number of rows and columns) of asterisks whose side is specified in integer parameter `side`. For example, if `side` is 4, the method displays the pattern of asterisks at the top of the next page.

```
****
****
****
****
```

Incorporate this method into an applet that reads an integer value for `side` from the user and performs the drawing with the `squareOfAsterisks` method. Note that this method should be called from the applet's `paint` method and should be passed the `Graphics` object from `paint`.

ANS:

```
// Exercise 6.18 Solution: Square.java
// Program draws a square of asterisks.

import java.awt.*;
import javax.swing.*;

public class Square extends JApplet {
   int inputNumber;

   // obtain value from user
   public void init()
   {
      String inputString = JOptionPane.showInputDialog(
         "Enter square size:" );

      inputNumber = Integer.parseInt( inputString );

   }  // end method init

   // draw square of asteriks on applet's background
   public void squareOfAsterisks( Graphics g, int side )
   {
      int y = 50, x = 5;

      for ( int count = 1; count <= side * side; count++ ) {
         g.drawString( "*", x += 5, y );

         if ( count % side == 0 ) {
            y += 10;
            x = 5;
         }

      } // end for loop

   }  // end method squareOfAsterisks

   // execute method squareOfAsterisks
   public void paint( Graphics g )
   {
      squareOfAsterisks( g, inputNumber );

   }  // end method paint

}  // end class Square
```



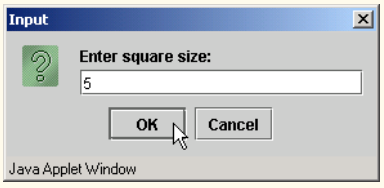
6.19 Modify the method created in Exercise 6.18 to form the square out of whatever character is contained in character parameter `fillCharacter`. Thus, if `side` is 5 and `fillCharacter` is "#", the method should print

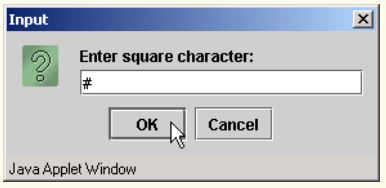
```
#####
#####
#####
#####
#####
```

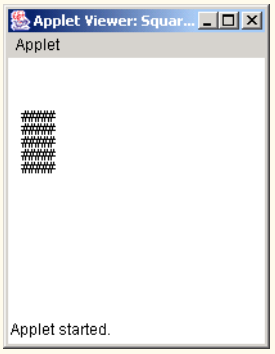
ANS:

```
1  // Exercise 6.19 Solution: Square2.java
2  // Program draws a square of asterisks
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Square2 extends JApplet {
9     int size;
10    String fillCharacter;
11
12    public void init()
13    {
14       // get user-specified square size
15       String input = JOptionPane.showInputDialog( "Enter square size:" );
16       size = Integer.parseInt( input );
17
18       // choose character to compose the square with
19       fillCharacter = JOptionPane.showInputDialog(
20          "Enter square character:" );
21    }
22
23    // draw a square filled with the specified char
24    public void paint( Graphics g )
25    {
26       fillSquare( g );
27    }
28
29    public void fillSquare( Graphics g )
30    {
31       // set initial position and value of output
32       int y = 50, x = 5;
```

```
33
34          // draw the square, one value at a time
35          for ( int a = 1; a <= size * size; a++ ) {
36             g.drawString( fillCharacter, x += 5, y );
37
38             if ( a % size == 0 ) {
39                y += 10;
40                x = 5;
41             }
42          }
43       }
44
45    }  // end class Square2
```





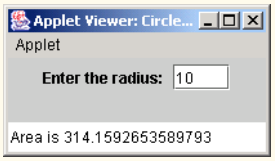


6.20 Write an applet that uses a method `circleArea` to prompt the user for the radius of a circle and to calculate and print the area of that circle..

ANS:

```
1  // Exercise 6.20 Solution: Circle.java
2  // Program calulates the area of a circle.
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class Circle extends JApplet implements ActionListener {
9     JTextField inputField;
10    JLabel prompt;
11
```

```
12     // set up GUI components
13     public void init()
14     {
15        prompt = new JLabel( "Enter the radius: " );
16        inputField = new JTextField( 4 );
17        inputField.addActionListener( this );
18
19        Container container = getContentPane();
20        container.setLayout( new FlowLayout() );
21        container.add( prompt );
22        container.add( inputField );
23     }
24
25     // obtain user input and call method circleArea
26     public void actionPerformed( ActionEvent actionEvent )
27     {
28        showStatus( "" );
29
30        int theRadius = Integer.parseInt( inputField.getText() );
31        circleArea( theRadius );
32     }
33
34     // calculate area
35     public void circleArea( int radius )
36     {
37        showStatus( "Area is " + Math.PI * radius * radius );
38     }
39
40  } // end class Circle
```

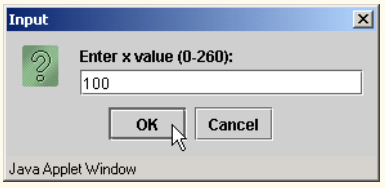


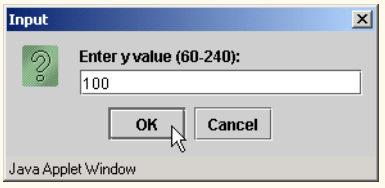
6.21 Modify the program of Exercise 6.18 to draw a solid square with the `fillRect` method of the `Graphics` class. Method `fillRect` requires four arguments: *x*-coordinate, *y*-coordinate, width and height. Allow the user to input the coordinates at which the square should appear and the length of the side.

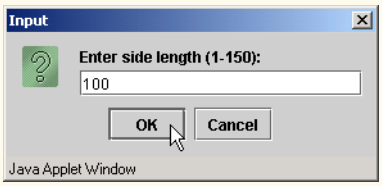
ANS:

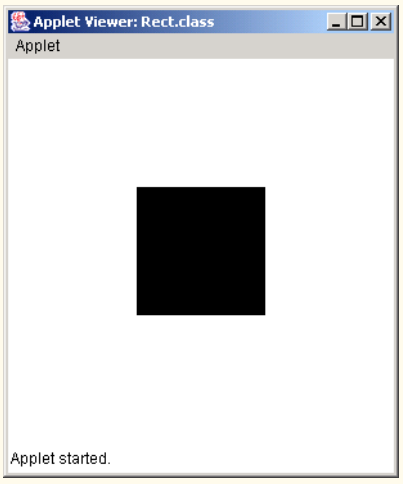
```
1  // Exercise 6.21 Solution: Rect.java
2  // Program draws a rectangle on the applet
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class Rect extends JApplet {
8     int x, y, sideLength;
9
```

```
10     public void init()
11     {
12        // determine x and y values of top left corner of
13        // rectangle and side length, keep asking if values
14        // are not in range
15        do {
16           String inputString = JOptionPane.showInputDialog(
17              "Enter x value (0-260):" );
18           x = Integer.parseInt( inputString );
19
20           inputString = JOptionPane.showInputDialog(
21              "Enter y value (60-240):" );
22           y = Integer.parseInt( inputString );
23
24           inputString = JOptionPane.showInputDialog(
25              "Enter side length (1-150):" );
26           sideLength = Integer.parseInt( inputString );
27
28           // test for validity
29           if ( x < 0 || x > 260 || y < 60 || y > 240
30              || sideLength < 1 || sideLength > 150 )
31              JOptionPane.showMessageDialog ( null, "Invalid entry!!!",
32                 "Rectangle input error", JOptionPane.INFORMATION_MESSAGE );
33
34        }  while ( ( x < 0 || x > 260 || y < 60 || y > 240
35              || sideLength < 1 || sideLength > 150 ) );
36
37     } // end method init
38
39     // draw a rectangle with the given coordinates
40     public void paint( Graphics g )
41     {
42        g.fillRect( x, y, width, height );
43     }
44
45  }  // end class Rect
```









6.22 Write program segments that accomplish each of the following tasks:

a) Calculate the integer part of the quotient when integer `a` is divided by integer `b`.

b) Calculate the integer remainder when integer `a` is divided by integer `b`.

c) Use the program pieces developed in parts (a) and (b) to write a method `displayDigits` that receives an integer between `1` and `99999` and displays it as a sequence of digits, separating each pair of digits by two spaces. For example, the integer `4562` should appear as

```
4  5  6  2
```

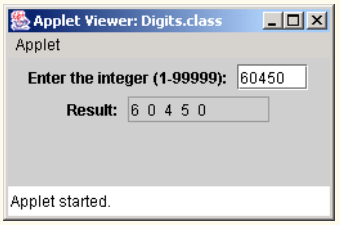
d) Incorporate the method developed in part (c) into an applet that inputs an integer from a textfield and calls `displayDigits` by passing the method the integer entered. Display the results in a second textfield.

ANS:

```
1  // Exercise 6.22 Solution: Digits.java
2  // Program separates a four digit number
3  // into its individual digits.
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class Digits extends JApplet implements ActionListener {
10    JTextField inputField, resultField;
11    JLabel inputLabel, resultLabel;
12
13    // set up GUI and obtain value from user
14    public void init()
15    {
16       inputLabel = new JLabel( "Enter the integer (1-99999): " );
17       inputField = new JTextField( 5 );
18       inputField.addActionListener( this );
19
```

```
20          resultLabel = new JLabel( "Result: " );
21          resultField = new JTextField( 10 );
22          resultField.setEditable( false );
23
24          Container container = getContentPane();
25          container.setLayout( new FlowLayout() );
26          container.add( inputLabel );
27          container.add( inputField );
28          container.add( resultLabel );
29          container.add( resultField );
30       }
31
32       // obtain user input and call method circleArea
33       public void actionPerformed( ActionEvent actionEvent )
34       {
35          int inputNumber =
36             Integer.parseInt( inputField.getText() );
37
38          if ( inputNumber <= 99999 && inputNumber >= 1 )
39             displayDigits( inputNumber );
40          else
41             JOptionPane.showMessageDialog( null,
42                "Input a number between 1 and 99999",
43                "Invalid Number", JOptionPane.ERROR_MESSAGE );
44       }
45
46       // part A
47       public int quotient( int a, int b )
48       {
49          return a / b;
50       }
51
52       // part B
53       public int remainder( int a, int b )
54       {
55          return a % b;
56       }
57
58       // part C
59       public void displayDigits( int number )
60       {
61          int divisor = 1, digit;
62          String result = "";
63
64          // Loop for highest divisor
65          for( int i = 1; i < number; i *= 10 )
66             divisor = i;
67
68          while ( divisor >= 1 ) {
69             digit = quotient( number, divisor );
70
71             result += digit + "   ";
72
73             number = remainder( number, divisor );
```

```
74             divisor = quotient( divisor, 10 );
75          }
76
77          resultField.setText( result );
78       }
79
80    }  // end class Digits
```



6.23 Implement the following integer methods:

a) Method `celsius` returns the Celsius equivalent of a Fahrenheit temperature, using the calculation

```
C = 5.0 / 9.0 * ( F - 32 );
```

b) Method `fahrenheit` returns the Fahrenheit equivalent of a Celsius temperature, using the calculation

```
F = 9.0 / 5.0 * C + 32;
```

c) Use the methods from parts (a) and (b) to write an applet that enables the user either to enter a Fahrenheit temperature and display the Celsius equivalent or to enter a Celsius temperature and display the Fahrenheit equivalent.

[*Note*: This applet will require two `JTextField` objects that have registered action events. When `actionPerformed` is called, the `ActionEvent` parameter has method `getSource()` to determine the GUI component with which the user interacted. Your `actionPerformed` method should contain an `if...else` statement of the form

```
if ( actionEvent.getSource() == input1 ) {
   // process input1 interaction here
}
else {  // e.getSource() == input2
   // process input2 interaction here
}
```

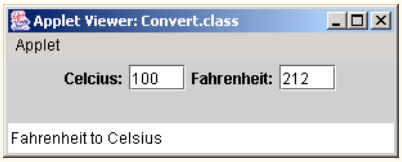
where `input1` and `input2` are `JTextField` references.]

ANS:

```
1  // Exercise 6.23 Solution: Convert.java
2  // Program converts Fahrenheit to Celsius. and vice versa.
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
```

```
 7
 8  public class Convert extends JApplet implements ActionListener {
 9     JTextField celsiusInput, fahrenheitInput;
10     JLabel celsiusLabel, fahrenheitLabel;
11
12     // set up GUI components
13     public void init()
14     {
15        celsiusLabel = new JLabel( "Celcius:" );
16        fahrenheitLabel = new JLabel( "Fahrenheit:" );
17        celsiusInput = new JTextField( 4 );
18        fahrenheitInput = new JTextField( 4 );
19        celsiusInput.addActionListener( this );
20        fahrenheitInput.addActionListener( this );
21
22        Container container = getContentPane();
23        container.setLayout( new FlowLayout() );
24        container.add( celsiusLabel );
25        container.add( celsiusInput );
26        container.add( fahrenheitLabel );
27        container.add( fahrenheitInput );
28
29     }  // end method init
30
31     // perform conversion
32     public void actionPerformed( ActionEvent actionEvent )
33     {
34        // convert from Celsius to Fahrenheit
35        if ( actionEvent.getSource() == celsiusInput ) {
36           int celsiusNumber = Integer.parseInt( celsiusInput.getText() );
37
38           fahrenheitInput.setText(
39              String.valueOf( fahrenheit( celsiusNumber ) ) );
40           showStatus( "Celsius to Fahrenheit" );
41        }
42
43        // convert from Fahrenheit to Celsius
44        else {
45           int fahrenheitNumber =
46              Integer.parseInt( fahrenheitInput.getText() );
47
48           celsiusInput.setText(
49              String.valueOf( celsius( fahrenheitNumber ) ) );
50           showStatus( "Fahrenheit to Celsius" );
51        }
52
53     }  // end method actionPerformed
54
55     // return Celsius equivalent of Fahrenheit temperature
56     public int celsius( int fahrenheitTemperature )
57     {
58        return ( ( int ) ( 5.0 / 9.0 * ( fahrenheitTemperature - 32 ) ) );
59
60     }  // end method celcius
```

```
61
62     // return Fahrenheit equivalent of Celsius temperature
63     public int fahrenheit( int celsiusTemperature )
64     {
65        return ( ( int ) ( 9.0 / 5.0 * celsiusTemperature + 32 ) );
66
67     }  // end method fahrenheit
68
69  }  // end class Convert
```

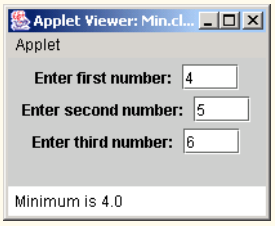


6.24 Write a method `minimum3` that returns the smallest of three floating-point numbers. Use the `Math.min` method to implement `minimum3`. Incorporate the method into an applet that reads three values from the user and determines the smallest value. Display the result in the status bar.

ANS:

```
1  // Exercise 6.24 Solution: Min.java
2  // Program finds the minimum of 3 numbers
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Min extends JApplet implements ActionListener {
9     JTextField input1, input2, input3;
10    JLabel label1, label2, label3;
11
12    public void init()
13    {
14       // create a label and text field for each number
15       label1 = new JLabel( "Enter first number: " );
16       label2 = new JLabel( "Enter second number: " );
17       label3 = new JLabel( "Enter third number: " );
18       input1 = new JTextField( 4 );
19       input2 = new JTextField( 4 );
20       input3 = new JTextField( 4 );
21
22       // only the final text field has a listener
23       input3.addActionListener( this );
24
25       // add components to container
26       Container container = getContentPane();
27       container.setLayout( new FlowLayout() );
28       container.add( label1 );
29       container.add( input1 );
30       container.add( label2 );
31       container.add( input2 );
```

```
32          container.add( label3 );
33          container.add( input3 );
34       }
35
36       // retrieve three values when user presses Enter
37       public void actionPerformed( ActionEvent e )
38       {
39          float one, two, three;
40
41          Float input = new Float( input1.getText() );
42          one = input.floatValue();
43
44          input = new Float( input2.getText() );
45          two = input.floatValue();
46
47          input = new Float( input3.getText() );
48          three = input.floatValue();
49
50          // print out result
51          showStatus( " Minimum is " + minimum3( one, two, three ) );
52       }
53
54       // determine the smallest of three numbers
55       public float minimum3( float one, float two, float three )
56       {
57          // use a nested pair of min statements
58          float x = Math.min( Math.min( one, two ), three );
59          return x;
60       }
61
62    }  // end class Min
```

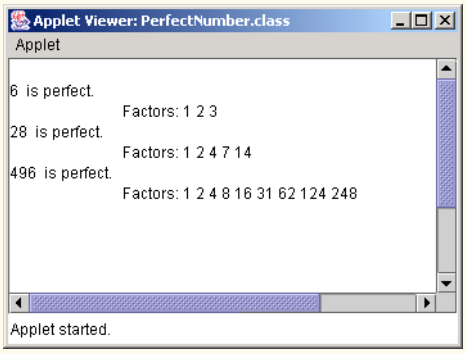


6.25 An integer number is said to be a *perfect number* if its factors, including 1 (but not the number itself), sum to the number. For example, 6 is a perfect number, because 6 = 1 + 2 + 3. Write a method `perfect` that determines whether parameter `number` is a perfect number. Use this method in an applet that determines and displays all the perfect numbers between 1 and 1000. Print the factors of each perfect number to confirm that the number is indeed perfect. Challenge the computing power of your computer by testing numbers much larger than 1000. Display the results in a `JTextArea` that has scrolling functionality.

ANS:

```
1  // Exercise 6.25 Solution: PerfectNumber.java
2  // Program displays all perfect numbers between 1 and 1000.
```

```
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class PerfectNumber extends JApplet {
8     JTextArea outputArea;
9     JScrollPane scroller;
10
11    // set up GUI and print out perfect numbers
12    public void init()
13    {
14       outputArea = new JTextArea( 17, 40 );
15       scroller = new JScrollPane( outputArea );
16
17       String outputString = "";
18
19       for ( int number = 2; number <= 1000; number++ ) {
20          String result = perfect( number );
21
22          if ( result != "0" )
23             outputString += "\n" + number + "  is perfect."
24                + "\n\tFactors: " + result;
25       }
26
27       outputArea.setText( outputString );
28       Container container = getContentPane();
29       container.add( scroller );
30
31    }  // end method init
32
33    // returns a string of factors if parameter is a
34    // perfect number, or a string containing 0 if it isn't.
35    public String perfect( int value )
36    {
37       int factorSum = 1;
38       String factors = "1 ";
39
40       for ( int test = 2; test <= value / 2; test++ ) {
41
42          if ( value % test == 0 ) {
43             factorSum += test;
44             factors += test + " ";
45          }
46       }
47
48       if ( factorSum == value )
49          return factors;
50
51       return "0";
52
53    }  // end method perfect
54
55 }  // end class PerfectNumber
```



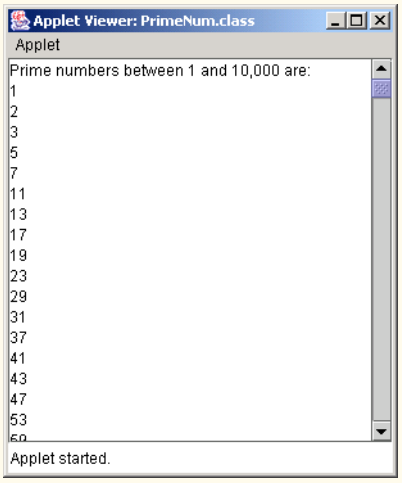
6.26 An integer is said to be *prime* if it is divisible only by 1 and itself. For example, 2, 3, 5 and 7 are prime, but 4, 6, 8 and 9 are not.

a) Write a method that determines whether a number is prime.

b) Use this method in an applet that determines and prints all the prime numbers less than 10,000. How many numbers up to 10,000 do you have to test to ensure that you have found all the primes? Display the results in a `JTextArea` that has scrolling functionality.

ANS:

```
1  // Exercise 6.26 Part B Solution: PrimeNum.java
2  // Program calculates prime numbers
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class PrimeNum extends JApplet {
8     String output;
9
10    public void init()
11    {
12       output = "Prime numbers between 1 and 10,000 are: ";
13
14       // test all numbers between 1 and 10000
15       for ( int m = 1; m <= 10000; m++ )
16          if ( prime( m ) == true )
17             output += "\n" + m;
18
19       // create components for display and add to applet
20       JTextArea outputArea = new JTextArea( 15, 20 );
21       JScrollPane scroller = new JScrollPane( outputArea );
22       outputArea.setText( output );
23
24       Container container = getContentPane();
25       container.add( scroller );
26    }
27
28    // a helper method for determining if a number is prime
29    // (This is the solution to 6.26, part A.)
30    public boolean prime( int n )
31    {
32       for ( int v = 2; v <= n / 2; v++ )
33          if ( n % v == 0 )
34             return false;
35
36       return true;
37    }
38
39 }  // end class PrimeNum
```

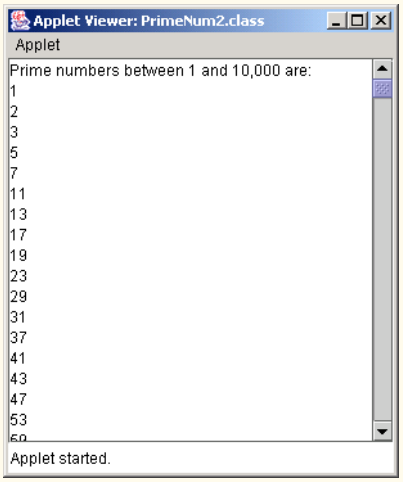


c) Initially, you might think that $n/2$ is the upper limit for which you must test to see whether a number is prime, but you need only go as high as the square root of n. Why? Rewrite the program, and run it both ways. Estimate the performance improvement.

ANS:

```
1  // Exercise 6.26 Part C Solution: PrimeNum2.java
2  // Program calculates prime numbers efficiently
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class PrimeNum2 extends JApplet {
8     String output;
9
10    public void init()
11    {
12       output = "Prime numbers between 1 and 10,000 are: ";
13
14       // test all numbers between 1 and 10000
15       for ( int m = 1; m <= 10000; m++ )
16          if ( prime( m ) == true )
17             output += "\n" + m;
18
19       // create components for display and add to applet
20       JTextArea outputArea = new JTextArea( 15, 20 );
21       JScrollPane scroller = new JScrollPane( outputArea );
22       outputArea.setText( output );
23
24       Container container = getContentPane();
25       container.add( scroller );
26    }
27
28    // a helper method for determining if a number is prime
29    // (This is the solution to 6.26, part A.)
```

```
30      public boolean prime( int n )
31      {
32         // max number to test is the square of n
33         int max = ( int ) Math.sqrt( n );
34
35         for ( int v = 2; v <= max; v++ )
36            if ( n % v == 0 )
37               return false;
38
39         return true;
40      }
41
42   }  // end class PrimeNum2
```

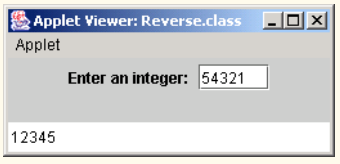


6.27 Write a method that takes an integer value and returns the number with its digits reversed. For example, given the number 7631, the method should return 1367. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

```
1  // Exercise 6.27 Solution: Reverse.java
2  // Program takes a four digit number
3  // and prints out its digits reversed.
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class Reverse extends JApplet implements ActionListener {
10    final int SIZE = 5;
11    JLabel prompt;
12    JTextField input;
13
```

```
14      // obtain value from user
15      public void init()
16      {
17         prompt = new JLabel( "Enter an integer: " );
18         input = new JTextField( SIZE );
19         input.addActionListener( this );
20         Container container = getContentPane();
21         container.setLayout( new FlowLayout() );
22         container.add( prompt );
23         container.add( input );
24      }
25
26      // perform action associated with pressing enter
27      public void actionPerformed( ActionEvent e )
28      {
29         int number = Integer.parseInt( input.getText() );
30
31         reverseDigits( number );
32      }
33
34      // print parameter number with digits reversed
35      public void reverseDigits( int number )
36      {
37         int oldPlace, newPlace = 1, temp, value;
38         String reverseNumber = "";
39
40         // create largest required devisor
41         for ( value = 1; value <= number; value *= 10 );
42         value /= 10;
43
44         oldPlace = value;
45
46         for ( int count = value; count > 0; count /= 10 ) {
47            temp = number / oldPlace;
48            reverseNumber =  temp + reverseNumber;
49            number %= oldPlace;
50            oldPlace /= 10;
51            newPlace *= 10;
52         }
53
54         showStatus( reverseNumber );
55      }
56
57   }  // end class Reverse
```

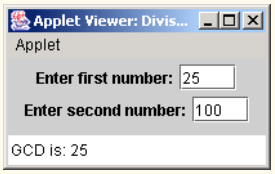


6.28 The *greatest common divisor* (*GCD*) of two integers is the largest integer that evenly divides each of the two numbers. Write a method `gcd` that returns the greatest common divisor of two integers. Incorporate the method into an applet that reads two values from the user. Display the result of the method in the status bar.

ANS:

```
1  // Exercise 6.28 Solution: Divisor.java
2  // Program finds the greatest common divisor of two numbers.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Divisor extends JApplet implements ActionListener {
9     JTextField input1, input2;
10    JLabel label1, label2;
11
12    public void init()
13    {
14       // create labels and text fields
15       label1 = new JLabel( "Enter first number:" );
16       label2 = new JLabel( "Enter second number:" );
17       input1 = new JTextField( 4 );
18       input2 = new JTextField( 4 );
19       input2.addActionListener( this );
20
21       // add components to applet
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24       container.add( label1 );
25       container.add( input1 );
26       container.add( label2 );
27       container.add( input2 );
28    }
29
30    // retrieve user input
31    public void actionPerformed( ActionEvent e )
32    {
33       int num1 = Integer.parseInt( input1.getText() );
34       int num2 = Integer.parseInt( input2.getText() );
35
36       showStatus( "GCD is: " + gcd( num1, num2 ) );
37    }
38
39    // calculate the greatest common divisor
40    public int gcd( int x, int y )
41    {
42       int greatest = 1;
43
44       // determine if x or y is larger
45       int smaller = ( x < y ) ? x : y;
46
47       // test all numbers up to smaller to see if
48       // they are divisors of both x and y
```

```
49          for ( int z = 2; z <= smaller; z++ )
50             if ( ( x % z == 0 ) && ( y % z == 0 ) )
51                greatest = z;
52
53          return greatest;
54       }
55
56    }  // end class Divisor
```

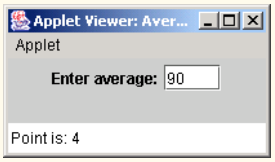


6.29 Write a method `qualityPoints` that inputs a student's average and returns 4 if the student's average is 90–100, 3 if the average is 80–89, 2 if the average is 70–79, 1 if the average is 60–69 and 0 if the average is lower than 60. Incorporate the method into an applet that reads a value from the user. Display the result of the method in the status bar.

ANS:

```
1  // Exercise 6.29 Solution: Average.java
2  // Program displays a number
3  // representing the student's average.
4
5  import java.awt.event.*;
6  import java.awt.*;
7  import javax.swing.*;
8
9  public class Average extends JApplet implements ActionListener {
10    JTextField inputField;
11    JLabel prompt;
12
13    // set up GUI components
14    public void init()
15    {
16       prompt = new JLabel( "Enter average:" );
17       inputField = new JTextField( 4 );
18       inputField.addActionListener( this );
19
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22       container.add( prompt );
23       container.add( inputField );
24
25    }  // end method init
26
27    // call method qualityPoints if user input is within range
28    public void actionPerformed( ActionEvent actionEvent )
29    {
30       int inputNumber = Integer.parseInt( inputField.getText() );
```

```
31
32          if ( inputNumber >= 0 && inputNumber <= 100 )
33             showStatus( "Point is: " + qualityPoints( inputNumber ) );
34          else
35             showStatus( "Invalid input." );
36
37       }  // end method actionPerformed
38
39       // return single digit value of grade
40       public int qualityPoints( int grade )
41       {
42          if ( grade >= 90 )
43             return 4;
44
45          else if ( grade >= 80 )
46             return 3;
47
48          else if ( grade >= 70 )
49             return 2;
50
51          else if ( grade >= 60 )
52             return 1;
53
54          else
55             return 0;
56
57       }  // end method qualityPoints
58
59    }  // end class Average
```

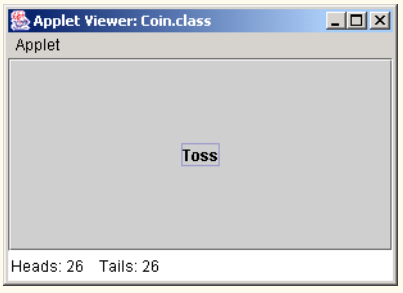


6.30 Write an applet that simulates coin tossing. Let the program toss a coin each time the user presses the "`Toss`" button. Count the number of times each side of the coin appears. Display the results. The program should call a separate method `flip` that takes no arguments and returns `false` for tails and `true` for heads. [*Note*: If the program realistically simulates coin tossing, each side of the coin should appear approximately half the time.]

ANS:

```
1  // Exercise 6.30 Solution: Coin.java
2  // Program simulates tossing a coin.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Coin extends JApplet implements ActionListener {
9     JButton button;
```

```
10      int heads, tails;
11
12      // set up GUI components
13      public void init()
14      {
15         heads = 0;
16         tails = 0;
17
18         button = new JButton( "Toss" );
19         button.addActionListener( this );
20         Container container = getContentPane();
21         container.add( button );
22      }
23
24      // display result of tossing coin
25      public void actionPerformed( ActionEvent e )
26      {
27         if ( flip() == true )
28            heads++;
29         else
30            tails++;
31
32         showStatus( "Heads: " + heads + "    Tails: " + tails );
33      }
34
35      // simulate flipping
36      public boolean flip()
37      {
38         if ( ( int ) ( Math.random() * 2 ) == 1 )
39            return true;
40         else
41            return false;
42      }
43
44   }  // end class Coin
```



6.31 Computers are playing an increasing role in education. Write a program that will help an elementary school student learn multiplication. Use `Math.random` to produce two positive one-digit integers. The program should then display a question in the status bar, such as

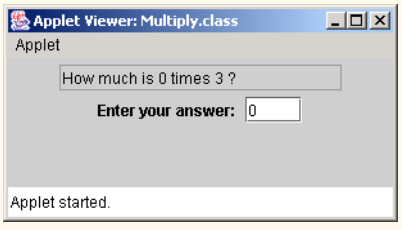
```
How much is 6 times 7?
```

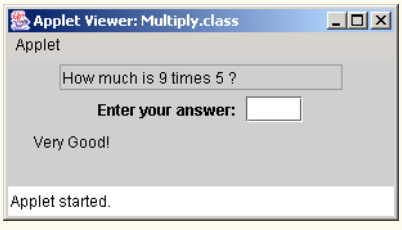
The student then types the answer into a `JTextField`. Next, the program checks the student's answer. If it is correct, draw the string `"Very good!"` on the applet and ask another multiplication question. If the answer is wrong, draw the string `"No. Please try again."` on the applet and let the student try the same question repeatedly until the student finally gets it right. A separate method should be used to generate each new question. This method should be called once when the applet begins execution and each time the user answers the question correctly. All drawing on the applet should be performed by the `paint` method.

ANS:

```
1  // Exercise 6.31 Solution: Multiply.java
2  // Program generates single digit multiplication problems
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Multiply extends JApplet implements ActionListener {
9     JTextField question, input;
10    JLabel prompt;
11    int answer, guess;
12    String questionString;
13
14    public void init()
15    {
16       // set guess to a flag value indicating no user input
17       guess = -999;
18
19       // create text fields and a label
20       question = new JTextField( 20 );
21       question.setEditable( false );
22
23       prompt = new JLabel( "Enter your answer: " );
24
25       input = new JTextField( 4 );
26       input.addActionListener( this );
27
28       // add components to applet
29       Container container = getContentPane();
30       container.setLayout( new FlowLayout() );
31       container.add( question );
32       container.add( prompt );
33       container.add( input );
34
35       // generate a question
36       createQuestion();
37    }
38
39    // show whether the answer was correct or not
40    public void paint( Graphics g ) {
41       super.paint( g );
42
43       // determine whether response is correct
44       // if guess isn't flag value
45       if ( guess != -999 ) {
```

```
46            if ( guess != answer )
47               g.drawString( "No. Please try again.", 20, 70 );
48            else {
49               g.drawString( "Very Good!", 20, 70 );
50               createQuestion();
51            }
52
53            guess = -999;
54         }
55      }
56
57      // verify the entered response
58      public void actionPerformed( ActionEvent e )
59      {
60         guess = Integer.parseInt( input.getText() );
61
62         // clear the text field
63         input.setText( "" );
64
65         // display the correct response
66         repaint();
67      }
68
69      // create a new question and a corresponding answer
70      public void createQuestion()
71      {
72         // get two random numbers between 0 and 9
73         int digit1 = ( int ) ( Math.random() * 10 );
74         int digit2 = ( int ) ( Math.random() * 10 );
75
76         answer = digit1 * digit2;
77         questionString = "How much is " + digit1 + " times " +
78            digit2 + " ?";
79
80         // add to applet
81         question.setText( questionString );
82      }
83
84   }  // end class Multiply
```





6.32 The use of computers in education is referred to as *computer-assisted instruction* (*CAI*). One problem that develops in CAI environments is student fatigue. This problem can be eliminated by varying the computer's dialogue to hold the student's attention. Modify the program of Exercise 6.31 so that the various comments are printed for each correct answer and each incorrect answer as follows:

Responses to a correct answer:

```
Very good!
Excellent!
Nice work!
Keep up the good work!
```

Responses to an incorrect answer:

```
No. Please try again.
Wrong. Try once more.
Don't give up!
No. Keep trying.
```

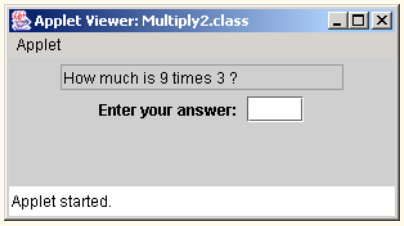
Use random-number generation to choose a number from 1 to 4 that will be used to select an appropriate response to each answer. Use a `switch` statement in the `paint` method to issue the responses.

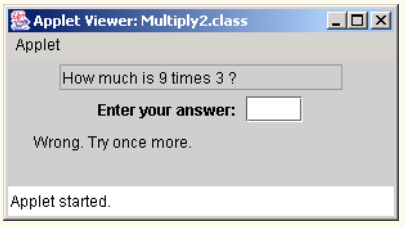
ANS:

```
1  // Exercise 6.32 Solution: Multiply2.java
2  // Program generates single digit multiplication
3  // problems with varying computer responses
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class Multiply2 extends JApplet implements ActionListener {
10    JTextField question, input, response;
11    JLabel prompt;
12    int answer, guess;
13    String questionString;
14
15    public void init()
16    {
17       // set guess to a flag value indicating no user input
18       guess = -999;
19
20       // create text fields and a label
21       question = new JTextField( 20 );
22       question.setEditable( false );
23
24       prompt = new JLabel( "Enter your answer: " );
25
26       input = new JTextField( 4 );
27       input.addActionListener( this );
28
29       // add components to applet
30       Container container = getContentPane();
```

```
31          container.setLayout( new FlowLayout() );
32          container.add( question );
33          container.add( prompt );
34          container.add( input );
35
36          // generate a question
37          createQuestion();
38       }
39
40       // show whether the answer was correct or not
41       public void paint( Graphics g ) {
42          super.paint( g );
43
44          // determine whether response is correct
45          // if guess isn't flag value
46          if ( guess != -999 ) {
47             if ( guess != answer )
48                g.drawString( createResponse( false ), 20, 70 );
49             else {
50                g.drawString( createResponse( true ), 20, 70 );
51                createQuestion();
52             }
53
54             guess = -999;
55          }
56       }
57
58       // verify the entered response
59       public void actionPerformed( ActionEvent e )
60       {
61          guess = Integer.parseInt( input.getText() );
62
63          // clear the text field
64          input.setText( "" );
65
66          // display the correct response
67          repaint();
68       }
69
70       // create a new question and a corresponding answer
71       public void createQuestion()
72       {
73          // get two random numbers between 0 and 9
74          int digit1 = ( int ) ( Math.random() * 10 );
75          int digit2 = ( int ) ( Math.random() * 10 );
76
77          answer = digit1 * digit2;
78          questionString = "How much is " + digit1 + " times " +
79             digit2 + " ?";
80
81          // add to applet
82          question.setText( questionString );
83       }
84
```

```
85       // create a new response
86       public String createResponse( boolean correct )
87       {
88          if ( correct )
89             switch ( ( int ) ( Math.random() * 4 ) ) {
90                case 0:
91                   return( "Very Good!" );
92
93                case 1:
94                   return( "Excellent!" );
95
96                case 2:
97                   return( "Nice work!" );
98
99                case 3:
100                  return( "Keep up the good work!" );
101            }
102
103         // otherwise, assume incorrect
104         switch ( ( int ) ( Math.random() * 4 ) ) {
105            case 0:
106               return( "No. Please try again." );
107
108            case 1:
109               return( "Wrong. Try once more." );
110
111            case 2:
112               return( "Don't give up!" );
113
114            case 3: default:
115               return( "No. Keep trying." );
116         }
117
118      }  // end method createResponse
119
120   }  // end class Multiply2
```





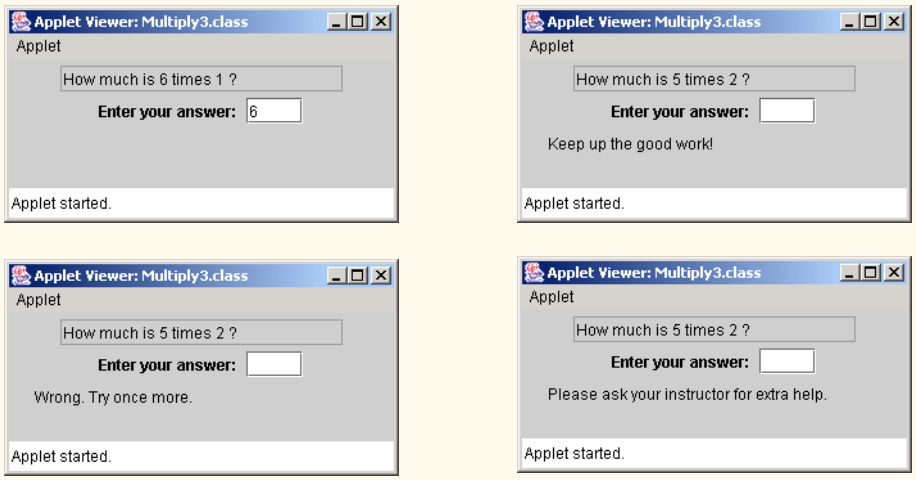
6.33 More sophisticated computer-assisted instruction systems monitor the student's performance over a period of time. The decision to begin a new topic is often based on the student's success with previous topics. Modify the program of Exercise 6.32 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage of correct responses. If the percentage is lower than 75%, print `Please ask your instructor for extra help` and reset the program so another student can try it.

ANS:

```
// Exercise 6.33 Solution: Multiply3.java
// Program generates single digit multiplication problems with
// varying instructor responses. It also provides a percentage feedback.

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Multiply3 extends JApplet implements ActionListener {

   // addition to monitor overall progress
   int count, right;

   JTextField question, input, response;
   JLabel prompt;
   int answer, guess;
   String questionString;

   public void init()
   {
      // set guess to a flag value indicating no user input
      guess = -999;
      count = 0;

      // create text fields and a label
      question = new JTextField( 20 );
      question.setEditable( false );

      prompt = new JLabel( "Enter your answer: " );

      input = new JTextField( 4 );
      input.addActionListener( this );

      // add components to applet
      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      container.add( question );
      container.add( prompt );
      container.add( input );

      // generate a question
      createQuestion();
   }

   // show whether the answer was correct or not
   public void paint( Graphics g ) {
      super.paint( g );

      // determine whether response is correct
      // if guess isn't flag value
      if ( guess != -999 ) {

```

```
53
54          if ( guess != answer ) {
55
56             // if student is struggling after 10 problem
57             if ( ( calculatePercentage() < 0.75 ) && ( count >= 10 ) ) {
58                g.drawString( "Please ask your instructor " +
59                   "for extra help.", 20, 70 );
60
61                // reset counters
62                right = 0;
63                count = 0;
64             }
65
66             else
67                g.drawString( createResponse( false ), 20, 70 );
68          }
69
70          // correct response
71          else {
72             right++;
73             g.drawString( createResponse( true ), 20, 70 );
74             createQuestion();
75          }
76
77          // reset flag to indicate that there is no input
78          guess = -999;
79       }
80    }
81
82    // verify the entered response
83    public void actionPerformed( ActionEvent e )
84    {
85       count++;
86       guess = Integer.parseInt( input.getText() );
87
88       // clear the text field
89       input.setText( "" );
90
91       // display the correct response
92       repaint();
93
94    }
95
96    // create a new question and a corresponding answer
97    public void createQuestion()
98    {
99       // get two random numbers between 0 and 9
100      int digit1 = ( int ) ( Math.random() * 10 );
101      int digit2 = ( int ) ( Math.random() * 10 );
102
103      answer = digit1 * digit2;
104      questionString = "How much is " + digit1 + " times " +
105         digit2 + " ?";
106
```

```
107         // add to applet
108         question.setText( questionString );
109      }
110
111      // create a new response
112      public String createResponse( boolean correct )
113      {
114         if ( correct )
115            switch ( ( int ) ( Math.random() * 4 ) ) {
116               case 0:
117                  return( "Very Good!" );
118
119               case 1:
120                  return( "Excellent!" );
121
122               case 2:
123                  return( "Nice work!" );
124
125               case 3:
126                  return( "Keep up the good work!" );
127            }
128
129         // otherwise, assume incorrect
130         switch ( ( int ) ( Math.random() * 4 ) ) {
131            case 0:
132               return( "No. Please try again." );
133
134            case 1:
135               return( "Wrong. Try once more." );
136
137            case 2:
138               return( "Don't give up!" );
139
140            case 3: default:
141               return( "No. Keep trying." );
142         }
143
144      }  // end method createResponse
145
146      // determine how the user is faring
147      public double calculatePercentage()
148      {
149         return ( double ) right / count;
150      }
151
152   }  // end class Multiply2
```

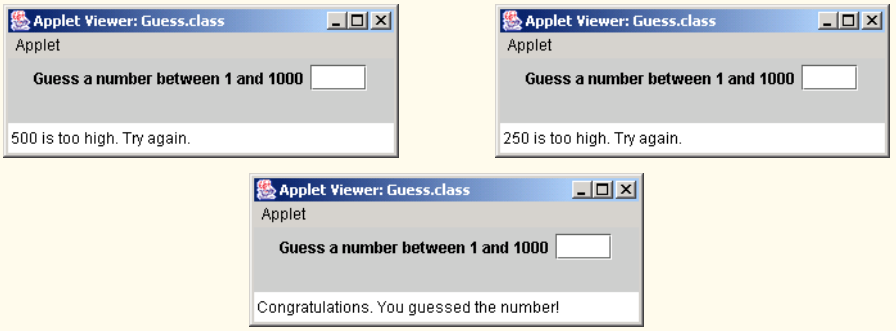


6.34 Write an applet that plays "guess the number" as follows: Your program chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The applet displays the prompt `Guess a number between 1 and 1000` next to a `JTextField`. The player types a first guess into the `JTextField` and presses the *Enter* key. If the player's guess is incorrect, your program should display `Too high. Try again.` or `Too low. Try again.` in the status bar to help the player "zero in" on the correct answer. The program should clear the `JTextField` so the user can enter the next guess. When the user enters the correct answer, display `Congratulations. You guessed the number!` in the status bar, and clear the `JTextField` so the user can play again. [*Note*: The guessing technique employed in this problem is similar to a *binary search*.]

ANS:

```
1  // Exercise 6.34 Solution: Guess.java
2  // Program plays guess the number.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Guess extends JApplet implements ActionListener {
9     JTextField input;
10    JLabel prompt;
11    int answer;
12
13    public void init()
14    {
15       // create components
16       input = new JTextField( 4 );
17       input.addActionListener( this );
18       prompt = new JLabel( "Guess a number between 1 and 1000" );
19
20       // add components to applet
21       Container container = getContentPane();
```

```
22          container.setLayout( new FlowLayout() );
23          container.add( prompt );
24          container.add( input );
25
26          // generate a new number
27          answer = getNumber();
28       }
29
30       // retrieve user input and compare to answer
31       public void actionPerformed( ActionEvent e )
32       {
33          int userGuess = Integer.parseInt( input.getText() );
34
35          checkUserGuess( userGuess );
36          input.setText( "" );
37       }
38
39       // create a new number to guess
40       public int getNumber()
41       {
42          return ( ( int ) ( 1 + Math.random() * 1000 ) );
43       }
44
45       // judge user input
46       public void checkUserGuess( int userGuess )
47       {
48          if ( userGuess < answer )
49             showStatus( userGuess + " is too low. Try again." );
50
51          else if ( userGuess > answer )
52             showStatus( userGuess + " is too high. Try again." );
53
54          else {
55             showStatus( "Congratulations. You guessed the number!" );
56             input.setText( "" );
57
58             // new search
59             answer = getNumber();
60          }
61       }
62
63    }  // end class Guess
```

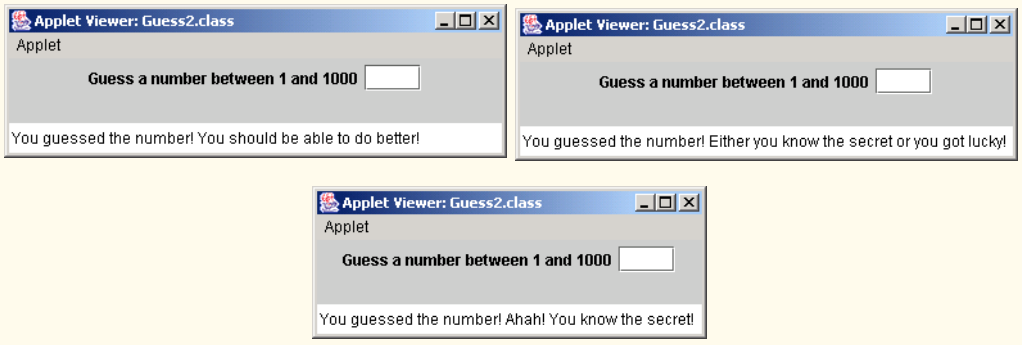


6.35 Modify the program of Exercise 6.34 to count the number of guesses the player makes. If the number is 10 or fewer, print `Either you know the secret or you got lucky!` If the player guesses the number in 10 tries, print `Aha! You know the secret!` If the player makes more than 10 guesses, print `You should be able to do better!` Why should it take no more than 10 guesses? Well, with each "good guess," the player should be able to eliminate half of the numbers. Now show why any number from 1 to 1000 can be guessed in 10 or fewer tries.

ANS:

```
1  // Exercise 6.35 Solution: Guess2.java
2  // Program plays guess the number.
3
4  import javax.swing.*;
5  import java.awt.*;
6  import java.awt.event.*;
7
8  public class Guess2 extends JApplet implements ActionListener {
9     JTextField input;
10    JLabel prompt;
11    int answer, userGuess, count;
12    boolean firstTime;
13
14    public void init()
15    {
16       // create components
17       input = new JTextField( 4 );
18       input.addActionListener( this );
19       prompt = new JLabel( "Guess a number between 1 and 1000" );
20
21       // add components to applet
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24       container.add( prompt );
25       container.add( input );
26
27       // generate a new number
28       answer = getNumber();
29    }
30
```

```
31     // retrieve user input and compare to answer
32     public void actionPerformed( ActionEvent e )
33     {
34        int userGuess = Integer.parseInt( input.getText() );
35
36        ++count;
37        checkUserGuess( userGuess );
38        input.setText( "" );
39     }
40
41     // create a new number to guess
42     public int getNumber()
43     {
44        return ( ( int ) ( 1 + Math.random() * 1000 ) );
45     }
46
47     // judge user input
48     public void checkUserGuess( int userGuess )
49     {
50        if ( userGuess < answer )
51           showStatus( userGuess + " is too low. Try again." );
52
53        else if ( userGuess > answer )
54           showStatus( userGuess + " is too high. Try again." );
55
56        else {
57           displayMessage();
58
59           // new search
60           input.setText( "" );
61           count = 0;
62           answer = getNumber();
63        }
64     }
65
66     // print a message based on the number of tries
67     public void displayMessage()
68     {
69        if ( count < 10 )
70           showStatus( "You guessed the number! Either you " +
71              "know the secret or you got lucky!" );
72
73        else if ( count == 10 )
74           showStatus( "You guessed the number! Ahah! You " +
75              "know the secret!" );
76
77        else
78           showStatus( "You guessed the number! You should " +
79              "be able to do better!" );
80     }
81
82  }  // end class Guess2
```



6.36 Write a recursive method `power( base, exponent )` that, when called, returns

$$base^{exponent}$$

For example, `power( 3, 4 ) = 3 * 3 * 3 * 3`. Assume that `exponent` is an integer greater than or equal to 1. (*Hint*: The recursion step should use the relationship

$$base^{exponent} = base \cdot base^{exponent - 1}$$

and the terminating condition occurs when `exponent` is equal to `1`, because

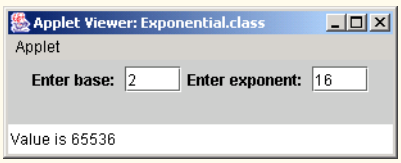
$$base^{1} = base$$

Incorporate this method into an applet that enables the user to enter the `base` and `exponent`.)

ANS:

```
1  // Exercise 6.36 Solution: Exponential.java
2  // Program calculates value of exponent.
3
4  import java.awt.event.*;
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class Exponential extends JApplet implements ActionListener {
9     JTextField baseInput, exponentInput;
10    JLabel basePrompt, exponentPrompt;
11
12    // set up GUI components
13    public void init()
14    {
15       basePrompt = new JLabel( "Enter base: " );
16       exponentPrompt = new JLabel( "Enter exponent: " );
17       baseInput = new JTextField( 4 );
18       exponentInput = new JTextField( 4 );
19       exponentInput.addActionListener( this );
20
21       Container container = getContentPane();
22       container.setLayout( new FlowLayout() );
23       container.add( basePrompt );
24       container.add( baseInput );
25       container.add( exponentPrompt );
```

```
26          container.add( exponentInput );
27       }
28
29       // call method power passing user input
30       public void actionPerformed( ActionEvent actionEvent )
31       {
32          int base = Integer.parseInt( baseInput.getText() );
33          int exponent = Integer.parseInt( exponentInput.getText() );
34
35          if ( exponent > 0 ) {
36             int result = power( base, exponent );
37             showStatus( "Value is " + result );
38          }
39          else
40             showStatus( "Invalid Exponent." );
41       }
42
43       // recursively calculate value of exponent.
44       public int power( int base, int exponent )
45       {
46          if ( exponent == 1 )
47             return base;
48          else
49             return base * power( base, exponent - 1 );
50       }
51
52    }  // end class Exponential
```



6.37 *(Towers of Hanoi)* Every budding computer scientist must grapple with certain classic problems, and the *Towers of Hanoi* (see Fig. 6.22) is one of the most famous. Legend has it that in a temple in the Far East, priests are attempting to move a stack of disks from one peg to another. The initial stack has 64 disks threaded onto one peg and arranged from bottom to top by decreasing size. The priests are attempting to move the stack from this peg to a second peg under the constraints that exactly one disk is moved at a time and at no time may a larger disk be placed above a smaller disk. A third peg is available for temporarily holding disks. Supposedly, the world will end when the priests complete their task, so there is little incentive for us to facilitate their efforts.

Let us assume that the priests are attempting to move the disks from peg 1 to peg 3. We wish to develop an algorithm that will print the precise sequence of peg-to-peg disk transfers.

If we were to approach this problem with conventional methods, we would rapidly find ourselves hopelessly knotted up in managing the disks. Instead, if we attack the problem with recursion in mind, it immediately becomes tractable. Moving n disks can be viewed in terms of moving only $n - 1$ disks (hence the recursion) as follows:

a) Move $n - 1$ disks from peg 1 to peg 2, using peg 3 as a temporary holding area.
b) Move the last disk (the largest) from peg 1 to peg 3.
c) Move the $n - 1$ disks from peg 2 to peg 3, using peg 1 as a temporary holding area.

The process ends when the last task involves moving $n = 1$ disk (i.e., the base case). This task is accomplished by simply moving the disk, without the need for a temporary holding area.

Write an applet to solve the Towers of Hanoi problem. Allow the user to enter the number of disks in a `JTextField`. Use a recursive `tower` method with four parameters:

a) the number of disks to be moved,
b) the peg on which these disks are initially threaded,
c) the peg to which this stack of disks is to be moved, and
d) the peg to be used as a temporary holding area.

Your program should display in a `JTextArea` with scrolling functionality the precise instructions it will take to move the disks from the starting peg to the destination peg. For example, to move a stack of three disks from peg 1 to peg 3, your program should print the following series of moves:

```
1 → 3 (This notation means "Move one disk from peg 1 to peg 3.")
1 → 2
3 → 2
1 → 3
2 → 1
2 → 3
1 → 3
```

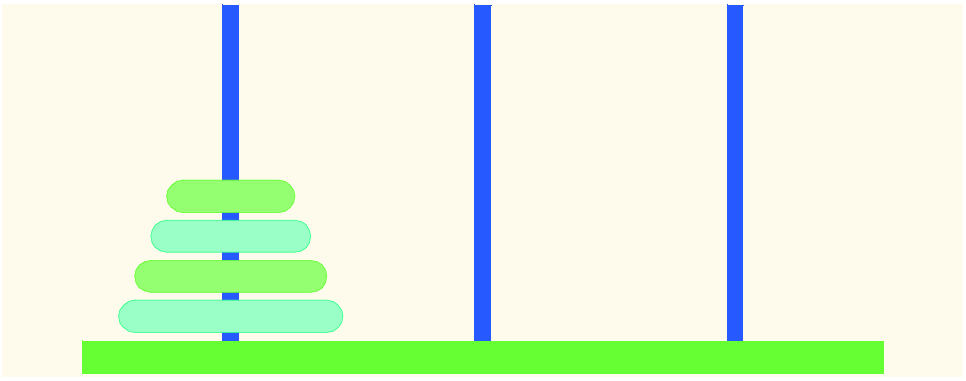
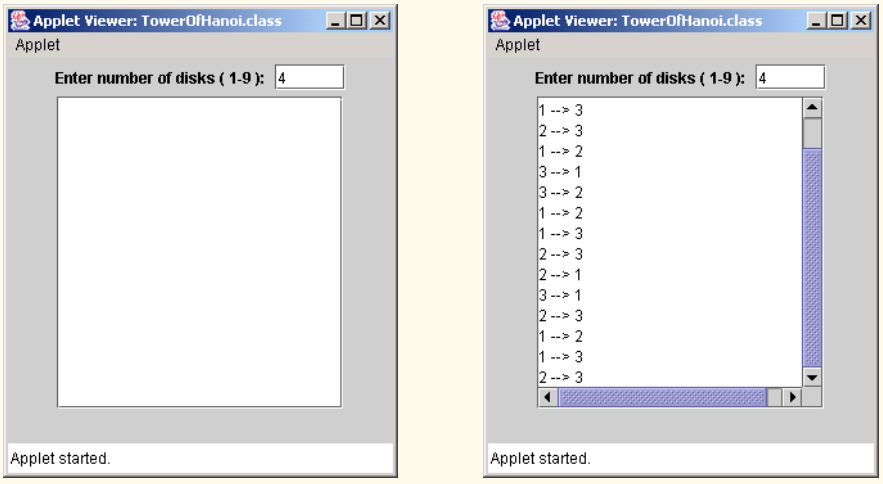


Fig. 6.22 The Towers of Hanoi for the case with four disks.

ANS:

```
1  // Exercise 6.37 Solution: TowerOfHanoi.java
2  // Program solves the towers of Hanoi problem, and
3  // demonstrates recursion
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class TowerOfHanoi extends JApplet implements ActionListener {
10    JLabel label;
11    JTextField input;
12    JTextArea outputArea;
13    String output;
```

```
14
15    public void init()
16    {
17       output = "";
18
19       // create components
20       label = new JLabel( "Enter number of disks ( 1-9 ): " );
21       input = new JTextField( 5 );
22       input.addActionListener( this );
23       outputArea = new JTextArea( 15, 20 );
24       JScrollPane scroller = new JScrollPane( outputArea );
25       outputArea.setText( output );
26
27       // add components to applet
28       Container container = getContentPane();
29       container.setLayout( new FlowLayout() );
30       container.add( label );
31       container.add( input );
32       container.add( scroller );
33    }
34
35    // recusively move disks through towers
36    public void tower( int disks, int peg1, int peg3, int peg2 )
37    {
38       if ( disks == 1 ) {
39          output += "\n" + peg1 + " --> " + peg3;
40          return;
41       }
42
43       // move ( disks - 1 ) disks from peg1 to peg2 recursively
44       tower( disks - 1, peg1, peg2, peg3 );
45
46       // move last disk from peg1 to peg3 recursively
47       output += "\n" + peg1 + " --> " + peg3;
48
49       // move ( disks - 1 ) disks from peg2 to peg3 recursively
50       tower( disks - 1, peg2, peg3, peg1 );
51    }
52
53    // actually sort the number of discs specified by user
54    public void actionPerformed( ActionEvent e )
55    {
56       output = "";
57
58       tower( Integer.parseInt( input.getText() ), 1, 3, 2 );
59       outputArea.setText( output );
60       repaint();
61    }
62
63 }  // end class TowerOfHanoi
```



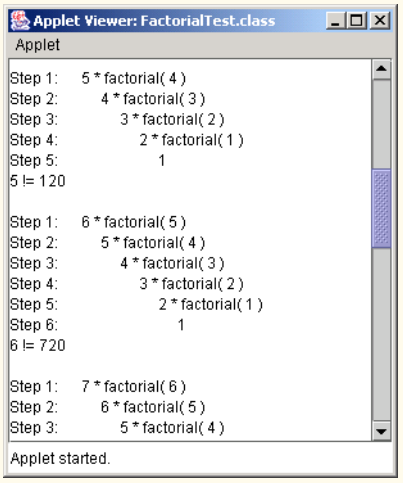
6.38 Any program that can be implemented recursively can be implemented iteratively, although sometimes with more difficulty and less clarity. Try writing an iterative version of the Towers of Hanoi. If you succeed, compare your iterative version with the recursive version you developed in Exercise 6.37. Investigate issues of performance, clarity and your ability to demonstrate the correctness of the programs.

6.39 (*Visualizing Recursion*) It is interesting to watch recursion "in action." Modify the factorial method of Fig. 6.15 to print its local variable and recursive-call parameter. For each recursive call, display the outputs on a separate line, and add a level of indentation. Do your utmost to make the outputs clear, interesting and meaningful. Your goal here is to design and implement an output format that helps a person understand recursion better. You may want to add such display capabilities to the many other recursion examples and exercises throughout the text.

ANS:

```
1  // Exercise 6.39: FactorialTest.java
2  // Recursive factorial method.
3
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class FactorialTest extends JApplet {
8     JTextArea outputArea;
9     int step;
10    long result;
11
12    // create GUI and calculate factorials of 0-10
13    public void init()
14    {
15       outputArea = new JTextArea();
16       JScrollPane scroller = new JScrollPane( outputArea );
17
18       Container container = getContentPane();
19       container.add( scroller );
20
```

```
21          // calculate the factorials of 0 through 10
22          for ( long counter = 0; counter <= 10; counter++ ) {
23             step = 0;
24             result = factorial( counter );
25             outputArea.append( counter + " != " + result + "\n\n" );
26          }
27
28       } // end method init
29
30       // recursive definition of method factorial
31       public long factorial( long number )
32       {
33          String indentation = "";
34          step++;
35
36          outputArea.append( "Step " + step + ": " );
37
38          for ( int i = 0; i < step; i++ )
39                indentation += "      ";
40
41          // base case
42          if ( number <= 1 ) {
43             outputArea.append( indentation + "1\n" );
44             return 1;
45          }
46
47          // recursive step
48          else {
49             outputArea.append( indentation + number +
50                " * factorial( " + ( number - 1 ) + " )\n" );
51             return number * factorial( number - 1 );
52          }
53       }
54
55    } // end class FactorialTest
```



6.40 The greatest common divisor of integers x and y is the largest integer that evenly divides into both x and y. Write a recursive method gcd that returns the greatest common divisor of x and y. The gcd of x and y is defined recursively as follows: If y is equal to 0, then gcd(x, y) is x; otherwise, gcd(x, y) is gcd(y, x % y), where % is the remainder operator. Use this method to replace the one you wrote in the applet of Exercise 6.28.

ANS:

```
1  // Exercise 6.40 Solution: Divisor.java
2  // Program recursively finds the greatest
3  // common divisor of two numbers.
4
5  import java.awt.event.*;
6  import java.awt.*;
7  import javax.swing.*;
8
9  public class Divisor extends JApplet implements ActionListener {
10    JTextField inputField1, inputField2;
11    JLabel label1, label2;
12
13    // set up GUI components
14    public void init()
15    {
16       label1 = new JLabel( "Enter first number:" );
17       label2 = new JLabel( "Enter second number:" );
18       inputField1 = new JTextField( 4 );
19       inputField2 = new JTextField( 4 );
20       inputField2.addActionListener( this );
21
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24       container.add( label1 );
25       container.add( inputField1 );
26       container.add( label2 );
27       container.add( inputField2 );
28    }
29
30    // display greatest common divisor of user input numbers
31    public void actionPerformed( ActionEvent actionEvent )
32    {
33       int input1, input2;
34
35       input1 = Integer.parseInt( inputField1.getText() );
36       input2 = Integer.parseInt( inputField2.getText() );
37       showStatus( "GCD is: " + gcd( input1, input2 ) );
38    }
39
40    // recursively calculate greatest common divisor
41    public int gcd( int x, int y )
42    {
43       if ( y == 0 )
44          return x;
45       else
46          return gcd( y, x % y );
47    }
```

```
48
49  } // end class Divisor
```

6.41 Exercise 6.31 through Exercise 6.33 developed a computer-assisted instruction program to teach an elementary school student multiplication. Perform the following enhancements:

a) Modify the program to allow the user to enter a school grade-level capability. A grade level of 1 means that the program should use only single-digit numbers in the problems, a grade level of 2 means that the program should use numbers as large as two digits, etc.

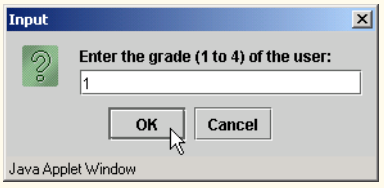
ANS:

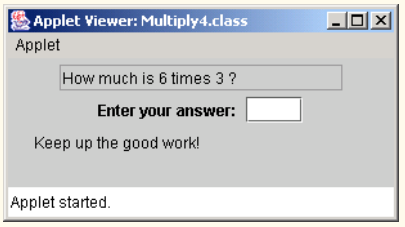
```
1  // Exercise 6.41a Solution: Multiply4.java
2  // Program generates single digit multiplication problems with
3  // varying instructor responses. It also provides a percentage feedback.
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.JOptionPane;
8  import javax.swing.*;
9
10 public class Multiply4 extends JApplet implements ActionListener {
11
12    // addition to monitor overall progress
13    int count, right, grade;
14
15    JTextField question, input, response;
16    JLabel prompt;
17    int answer, guess;
18    String questionString, gradeString;
19
20    public void init()
21    {
22       // set guess to a flag value indicating no user input
23       guess = -999;
24       count = 0;
25
26       // create text fields and a label
27       question = new JTextField( 20 );
28       question.setEditable( false );
29
30       prompt = new JLabel( "Enter your answer: " );
31
32       input = new JTextField( 4 );
33       input.addActionListener( this );
34
35       // add components to applet
36       Container container = getContentPane();
```

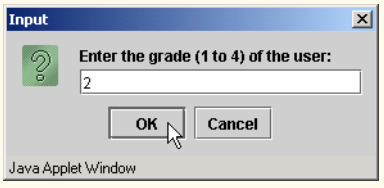
```
37          container.setLayout( new FlowLayout() );
38          container.add( question );
39          container.add( prompt );
40          container.add( input );
41
42          // promts the user to enter a grade level
43          // continues until it is in the range of 1 to 4
44          do {
45             gradeString = JOptionPane.showInputDialog(
46                "Enter the grade (1 to 4) of the user: " );
47             grade = Integer.parseInt( gradeString );
48          } while (( grade < 1 ) || ( grade > 4 ));
49
50          // generate a question
51          createQuestion();
52
53       } // end method init
54
55       // show whether the answer was correct or not
56       public void paint( Graphics g ) {
57          super.paint( g );
58
59          // determine whether response is correct
60          // if guess isn't flag value
61          if ( guess != -999 ) {
62
63             if ( guess != answer ) {
64
65                // if student is struggling after 10 problem
66                if ( ( calculatePercentage() < 0.75 ) && ( count >= 10 ) ) {
67                   g.drawString( "Please ask your instructor " +
68                      "for extra help.", 20, 70 );
69
70                   // reset counters
71                   right = 0;
72                   count = 0;
73                }
74
75                else
76                   g.drawString( createResponse( false ), 20, 70 );
77             }
78
79             // correct response
80             else {
81                right++;
82                g.drawString( createResponse( true ), 20, 70 );
83                createQuestion();
84             }
85
86             // reset flag to indicate that there is no input
87             guess = -999;
88          }
89
90       } // end method paint
```

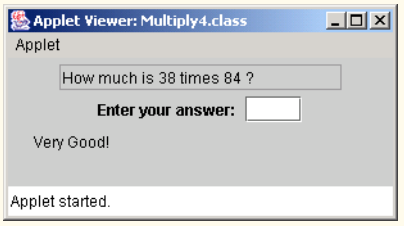
```
91
92       // verify the entered response
93       public void actionPerformed( ActionEvent e )
94       {
95          count++;
96          guess = Integer.parseInt( input.getText() );
97
98          // clear the text field
99          input.setText( "" );
100
101         // display the correct response
102         repaint();
103      }
104
105      // create a new question and a corresponding answer
106      public void createQuestion()
107      {
108         int gradeLvl = 1;
109         int digit1;
110         int digit2;
111
112         // figure out the user's level
113         // 1, 10, 100, or 1000; based on grade
114         for ( int i = 0; i < grade; i++ )
115             gradeLvl *= 10;
116
117         // get two random numbers between 0 and 9
118         digit1 = ( int ) ( Math.random() * gradeLvl );
119         digit2 = ( int ) ( Math.random() * gradeLvl );
120
121         answer = digit1 * digit2;
122         questionString = "How much is " + digit1 + " times " +
123            digit2 + " ?";
124
125         // add to applet
126         question.setText( questionString );
127      }
128
129      // create a new response
130      public String createResponse( boolean correct )
131      {
132         if ( correct )
133            switch ( ( int ) ( Math.random() * 4 ) ) {
134               case 0:
135                  return( "Very Good!" );
136
137               case 1:
138                  return( "Excellent!" );
139
140               case 2:
141                  return( "Nice work!" );
142
143               case 144               case 3:
145                  return( "Keep up the good work!" );
```

```
146
147         // otherwise, assume incorrect
148         switch ( ( int ) ( Math.random() * 4 ) ) {
149            case 0:
150               return( "No. Please try again." );
151
152            case 1:
153               return( "Wrong. Try once more." );
154
155            case 2:
156               return( "Don't give up!" );
157
158            case 3: default:
159               return( "No. Keep trying." );
160         }
161
162      }  // end method createResponse
163
164      // determine how the user is faring
165      public double calculatePercentage()
166      {
167         return ( double ) right / count;
168      }
169
170   }  // end class Multiply4
```









b) Modify the program to allow the user to pick the type of arithmetic problems he or she wishes to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only and 5 means a random mixture of problems of all these types.

ANS:

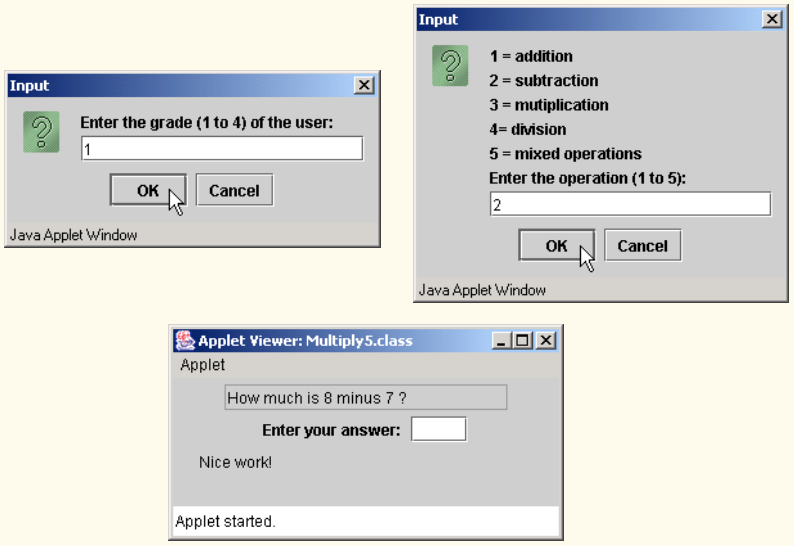
```
1  // Exercise 6.41b Solution: Multiply5.java
2  // Program generates single digit multiplication problems with
```

```
3  // varying instructor responses. It also provides a percentage feedback.
4
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.JOptionPane;
8  import javax.swing.*;
9
10 public class Multiply5 extends JApplet implements ActionListener {
11
12    // addition to monitor overall progress
13    int count, right, grade, operationNum;
14
15    JTextField question, input, response;
16    JLabel prompt;
17    int answer, guess;
18    String questionString, gradeString, operationString, operation;
19
20    public void init()
21    {
22       // set guess to a flag value indicating no user input
23       guess = -999;
24       count = 0;
25
26       // create text fields and a label
27       question = new JTextField( 20 );
28       question.setEditable( false );
29
30       prompt = new JLabel( "Enter your answer: " );
31
32       input = new JTextField( 4 );
33       input.addActionListener( this );
34
35       // add components to applet
36       Container container = getContentPane();
37       container.setLayout( new FlowLayout() );
38       container.add( question );
39       container.add( prompt );
40       container.add( input );
41
42       // promts the user to enter a grade level
43       // continues until it is in the range of 1 to 4
44       do {
45          gradeString = JOptionPane.showInputDialog(
46             "Enter the grade (1 to 4) of the user: " );
47          grade = Integer.parseInt( gradeString );
48       } while (( grade < 1 ) || ( grade > 4 ));
49
50       // promts the user to enter an operation level
51       // continues until it is in the range of 1 to 5
52       do {
53          operationString = JOptionPane.showInputDialog(
54             "1 = addition\n2 = subtraction\n3 = mutiplication\n" +
55             "4= division\n5 = mixed operations\n" +
56             "Enter the operation (1 to 5): " );
```

```
57             operationNum = Integer.parseInt( operationString );
58          } while (( operationNum < 1 ) || ( operationNum > 5 ));
59
60          // generate a question
61          createQuestion();
62
63       } // end method init
64
65       // show whether the answer was correct or not
66       public void paint( Graphics g )
67       {
68          super.paint( g );
69
70          // determine whether response is correct
71          // if guess isn't flag value
72          if ( guess != -999 ) {
73
74             if ( guess != answer ) {
75
76                // if student is struggling after 10 problem
77                if ( ( calculatePercentage() < 0.75 ) && ( count >= 10 ) ) {
78                   g.drawString( "Please ask your instructor " +
79                      "for extra help.", 20, 70 );
80
81                   // reset counters
82                   right = 0;
83                   count = 0;
84                }
85
86                else
87                   g.drawString( createResponse( false ), 20, 70 );
88             }
89
90             // correct response
91             else {
92                right++;
93                g.drawString( createResponse( true ), 20, 70 );
94                createQuestion();
95             }
96
97             // reset flag to indicate that there is no input
98             guess = -999;
99          }
100
101      } // end method paint
102
103      // verify the entered response
104      public void actionPerformed( ActionEvent e )
105      {
106         count++;
107         guess = Integer.parseInt( input.getText() );
108
109         // clear the text field
110         input.setText( "" );
```

```
111
112        // display the correct response
113        repaint();
114     }
115
116     // create a new question and a corresponding answer
117     public void createQuestion()
118     {
119        int gradeLvl = 1;
120        int digit1;
121        int digit2;
122
123        // figure out the user's level
124        // 1, 10, 100, or 1000; based on grade
125        for ( int i = 0; i < grade; i++ )
126            gradeLvl *= 10;
127
128        // get two random numbers based on the users level
129        digit1 = ( int ) ( Math.random() * gradeLvl );
130        digit2 = ( int ) ( Math.random() * gradeLvl );
131
132        getOperation( operationNum, digit1, digit2 );
133
134        questionString = "How much is " + digit1 + operation +
135           digit2 + " ?";
136
137        // add to applet
138        question.setText( questionString );
139     }
140
141     public void getOperation( int number, int digit1, int digit2 ) {
142        switch ( number ) {
143           case 1:
144              operation = " plus ";
145              answer = digit1 + digit2;
146              return;
147
148           case 2:
149              operation = " minus ";
150              answer = digit1 - digit2;
151              return;
152
153           case 3:
154              operation = " times ";
155              answer = digit1 * digit2;
156              return;
157
158           case 4:
159              if ( digit2 == 0 )
160                  digit2 = 1;
161              operation = " divided by ";
162              answer = digit1 / digit2;
163              return;
164
```

```
165          case 5:
166             int randomOperation = ( ( int ) ( 1 + Math.random() * 4 ) );
167             getOperation( randomOperation, digit1, digit2 );
168             return;
169
170       } // end switch
171
172    } // end method getOperation
173
174    // create a new response
175    public String createResponse( boolean correct )
176    {
177       if ( correct )
178          switch ( ( int ) ( Math.random() * 4 ) ) {
179             case 0:
180                return( "Very Good!" );
181
182             case 1:
183                return( "Excellent!" );
184
185             case 2:
186                return( "Nice work!" );
187
188             case 3:
189                return( "Keep up the good work!" );
190          }
191
192       // otherwise, assume incorrect
193       switch ( ( int ) ( Math.random() * 4 ) ) {
194          case 0:
195             return( "No. Please try again." );
196
197          case 1:
198             return( "Wrong. Try once more." );
199
200          case 2:
201             return( "Don't give up!" );
202
203          case 3: default:
204             return( "No. Keep trying." );
205       }
206
207    }  // end method createResponse
208
209    // determine how the user is faring
210    public double calculatePercentage()
211    {
212       return ( double ) right / count;
213    }
214
215 }  // end class Multiply5
```

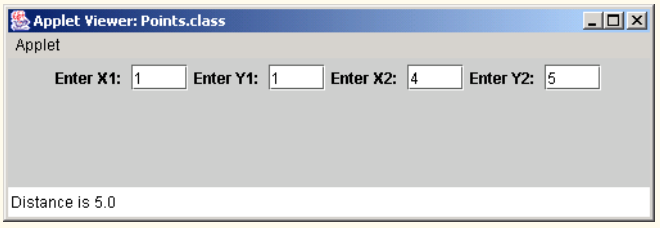


6.42 Write method `distance`, to calculate the distance between two points (*x1*, *y1*) and (*x2*, *y2*). All numbers and return values should be of type `double`. Incorporate this method into an applet that enables the user to enter the coordinates of the points.

ANS:

```
1  // Exercise 6.42 Solution: Points.java
2  // Program calculates the distance between two points.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Points extends JApplet implements ActionListener {
9     JTextField x1Input, x2Input, y1Input, y2Input;
10    JLabel labelX1, labelY1, labelX2, labelY2;
11
12    // set up GUI components
13    public void init()
14    {
15       labelX1 = new JLabel( "Enter X1: " );
16       labelY1 = new JLabel( "Enter Y1: " );
17       labelX2 = new JLabel( "Enter X2: " );
18       labelY2 = new JLabel( "Enter Y2: " );
19       x1Input = new JTextField( 4 );
20       x2Input = new JTextField( 4 );
21       y1Input = new JTextField( 4 );
22       y2Input = new JTextField( 4 );
23       y2Input.addActionListener( this );
24
25       Container container = getContentPane();
26       container.setLayout( new FlowLayout() );
```

```
27          container.add( labelX1 );
28          container.add( x1Input );
29          container.add( labelY1 );
30          container.add( y1Input );
31          container.add( labelX2 );
32          container.add( x2Input );
33          container.add( labelY2 );
34          container.add( y2Input );
35       }
36
37       // display distance between user input points
38       public void actionPerformed( ActionEvent e )
39       {
40          double x1, y1, x2, y2;
41
42          // read in two points
43          x1 = Double.parseDouble( x1Input.getText() );
44          y1 = Double.parseDouble( y1Input.getText() );
45          x2 = Double.parseDouble( x2Input.getText() );
46          y2 = Double.parseDouble( y2Input.getText() );
47
48          double theDistance = distance( x1, y1, x2, y2 );
49          showStatus( "Distance is " + theDistance );
50       }
51
52       // calculate distance between two points
53       public double distance( double x1, double y1, double x2, double y2 )
54       {
55          return Math.sqrt( Math.pow( ( x1 - x2 ), 2 ) +
56             Math.pow( ( y1 - y2 ), 2 ) );
57       }
58
59    } // end class Points
```



6.43 What does the following method do?

```
// Parameter b must be a positive
// integer to prevent infinite recursion
public int mystery( int a, int b )
{
   if ( b == 1 )
      return a;
   else
      return a + mystery( a, b - 1 );
}
```

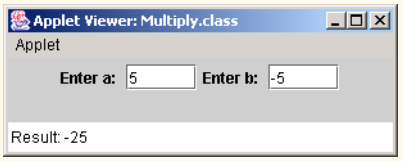
ANS: The method returns the equivalent value of a times b.

6.44 After you determine what the program in Exercise 6.43 does, modify the method to operate properly following the removal of the restriction that the second argument must be nonnegative. Also, incorporate the method into an applet that enables the user to enter two integers. Test the method.

ANS:

```
1  // Exercise 6.44 Solution: Multiply.java
2  // Program that does multiplication using recursion.
3
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Multiply extends JApplet implements ActionListener {
9     JLabel aLabel, bLabel;
10    JTextField aInput, bInput;
11
12    public void init()
13    {
14       // create components
15       aLabel = new JLabel( "Enter a: " );
16       aInput = new JTextField( 5 );
17       bLabel = new JLabel( "Enter b: " );
18       bInput = new JTextField( 5 );
19       bInput.addActionListener( this );
20
21       // add components to applet
22       Container container = getContentPane();
23       container.setLayout( new FlowLayout() );
24       container.add( aLabel );
25       container.add( aInput );
26       container.add( bLabel );
27       container.add( bInput );
28    }
29
30    // get the result
31    public void actionPerformed( ActionEvent e )
32    {
33       int a = Integer.parseInt( aInput.getText() );
34       int b = Integer.parseInt( bInput.getText() );
35       showStatus( "Result: " + mystery( a, b ) );
36    }
37
38    public int mystery( int a, int b )
39    {
40       // b is positive
41       if ( b < 0 ) {
42
43          if ( b == -1 )
44             return 0 - a;
45          else
46             return 0 - a + mystery( a, b + 1 );
47       }
```

```
48
49          // b is negative
50          else if ( b > 0 ) {
51
52             if ( b == 1 )
53                return a;
54             else
55                return a + mystery( a, b - 1 );
56          }
57
58          // b is zero
59          else
60             return 0;
61       }
62
63    }  // end class Multiply
```



6.45 Find the error in the following recursive method, and explain how to correct it:

```
public int sum( int n )
{
   if ( n == 0 )
      return 0;
   else
      return n + sum( n );
}
```

ANS: The method never reaches the base case. The recursive step should be:

```
return n + sum( n - 1 );
```

6.46 Modify the craps program of Fig. 6.9 to allow wagering. Initialize variable `bankBalance` to 1000 dollars. Prompt the player to enter a `wager`. Check that `wager` is less than or equal to `bankBalance`, and if not, have the user reenter `wager` until a valid `wager` is entered. After a correct `wager` is entered, run one game of craps. If the player wins, increase `bankBalance` by `wager` and print the new `bankBalance`. If the player loses, decrease `bankBalance` by `wager`, print the new `bankBalance`, check whether `bankBalance` has become zero and, if so, print the message `"Sorry. You busted!"` As the game progresses, print various messages to create some "chatter," such as `"Oh, you're going for broke, huh?"` or `"Aw c'mon, take a chance!"` or `"You're up big. Now's the time to cash in your chips!"`. Implement the "chatter" as a separate method that randomly chooses the string to display.

ANS:

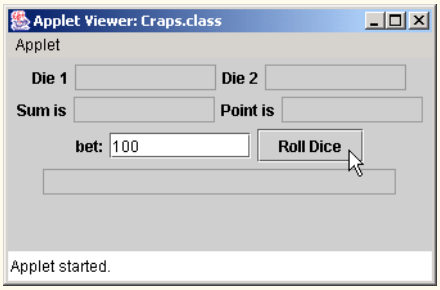
```
1  // Exercise 6.46 Solution: Craps.java
2  // Program plays Craps
3
4  import java.awt.*;
```

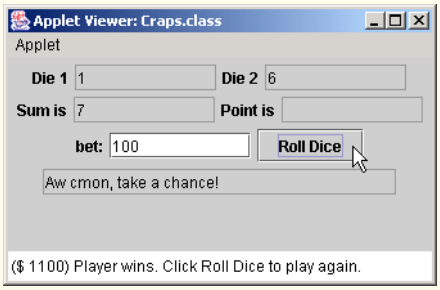
```
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Craps extends JApplet implements ActionListener {
9     boolean firstRoll = true;  // true if first roll
10    int dieSum;                // sum of the dice
11    int myPoint;          // point if no win/loss on first roll
12    int gameStatus;
13    int bankBalance, wager;
14
15    // graphical user interface components
16    JLabel die1Label, die2Label, sumLabel, pointLabel, betLabel;
17    JTextField firstDie, secondDie, sum, point, better, chatter;
18    JButton roll;
19
20    // constant variables for status of game
21    final int WON = 0, LOST = 1, CONTINUE = 2;
22
23    // setup graphical user interface components
24    public void init()
25    {
26       bankBalance = 1000;
27       gameStatus = CONTINUE;
28
29       // create GUI components
30       betLabel = new JLabel( "bet:" );
31       better = new JTextField( "100", 10 );
32       die1Label = new JLabel( "Die 1" );
33       firstDie = new JTextField( 10 );
34       firstDie.setEditable( false );
35       die2Label = new JLabel( "Die 2" );
36       secondDie = new JTextField( 10 );
37       secondDie.setEditable( false );
38       sumLabel = new JLabel( "Sum is" );
39       sum = new JTextField( 10 );
40       sum.setEditable( false );
41       roll = new JButton( "Roll Dice" );
42       roll.addActionListener( this );
43       pointLabel = new JLabel( "Point is" );
44       point = new JTextField( 10 );
45       point.setEditable( false );
46       chatter = new JTextField( 25 );
47       chatter.setEditable( false );
48
49       // add components to container
50       Container container = getContentPane();
51       container.setLayout( new FlowLayout() );
52       container.add( die1Label );
53       container.add( firstDie );
54       container.add( die2Label );
55       container.add( secondDie );
56       container.add( sumLabel );
57       container.add( sum );
58       container.add( pointLabel );
```

```
59          container.add( point );
60          container.add( betLabel );
61          container.add( better );
62          container.add( roll );
63          container.add( chatter );
64       }
65
66       // process one roll of the dice
67       public void play()
68       {
69          // first roll of the dice
70          if ( firstRoll ) {
71             dieSum = rollDice();
72
73             switch ( dieSum ) {
74
75                // win on first roll
76                case 7: case 11:
77                   gameStatus = WON;
78
79                   // clear point text field and allow restart
80                   point.setText( "" );
81                   firstRoll = true;
82                   break;
83
84                // lose on first roll
85                case 2: case 3: case 12:
86                   gameStatus = LOST;
87                   point.setText( "" );  // clear point text field
88                   firstRoll = true;     // allow new game to start
89                   break;
90
91                // remember point
92                default:
93                   gameStatus = CONTINUE;
94                   myPoint = dieSum;
95                   point.setText( Integer.toString( myPoint ) );
96                   firstRoll = false;
97                   break;
98             }
99          }
100
101         // after first roll
102         else {
103            dieSum = rollDice();
104
105            // win by making point
106            if ( dieSum == myPoint )
107               gameStatus = WON;
108
109            // lose by rolling 7
110            else if ( dieSum == 7 )
111               gameStatus = LOST;
112         }
```

```
113
114       // game still in play
115       if ( gameStatus == CONTINUE )
116          showStatus( "($ " + bankBalance + ") Roll again." );
117
118       else {
119
120          // game won
121          if ( gameStatus == WON ) {
122             bankBalance += wager;
123             showStatus( "($ " + bankBalance + ") Player wins. " +
124                         "Click Roll Dice to play again." );
125          }
126
127          // game lost
128          else {
129             bankBalance -= wager;
130             checkBalance();
131             showStatus( "($ " + bankBalance + ") Player loses. " +
132                         "Click Roll Dice to play again." );
133          }
134
135          // restart game
136          better.setEditable( true );
137          firstRoll = true;
138       }
139    }
140
141    // provides feedback if the balance is 0
142    void checkBalance()
143    {
144       if ( bankBalance <= 0 ) {
145          System.out.println( "Sorry. You busted!" );
146          System.exit( 0 );
147       }
148    }
149
150    // call method play when button is clicked
151    public void actionPerformed( ActionEvent e )
152    {
153       int wage = Integer.parseInt( better.getText() );
154
155       if ( wage > bankBalance || wage < 0 )
156          showStatus( "( $" + bankBalance + " ) " +
157                      "Enter a valid wager!" );
158       else {
159          wager = wage;
160          better.setEditable( false );
161          play();
162       }
163
164       chatter.setText( chatter() );
165    }
166
```

```
167    // roll the dice
168    public int rollDice()
169    {
170       int die1, die2, workSum;
171
172       die1 = 1 + ( int ) ( Math.random() * 6 );
173       die2 = 1 + ( int ) ( Math.random() * 6 );
174       workSum = die1 + die2;
175
176       firstDie.setText( Integer.toString( die1 ) );
177       secondDie.setText( Integer.toString( die2 ) );
178       sum.setText( Integer.toString( workSum ) );
179
180       return workSum;
181    }
182
183    // randomly chooses a phrase to respond to the player's action
184    public String chatter()
185    {
186       String statement = null;
187
188       switch ( ( int ) ( Math.random() * 5 ) ) {
189          case 0:
190             statement = "Oh, you're going for broke huh?";
191             break;
192
193          case 1:
194             statement = "Aw cmon, take a chance!";
195             break;
196
197          case 2:
198             statement = "You're up big. Now's the " +
199                "time to cash in your chips!";
200             break;
201
202          case 3:
203             statement = "You're way too lucky! I think you're a cheat!!!";
204             break;
205
206          case 4:
207             statement = "I'm betting all my money on you.";
208             break;
209       }
210
211       return statement;
212    }
213
214 }  // end class Craps
```





7

Arrays

Objectives

- To introduce the array data structure.
- To understand the use of arrays to store, sort and search lists and tables of values.
- To understand how to declare an array, initialize an array and refer to individual elements of an array.
- To be able to pass arrays to methods.
- To be able to declare and manipulate multidimensional arrays.

With sobs and tears he sorted out
Those of the largest size …
Lewis Carroll

Attempt the end, and never stand to doubt;
Nothing's so hard, but search will find it out.
Robert Herrick

Now go, write it before them in a table,
and note it in a book.
Isaiah 30:8

'Tis in my memory lock'd,
And you yourself shall keep the key of it.
William Shakespeare

SELF-REVIEW EXERCISES

7.1 Fill in the blank(s) in each of the following statements:

a) Lists and tables of values can be stored in _____.

ANS: arrays

b) The elements of an array are related by the fact that they have the same _____ and _____.

ANS: name, type

c) The number used to refer to a particular element of an array is called the element's _____.

ANS: index (or subscript or position number)

d) The process of placing the elements of an array in order is called _____ the array.

ANS: sorting

e) Determining whether an array contains a certain key value is called _____ the array.

ANS: searching

f) An array that uses two indices is referred to as a(n) _____ array.

ANS: two-dimensional

7.2 Determine whether each of the following is *true* or *false*. If *false*, explain why.

a) An array can store many different types of values.

ANS: False. An array can store only values of the same type.

b) An array index should normally be of type `float`.

ANS: False. An array index must be an integer or an integer expression.

c) An individual array element that is passed to a method and modified in that method will contain the modified value when the called method completes execution.

ANS: False. For individual primitive-type elements of an array: False. Such elements are passed by value. If a reference to an array is passed, then modifications to the array elements are reflected in the original. For individual elements of a nonprimitive type: True. Such elements are passed by reference, and changes to the object will be reflected in the original array element.

7.3 Perform the following tasks for an array called `fractions`:

a) Declare a constant `ARRAY_SIZE` that is initialized to 10.

ANS:
```
final int ARRAY_SIZE = 10;
```

b) Declare an array with `ARRAY_SIZE` elements of type `float`, and initialize the elements to 0.

ANS:
```
float fractions[] = new float[ ARRAY_SIZE ];
```

c) Name the fourth element of the array.

ANS:
```
fractions[ 3 ]
```

d) Refer to array element four.

ANS:
```
fractions[ 4 ]
```

e) Assign the value `1.667` to array element nine.

ANS:
```
fractions[ 9 ] = 1.667;
```

f) Assign the value `3.333` to the seventh element of the array.

ANS:
```
fractions[ 6 ] = 3.333;
```

g) Sum all the elements of the array, using a `for` statement. Declare the integer variable `x` as a control variable for the loop.

ANS:
```
float total = 0.0;
for ( int x = 0; x < fractions.length; x++ )
   total += fractions[ x ];
```

7.4 Perform the following tasks for an array called `table`:

a) Declare and create the array as an integer array that has three rows and three columns. Assume that the constant `ARRAY_SIZE` has been declared to be `3`.

ANS:
```
int table[][] = new int[ ARRAY_SIZE ][ ARRAY_SIZE ];
```

b) How many elements does the array contain?

ANS: Nine.

c) Use a `for` statement to initialize each element of the array to the sum of its indices. Assume that the integer variables `x` and `y` are declared as control variables.

ANS:
```
for ( int x = 0; x < table.length; x++ )
   for ( int y = 0; y < table[ x ].length; y++ )
      table[ x ][ y ] = x + y;
```

7.5 Find and correct the error in each of the following program segments:

a)
```
final int ARRAY_SIZE = 5;
ARRAY_SIZE = 10;
```

ANS: Error: Assigning a value to a constant after it has been initialized.
Correction: Assign the correct value to the constant in a `final int ARRAY_SIZE` declaration or create another variable.

b)
```
Assume int b[] = new int[ 10 ];
for ( int i = 0; i <= b.length; i++ )
   b[ i ] = 1;
```

ANS: Error: Referencing an array element outside the bounds of the array (`b[10]`).
Correction: Change the <= operator to <.

c)
```
Assume int a[][] = { { 1, 2 }, { 3, 4 } };
   a[ 1, 1 ] = 5;
```

ANS: Error: Array indexing is performed incorrectly.
Correction: Change the statement to `a[ 1 ][ 1 ] = 5;`.

EXERCISES

7.6 Fill in the blanks in each of the following statements:

a) Java stores lists of values in _____.

ANS: arrays.

b) The elements of an array are related by the fact that they _____.

ANS: have the same name and type.

c) When referring to an array element, the position number contained within brackets is called a(n) _____.

ANS: subscript

d) The names of the four elements of one-dimensional array `p` are _____, _____, _____ and _____.

ANS: `p[ 0 ]`, `p[ 1 ]`, `p[ 2 ]`, and `p[ 3 ]`

e) Naming an array, stating its type and specifying the number of dimensions in the array is called _____ the array.

ANS: declaring and instantiating

f) The process of placing the elements of an array into either ascending or descending order is called _____.

ANS: sorting

g) In a two-dimensional array, the first index identifies the _____ of an element and the second index identifies the _____ of an element.

ANS: row, column

h) An *m*-by-*n* array contains _____ rows, _____ columns and _____ elements.

ANS: `m`, `n`, `m * n`

i) The name of the element in row 3 and column 5 of array `d` is _____.

ANS: `d[ 3 ][ 5 ]`

7.7 Determine whether each of the following is *true* or *false*. If *false*, explain why.

a) To refer to a particular location or element within an array, we specify the name of the array and the value of the particular element.

ANS: False. The name of the array and the subscript are specified.

b) An array declaration reserves space for the array.

ANS: False. Arrays must be dynamically allocated in Java.

c) To indicate that 100 locations should be reserved for integer array `p`, the programmer writes the declaration

```
p[ 100 ];
```

ANS: False. The correct declaration is `int p[] = new int[ 100 ];`

d) A Java program that initializes the elements of a 15-element array to zero must contain at least one `for` statement.

ANS: False. Numeric arrays are automatically initialized to zero. Also, a member initializer list can be used.

e) A Java program that totals the elements of a two-dimensional array must contain nested `for` statements.

ANS: False. It is possible to total the elements of a double-subscripted array by enumerating all the elements in an assignment statement.

7.8 Write Java statements to accomplish each of the following tasks:

a) Display the value of the seventh element of character array `f`.

ANS:
```
System.out.print( f[ 6 ] );
```

b) Initialize each of the five elements of one-dimensional integer array `g` to 8.

ANS:
```
int g[] = { 8, 8, 8, 8, 8 );
```

c) Total the 100 elements of floating-point array `c`.

ANS:
```
for ( int k = 0; k < c.length; k++ )
   total += c[ k ];
```

d) Copy 11-element array `a` into the first portion of array `b`, which contains 34 elements.

ANS:
```
for ( int j = 0; j < a.length; j++ )
   b[ j ] = a[ j ];
```

e) Determine and print the smallest and largest values contained in 99-element floating-point array `w`.

ANS:
```
// Assume small and large are properly
// declared and initialized.
for ( int i = 0;  i < w.length; i++ )
   if ( w[ i ] < small )
      small = w[ i ];
   else if ( w[ i ] > large )
      large = w[ i ];

System.out.println( small + "   " + large );
```

7.9 Consider a two-by-three integer array `t`.

a) Write a statement that declares and creates `t`.

ANS:
```
int t[][] = new int[ 2 ][ 3 ];
```

b) How many rows does `t` have?

ANS: two

c) How many columns does `t` have?
ANS: three
d) How many elements does `t` have?
ANS: six
e) Write the names of all the elements in the second row of `t`.
ANS: `t[ 1 ][ 0 ], t[ 1 ][ 1 ], t[ 1 ][ 2 ]`
f) Write the names of all the elements in the third column of `t`.
ANS: `t[ 0 ][ 2 ], t[ 1 ][ 2 ]`
g) Write a single statement that sets the element of `t` in row 1 and column 2 to zero.
ANS: `t[ 0 ][ 1 ] = 0;`
h) Write a series of statements that initializes each element of `t` to zero. Do not use a repetition statement.

```
ANS: t[ 0 ][ 0 ] = 0;
     t[ 0 ][ 1 ] = 0;
     t[ 0 ][ 2 ] = 0;
     t[ 1 ][ 0 ] = 0;
     t[ 1 ][ 1 ] = 0;
     t[ 1 ][ 2 ] = 0;
```

i) Write a nested `for` statement that initializes each element of `t` to zero.

```
ANS: for ( int j = 0; j < t.length; j++ )
        for ( int k = 0; k < t[ j ].length; k++ )
           t[ j ][ k ] = 0;
```

j) Write a nested `for` statement that inputs the values for the elements of `t` from the user.

```
ANS: for ( int r = 0; r < t.length; r++ )
        for ( int c = 0; c < t[ r ].length; c++ )
           t[ r ][ c ] = System.in.read();
```

k) Write a series of statements that determines and prints the smallest value in `t`.

```
ANS: // assume small is declared and initialized
     for ( int x = 0; x < t.length; x++ )
        for ( int y = 0; y < t[ x ].length; y++ )
           if ( t[ x ][ y ] < small )
              small = t[ x ][ y ];
```

l) Write a statement that displays the elements of the first row of `t`.

```
ANS: System.out.println( t[ 0 ][ 0 ] + " " + t[ 0 ][ 1 ] + " " +
        t[ 0 ][ 2 ] );
```

m) Write a statement that totals the elements of the third column of `t`.

```
ANS: total = t[ 0 ][ 2 ] + t[ 1 ][ 2 ];
```

n) Write a series of statements that prints the contents of `t` in neat, tabular format. List the column indices as headings across the top, and list the row indices at the left of each row.

```
ANS: System.out.println( "  0     1     2" );
     for ( int e = 0; e < t.length; e++ ) {
        System.out.print( e + " " );

        for ( int r = 0; r < t[ e ].length; r++ )
           System.out.print( t[ e ][ r ] + "     " );

        System.out.println();
     }
```

7.10 Use a one-dimensional array to solve the following problem: A company pays its salespeople on a commission basis. The salespeople receive $200 per week plus 9% of their gross sales for that week. For example, a salesperson who grosses $5000 in sales in a week receives $200 plus 9% of $5000, or a total of $650. Write an applet (using an array of counters) that determines how many of

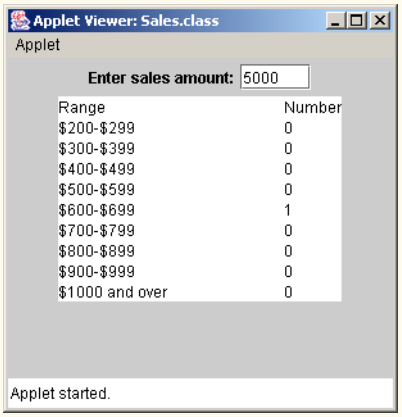
the salespeople earned salaries in each of the following ranges (assume that each salesperson's salary is truncated to an integer amount):

a) $200–299
b) $300–399
c) $400–499
d) $500–599
e) $600–699
f) $700–799
g) $800–899
h) $900–999
i) $1000 and over

ANS:

```
1  // Exercise 7.10 Solution: Sales.java
2  // Program calculates the amount of pay for a salesperson and counts the
3  // number of salespeople that earned salaries in given ranges.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Sales extends JApplet implements ActionListener {
9     JTextField inputField;
10    JTextArea outputArea;
11    JLabel prompt;
12    int total[];
13
14    // set up GUI components
15    public void init()
16    {
17       total = new int[ 9 ];
18
19       for ( int counter = 0; counter < total.length; counter++ )
20          total[ counter ] = 0;
21
22       prompt = new JLabel( "Enter sales amount:" );
23       inputField = new JTextField( 5 );
24       outputArea = new JTextArea();
25       inputField.addActionListener( this );
26
27       Container container = getContentPane();
28       container.setLayout( new FlowLayout() );
29       container.add( prompt );
30       container.add( inputField );
31       container.add( outputArea );
32
33    }  // end method init
34
35    // calculate which range salary is in
36    public void actionPerformed( ActionEvent actionEvent )
37    {
38       double dollars = Double.parseDouble( inputField.getText() );
39       double salary = dollars * 0.09 + 200;
40       int range = ( int ) ( salary / 100 );
```

```
41
42          if ( range > 9 )
43             range = 10;
44
45          ++total[ range - 2 ];
46          inputField.setText( "" );
47
48          // print chart in JTextArea
49          String output = "Range\t\tNumber";
50
51          for ( int range2 = 0; range2 < total.length - 1; range2++ )
52             output += "\n$" + (200 + 100 * range2) + "-$" +
53                (299 + 100 * range2) + "\t\t" + total[ range2 ] ;
54
55          output += "\n$1000 and over\t\t" + total[ total.length - 1 ];
56
57          outputArea.setText( output );
58
59       }  // end method actionPerformed
60
61    }  // end class Sales
```



The applet should use the GUI techniques introduced in Chapter 6. Display the results in a `JTextArea`. Use `JTextArea` method `setText` to update the results after each value is input by the user.

7.11 The bubble sort presented in Fig. 7.10 is inefficient for large arrays. Make the following simple modifications to improve the performance of the bubble sort:

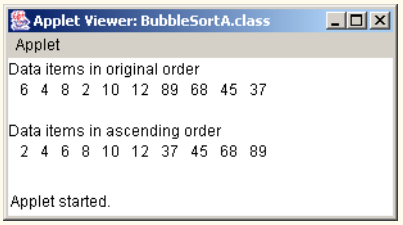
a) After the first pass, the largest number is guaranteed to be in the highest-numbered element of the array; after the second pass, the two highest numbers are "in place"; etc. Instead of making nine comparisons on every pass, modify the bubble sort to make eight comparisons on the second pass, seven on the third pass, etc.

ANS:

```
1  // Exercise 7.11 Part A Solution: BubbleSortA.java
2  // Applet sorts an array's values using an efficient bubble sort, so the
3  // number of comparisons will decrease by one after each sort.
```

```
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class BubbleSortA extends JApplet {
8
9     // initialize applet
10    public void init()
11    {
12       JTextArea outputArea = new JTextArea();
13       Container container = getContentPane();
14       container.add( outputArea );
15
16       int array[] = { 6, 4, 8, 2, 10, 12, 89, 68, 45, 37 };
17
18       String output = "Data items in original order\n";
19
20       // append original array values to String output
21       for ( int counter = 0; counter < array.length; counter++ )
22          output += "   " + array[ counter ];
23
24       bubbleSort( array ); // sort array
25
26       output += "\n\nData items in ascending order\n";
27
28       // append sorted array values to String output
29       for ( int counter = 0; counter < array.length; counter++ )
30          output += "   " + array[ counter ];
31
32       outputArea.setText( output );
33
34    }  // end method init
35
36    // sort elements of array with bubble sort
37    public void bubbleSort( int array2[] )
38    {
39       // loop to control number of passes
40       for ( int pass = 1; pass < array2.length; pass++ ) {
41
42          // loop to control number of comparisons
43          for ( int element = 0; element < array2.length - pass;
44          element++ ) {
45
46             // compare side-by-side elements and swap them if
47             // first element is greater than second element
48             if ( array2[ element ] > array2[ element + 1 ] )
49                swap( array2, element, element + 1 );
50
51          }  // end loop to control comparisons
52
53       }  // end loop to control passes
54
55    }  // end method bubbleSort
56
```

```
57       // swap two elements of an array
58       public void swap( int array3[], int first, int second )
59       {
60          int hold;  // temporary holding area for swap
61
62          hold = array3[ first ];
63          array3[ first ] = array3[ second ];
64          array3[ second ] = hold;
65       }
66
67    }  // end class BubbleSortA
```



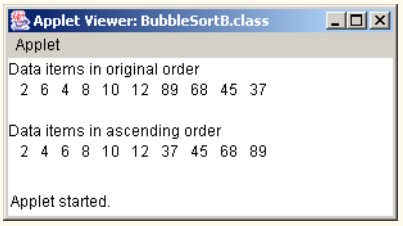
b) The data in the array may already be in the proper order or near-proper order, so why make nine passes if fewer will suffice? Modify the sort to check at the end of each pass if any swaps have been made. If none have been made, the data must already be in the proper order, so the program should terminate. If swaps have been made, at least one more pass is needed.

ANS:

```
1  // Exercise 7.11 Part B Solution: BubbleSortB.java
2  // Applet sorts an array's values using an efficient bubble sort, so the
3  // number of comparisons will decrease by one after each sort. The bubble
4  // sort will also end if there is nothing to sort in a pass.
5  import java.awt.*;
6  import javax.swing.*;
7
8  public class BubbleSortB extends JApplet {
9
10    // initialize applet
11    public void init()
12    {
13       JTextArea outputArea = new JTextArea();
14       Container container = getContentPane();
15       container.add( outputArea );
16
17       int array[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
18
19       String output = "Data items in original order\n";
20
21       // append original array values to String output
22       for ( int counter = 0; counter < array.length; counter++ )
23          output += "   " + array[ counter ];
```

```
24
25          bubbleSort( array );  // sort array
26
27          output += "\n\nData items in ascending order\n";
28
29          // append sorted array values to String output
30          for ( int counter = 0; counter < array.length; counter++ )
31             output += "   " + array[ counter ];
32
33          outputArea.setText( output );
34
35       }  // end method init
36
37       // sort elements of array with bubble sort
38       public void bubbleSort( int array2[] )
39       {
40          // boolean indicating if a swap took place during pass
41          boolean didSwap;
42
43          // loop to control number of passes
44          for ( int pass = 1; pass < array2.length; pass++ ) {
45
46             didSwap = false;
47
48             // loop to control number of comparisons
49             for ( int element = 0;
50                   element < array2.length - pass; element++ ) {
51
52                // compare side-by-side elements and swap them if
53                // first element is greater than second element
54                if ( array2[ element ] > array2[ element + 1 ] ) {
55
56                   swap( array2, element, element + 1 );
57                   didSwap = true;
58                }
59
60             }  // end loop to control comparisons
61
62             // if no swaps, terminate bubble sort
63             if ( !didSwap )
64                return;
65
66          }  // end loop to control passes
67
68       }  // end method bubbleSort
69
70       // swap two elements of an array
71       public void swap( int array3[], int first, int second )
72       {
73          // temporary holding area for swap
74          int hold;
75
76          hold = array3[ first ];
77          array3[ first ] = array3[ second ];
```

```
78          array3[ second ] = hold;
79       }
80
81    }  // end class BubbleSortB
```



7.12 Write statements that perform the following one-dimensional-array operations:

a) Set the 10 elements of integer array `counts` to zero.

ANS:
```
for ( int u = 0; u < counts.length; u++ )
   counts[ u ] = 0;
```

b) Add one to each of the 15 elements of integer array `bonus`.

ANS:
```
for ( int v = 0; v < bonus.length; v++ )
   bonus[ v ]++;
```

c) Print the five values of integer array `bestScores` in column format.

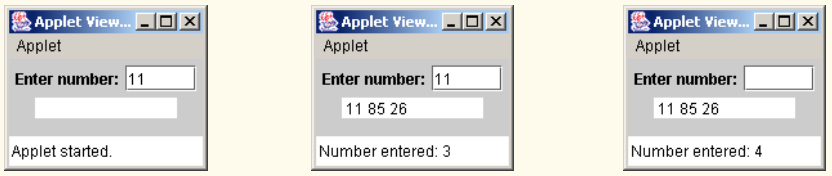
ANS:
```
for ( int w = 0; w < bestScores.length; w++ )
   System.out.println( bestScores[ w ] );
```

7.13 Use a one-dimensional array to solve the following problem: Write an applet that inputs 5 numbers, each of which is between 10 and 100, inclusive. As each number is read, display it only if it is not a duplicate of a number already read. Provide for the "worst case," in which all 5 numbers are different. Use the smallest possible array to solve this problem. The applet should use the GUI techniques introduced in Chapter 6. Display the results in a `JTextArea`. Use `JTextArea` method `setText` to update the results after each value is input by the user.

ANS:

```
1  // Exercise 7.13 Solution: Unique.java
2  // Applet reads in 20 numbers and prints only the non-duplicates.
3  import java.awt.event.*;
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class Unique extends JApplet implements ActionListener {
8
9     JTextField input;  // input text field
10    JTextArea output;  // output text area
11    JLabel prompt;
12
13    int array[], counter = 0, numberCount = 0;
14
15    // initialize applet
16    public void init()
17    {
18       array = new int[ 5 ];
```

```
19
20          input = new JTextField( 5 );
21          input.addActionListener( this );
22          prompt = new JLabel( "Enter number:" );
23
24          output = new JTextArea( 1, 10 );
25          output.setEditable( false );
26
27          Container container = getContentPane();
28          container.setLayout( new FlowLayout() );
29          container.add( prompt );
30          container.add( input );
31          container.add( output );
32
33       }  // end method init
34
35       // read number input and print if non-duplicate
36       public void actionPerformed( ActionEvent actionEvent )
37       {
38          int number = Integer.parseInt( input.getText() );
39
40          input.setText("");
41
42          // validate the input
43          if ( number < 10 || number > 100 ) {
44
45             showStatus( "Invalid Entry!!!" );
46             return;
47          }
48
49          // count to see if all the numbers have been entered
50          if ( numberCount < 5 ) {
51             showStatus( "Number entered: " + ( ++numberCount ) );
52
53             // compare input number to unique numbers in array
54             for ( int i = 0; i < counter; i++ )
55
56                // if new number is duplicate, do nothing
57                if ( number == array[ i ] )
58                   return;
59
60             // append new number to text area
61             output.append( " " + number );
62
63             // store new and unique number
64             array[ counter++ ] = number;
65          }
66          else
67             showStatus( "Number entered exceed 5." );
68
69       }  // end method actionPerformed
70
71    }  // end class Unique
```



7.14 Label the elements of three-by-five two-dimensional array `sales` to indicate the order in which they are set to zero by the following program segment:

```
for ( int row = 0; row < sales.length; row++ )

   for ( int col = 0; col < sales[ row ].length; col++ )

      sales[ row ][ col ] = 0;
```

ANS:

```
sales[ 0 ][ 0 ], sales[ 0 ][ 1 ], sales[ 0 ][ 2 ], sales[ 0 ][ 3 ],
sales[ 0 ][ 4 ], sales[ 1 ][ 0 ], sales[ 1 ][ 1 ], sales[ 1 ][ 2 ],
sales[ 1 ][ 3 ], sales[ 1 ][ 4 ], sales[ 2 ][ 0 ], sales[ 2 ][ 1 ],
sales[ 2 ][ 2 ], sales[ 2 ][ 3 ], sales[ 2 ][ 4 ]
```

7.15 Write an applet to simulate the rolling of two dice. The program should use `Math.random` once to roll the first die and again to roll the second die. The sum of the two values should then be calculated. Each die can show an integer value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum and 2 and 12 being the least frequent sums. Figure 7.20 shows the 36 possible combinations of the two dice. Your program should roll the dice 36,000 times. Use a one-dimensional array to tally the numbers of times each possible sum appears. Display the results in a `JTextArea` in tabular format. Also, determine whether the totals are reasonable (i.e., there are six ways to roll a 7, so approximately one-sixth of the rolls should be 7). The applet should use the GUI techniques introduced in Chapter 6. Provide a `JButton` to allow the user of the applet to roll the dice another 36,000 times. The applet should reset the elements of the one-dimensional array to zero before rolling the dice again.

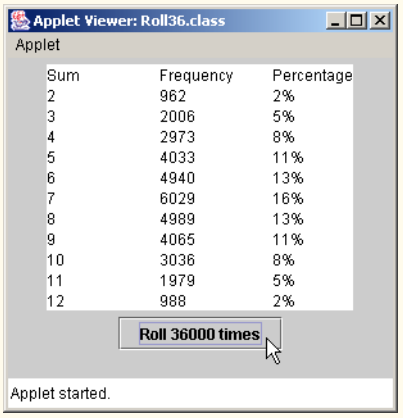
	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 7.20 The 36 possible sums of two dice.

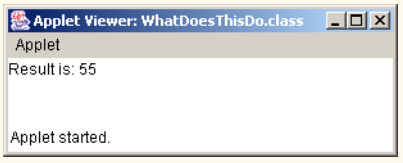
ANS:

```
1  // Exercise 7.15 Solution: Roll36.java
2  // Program simulates rolling two six-sided dice 36,000 times.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Roll36 extends JApplet implements ActionListener {
8     int total[];
9     int totalRolls;
10    JTextArea outputArea;
11    JButton button;
12
13    // set up GUI components, initialize array and roll dice
14    public void init()
15    {
16       button = new JButton("Roll 36000 times");
17       outputArea = new JTextArea();
18       button.addActionListener( this );
19
20       Container container = getContentPane();
21       container.setLayout( new FlowLayout() );
22       container.add( outputArea );
23       container.add( button );
24
25       totalRolls = 0;
26       total = new int[ 13 ];
27
28       for ( int index = 0; index < total.length; index++ )
29          total[ index ] = 0;
30
31       rollDice();
32
33    }  // end method init
34
35    // simulate rolling of dice 36000 times
36    public void rollDice()
37    {
38       int face1, face2;
39
40       for ( int roll = 1; roll <= 36000; roll++ ) {
41          face1 = ( int ) ( 1 + Math.random() * 6 );
42          face2 = ( int ) ( 1 + Math.random() * 6 );
43          total[ face1 + face2 ]++;
44       }
45
46       totalRolls += 36000;
47
48       // print table on text area
49       String output = "Sum\tFrequency\tPercentage";
50
51       // ignore subscripts 0 and 1
52       for ( int k = 2; k < total.length; k++ ) {
```

```
53
54             int percent = total[ k ] / ( totalRolls / 100 );
55             output += "\n" + k + "\t" + total[ k ] + "\t" + percent + "%";
56          }
57
58          outputArea.setText( output );
59
60       }  // end method roll2Dice
61
62       // roll dice again
63       public void actionPerformed( ActionEvent actionEvent )
64       {
65          for ( int i=0; i < total.length; i++ )
66             total[ i ] = 0;
67
68          totalRolls = 0;
69             rollDice();
70
71       }  // end method actionPerformed
72
73    }  // end class Roll36
```



7.16 What does the program of Fig. 7.21 do?
ANS:



7.17 Write a program that runs 1000 games of craps (Fig. 6.9) and answers the following questions:

a) How many games are won on the first roll, second roll, …, twentieth roll and after the twentieth roll?

b) How many games are lost on the first roll, second roll, …, twentieth roll and after the twentieth roll?
c) What are the chances of winning at craps? [*Note*: You should discover that craps is one of the fairest casino games. What do you suppose this means?]
d) What is the average length of a game of craps?
e) Do the chances of winning improve with the length of the game?

ANS:

```
// Exercise 7.17 Solution: Craps.java
// Program plays 1000 games of craps and displays winning
// and losing statistics.
import java.awt.*;

public class Craps {

   // game status constants
   static final int CONTINUE = 0, LOSE = 1, WIN = 2;

   // number of rolls, number of games
   static final int SIZE = 22, GAMES = 1000;

   public static void main( String args[] )
   {
      int gameStatus, sum, roll;
      int myPoint = 0;  // point total goal
      int length = 0;   // length of game (in rolls)

      // number of wins and losses on roll index
      int wins[], losses[];

      int winSum = 0, loseSum = 0; // total number of wins and losses

      wins = new int[ SIZE ];
      losses = new int[ SIZE ];

      // play GAMES number of crap games and determine results,
      // number of rolls, and number of wins/losses on that roll
      for ( int i = 1; i <= GAMES; i++ ) {

         sum = rollDice();

         roll = 1; // initialize to first roll of game

         // determine result of first roll
         switch ( sum ) {

            // win
            case 7: case 11:
               gameStatus = WIN;
               break;

            // lose
            case 2: case 3: case 12:
```

```
46                gameStatus = LOSE;
47                break;
48
49             // game not ended
50             default:
51                gameStatus = CONTINUE;
52
53                myPoint = sum; // assign point goal as sum of dices
54                break;
55
56          }  // end switch
57
58          // continue to roll until a win or loss
59          while ( gameStatus == CONTINUE ) {
60
61             sum = rollDice();
62
63             ++roll; // increment number of rolls of this game
64
65             // dice roll equals point goal
66             if ( sum == myPoint )
67                gameStatus = WIN;
68
69             else // rolled a 7 before getting point
70                if ( sum == 7 )
71                   gameStatus = LOSE;
72
73          }  // end while
74
75          // all roll results after 20th roll placed in last array element
76          if ( roll > 21 )
77             roll = 21;
78
79          // increment number of wins in that roll
80          if ( gameStatus == WIN ) {
81
82             ++wins[ roll ];
83             ++winSum;
84          }
85
86          // increment number of losses in that roll
87          else {
88
89             ++losses[ roll ];
90             ++loseSum;
91          }
92
93       }  // end for
94
95       // display number of wins and losses on all rolls
96       for ( int i = 1; i <= 21; i++ ) {
97
98          if ( i == 21 )
99             System.out.println( wins[ i ] + " games won and " +
```

```
100                losses[ i ] + " games lost on rolls after the 20th roll" );
101
102             else
103                System.out.println( wins[ i ] + " games won and " +
104                   losses[ i ] + " games lost on roll #" + i );
105          }
106
107          // calculate chances of winning
108          System.out.println( "\nThe chances of winning are " +
109             winSum + " / " + ( winSum + loseSum ) + " = " +
110             ( 100.0 * winSum / ( winSum + loseSum ) ) + "%" );
111
112          // calculate length of game
113          for ( int i = 1; i <= 21; i++ )
114
115             // number of wins/losses on that roll multiplied
116             // by the roll number, then add them to length
117             length += wins[ i ] * i + losses[ i ] * i;
118
119          System.out.println( "The average game length is " +
120             ( length / 1000.0 ) + " rolls." );
121
122       }  // end method main
123
124       // rolls dice, returns sum of dice rolls
125       public static int rollDice()
126       {
127          int die1 = ( int ) ( 1 + Math.random() * 6 );
128          int die2 = ( int ) ( 1 + Math.random() * 6 );
129          int workSum = die1 + die2;
130
131          return workSum;
132       }
133
134    }  // end class Craps
```

```
219 games won and 105 games lost on roll #1
86 games won and 109 games lost on roll #2
59 games won and 82 games lost on roll #3
41 games won and 56 games lost on roll #4
34 games won and 41 games lost on roll #5
16 games won and 29 games lost on roll #6
9 games won and 26 games lost on roll #7
11 games won and 14 games lost on roll #8
12 games won and 11 games lost on roll #9
7 games won and 3 games lost on roll #10
4 games won and 4 games lost on roll #11
2 games won and 4 games lost on roll #12
2 games won and 4 games lost on roll #13
0 games won and 0 games lost on roll #14
1 games won and 2 games lost on roll #15
1 games won and 3 games lost on roll #16
0 games won and 0 games lost on roll #17
0 games won and 0 games lost on roll #18
0 games won and 0 games lost on roll #19
1 games won and 1 games lost on roll #20
0 games won and 1 games lost on rolls after the 20th roll

The chances of winning are 505 / 1000 = 50.5%
The average game length is 3.33 rolls.
```

7.18 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. You have been asked to program the new system. You are to write an applet to assign seats on each flight of the airline's only plane (capacity: 10 seats).

Your program should display the following alternatives: `Please type 1 for First Class` and `Please type 2 for Economy`. If the user types `1`, your program should assign a seat in the first-class section (seats 1–5). If the person types `2`, your program should assign a seat in the economy section (seats 6–10). Your program should then print a boarding pass indicating the person's seat number and whether it is in the first-class or economy section of the plane.

Use a one-dimensional array of primitive type `boolean` to represent the seating chart of the plane. Initialize all the elements of the array to `false` to indicate that all seats are empty. As each seat is assigned, set the corresponding elements of the array to `true` to indicate that the seat is no longer available.

```
1  // Exercise 7.16: WhatDoesThisDo.java
2  import java.awt.*;
3  import javax.swing.*;
4
5  public class WhatDoesThisDo extends JApplet {
6     int result;
7
8     public void init()
9     {
10       int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
11
12       result = whatIsThis( array, array.length );
13
```

Fig. 7.21 What does this program do?

```
14          Container container = getContentPane();
15          JTextArea output = new JTextArea();
16          output.setText( "Result is: " + result );
17          container.add( output );
18       }
19
20       public int whatIsThis( int array2[], int length )
21       {
22          if ( length == 1 )
23             return array2[ 0 ];
24          else
25             return array2[ length - 1 ] + whatIsThis( array2, length - 1 );
26       }
27
28    } // end class WhatDoesThisDo
```

Fig. 7.21 What does this program do?

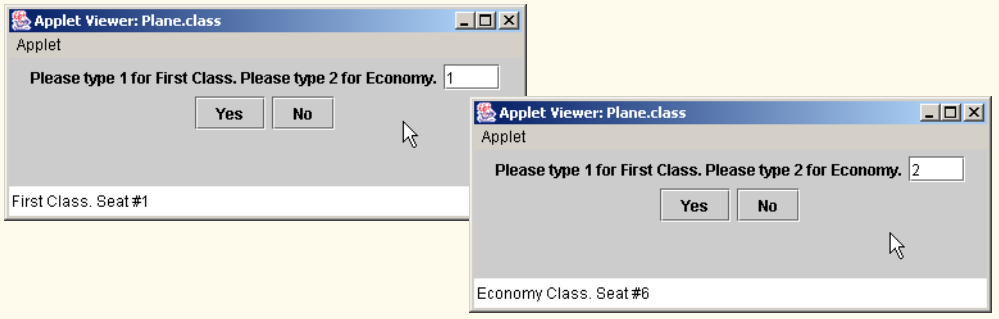
Your program should never assign a seat that has already been assigned. When the economy section is full, your program should ask the person if it is acceptable to be placed in the first-class section (and vice versa). If yes, make the appropriate seat assignment. If no, print the message "Next flight leaves in 3 hours."

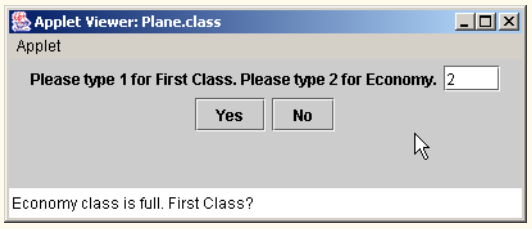
ANS:

```
1  // Exercise 7.18 Solution: Plane.java
2  // Program reserves airline seats.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Plane extends JApplet implements ActionListener {
8     JTextField input;
9     JLabel prompt;
10    JButton yesButton, noButton;
11    int section, firstClass, economyClass;
12    boolean seats[];
13    boolean questionPosed = false;
14
15    // set up GUI components and initialize instance variables
16    public void init()
17    {
18       prompt = new JLabel( "Please type 1 for First Class. " +
19          "Please type 2 for Economy." );
20       input = new JTextField( 4 );
21       yesButton = new JButton( "Yes" );
22       noButton = new JButton( "No" );
23
24       input.addActionListener( this );
25       yesButton.addActionListener( this );
26       noButton.addActionListener( this );
27
28       Container container = getContentPane();
29       container.setLayout( new FlowLayout() );
30       container.add( prompt );
```

```
31          container.add( input );
32          container.add( yesButton );
33          container.add( noButton );
34
35          // initialize values
36          firstClass = 0;
37          economyClass = 5;
38          seats = new boolean[ 10 ];
39          for ( int index = 0; index < seats.length; index++ )
40             seats[ index ] = false;
41
42       }  // end method init
43
44       // perform appropriate action
45       public void actionPerformed( ActionEvent actionEvent )
46       {
47          // input field
48          if ( actionEvent.getSource() == input ) {
49             section = Integer.parseInt( input.getText() );
50             String output = "";
51             questionPosed = false;
52
53             // first class
54             if ( section == 1 ) {
55
56                // if firstClass isn't full, reserve a seat
57                if ( firstClass < 5 ) {
58                   seats[ firstClass ] = true;
59                   output = "First Class. Seat #" + ++firstClass;
60                }
61
62                // if it is full, offer a seat in economy class
63                else if ( firstClass >= 5 && economyClass < 10 ) {
64                   output = "First Class is full. Economy Class?";
65                   questionPosed = true;
66                }
67
68                else
69                   output = "Flight is full. Try next flight.";
70             }
71
72             // economy class
73             else if ( section == 2 ) {
74
75                // if non-smoking isn't full, reserve a seat
76                if ( economyClass < 10 ) {
77                   seats[ economyClass ] = true;
78                   output = "Economy Class. Seat #" + ++economyClass;
79                }
80
81                // if it is full, offer a seat in smoking
82                else if ( economyClass == 10 && firstClass < 5 ) {
83                   output = "Economy class is full. First Class?";
84                   questionPosed = true;
85                }
```

```
86
87                else
88                   output = "Flight is full. Try next flight.";
89             }
90
91             else
92                output = "Invalid input.";
93
94             showStatus( output );
95          }
96
97          // yes button
98          else if ( actionEvent.getSource() == yesButton ) {
99
100            if ( questionPosed ) {
101
102               if ( section == 1 ) {
103                  seats[ economyClass ] = true;
104                  showStatus( "Economy Class. Seat #" + ++economyClass );
105               }
106
107               else {  // section is 2
108                  seats[ firstClass ] = true;
109                  showStatus( "First Class. Seat #" + ++firstClass );
110               }
111
112               questionPosed = false;
113            }
114         }
115
116         // no button
117         else if ( actionEvent.getSource() == noButton ) {
118
119            if ( questionPosed )
120               showStatus( "Next flight leaves in 3 hours." );
121
122            questionPosed = false;
123         }
124
125      }  // end method actionPerformed
126
127   }  // end class Plane
```



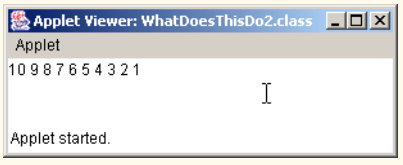


7.19 What does the program of Fig. 7.22 do?

```
1  // Exercise 7.19: WhatDoesThisDo2.java
2  import java.awt.*;
3  import javax.swing.*;
4
5  public class WhatDoesThisDo2 extends JApplet {
6
7     public void init()
8     {
9        int array[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
10       JTextArea outputArea = new JTextArea();
11
12       someFunction( array, 0, outputArea );
13
14       Container container = getContentPane();
15       container.add( outputArea );
16    }
17
18    public void someFunction( int array2[], int x, JTextArea out )
19    {
20       if ( x < array2.length ) {
21          someFunction( array2, x + 1, out );
22          out.append( array2[ x ] + "   " );
23       }
24    }
25 }
```

Fig. 7.22 What does this program do?

ANS:



7.20 Use a two-dimensional array to solve the following problem: A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each type of product sold. Each slip contains the following:

a) The salesperson number
b) The product number

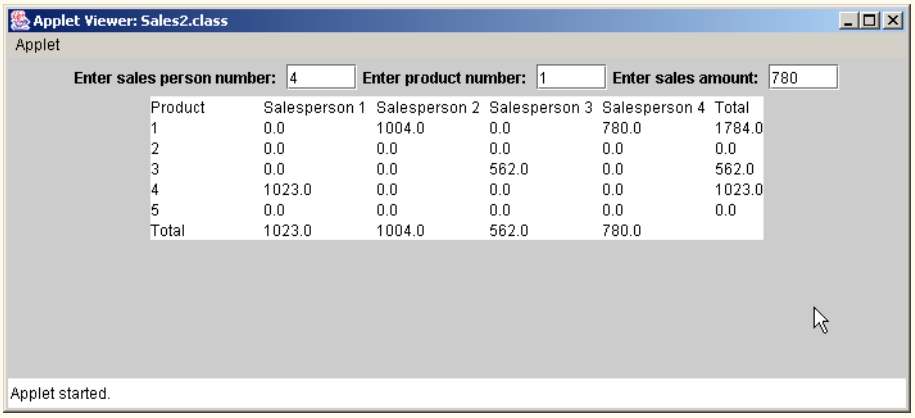
c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all of the slips for last month is available. Write an applet that will read all this information for last month's sales and summarize the total sales by salesperson by product. All totals should be stored in the two-dimensional array `sales`. After processing all the information for last month, display the results in tabular format, with each column representing a particular salesperson and each row representing a particular product. Cross-total each row to get the total sales of each product for last month; cross-total each column to get the total sales by salesperson for last month. Your tabular printout should include these cross-totals to the right of the totaled rows and to the bottom of the totaled columns. Display the results in a `JTextArea`.

ANS:

```
1  // Exercise 7.20 Solution: Sales2.java
2  // Program totals sales for salespeople and products.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Sales2 extends JApplet implements ActionListener {
8     JLabel prompt1, prompt2, prompt3;
9     JTextField input1, input2, input3;
10    JTextArea outputArea;
11    double sales[][];
12
13    // set up GUI components
14    public void init()
15    {
16       prompt1 = new JLabel( "Enter sales person number: " );
17       prompt2 = new JLabel( "Enter product number: " );
18       prompt3 = new JLabel( "Enter sales amount: " );
19
20       // sales array holds data on number of each product sold
21       // by each salesman
22       sales = new double[ 5 ][ 4 ];
23
24       input1 = new JTextField( 5 );
25       input2 = new JTextField( 5 );
26       input3 = new JTextField( 5 );
27       input3.addActionListener( this );
28       outputArea = new JTextArea();
29       outputArea.setEditable( false );
30
31       // add all components to GUI
32       Container container = getContentPane();
33       container.setLayout( new FlowLayout() );
34       container.add( prompt1 );
35       container.add( input1 );
36       container.add( prompt2 );
37       container.add( input2 );
38       container.add( prompt3 );
39       container.add( input3 );
40       container.add( outputArea );
41
```

```
42     }  // end method init
43
44     // obtain user input
45     public void actionPerformed( ActionEvent actionEvent )
46     {
47        // read in sales of a product by a person
48        int person = Integer.parseInt( input1.getText() );
49        int product = Integer.parseInt( input2.getText() );
50        double d = Double.parseDouble( input3.getText() );
51
52        // error-check the input
53        if ( person >= 1 && person < 5 &&
54           product >= 1 && product < 6 && d >= 0 )
55
56           sales[ product - 1 ][ person - 1 ] += d;
57
58        else
59           showStatus( "Invalid input!" );
60
61        // display the updated table
62        double salesPersonTotal[] = new double[ 4 ];
63
64        for ( int column = 0; column < 4; column++ )
65           salesPersonTotal[ column ] = 0;
66
67        String output = "Product\tSalesperson 1\tSalesperson 2" +
68              "\tSalesperson 3\tSalesperson 4\tTotal";
69
70        // for each column of each row, print the appropriate
71        // value representing a person's sales of a product
72        for ( int row = 0; row < 5; row++ ) {
73
74           double productTotal = 0.0;
75           output += "\n" + ( row + 1 );
76
77           for ( int column = 0; column < 4; column++ ) {
78              output += "\t" + sales[ row ][ column ];
79              productTotal += sales[ row ][ column ];
80              salesPersonTotal[ column ] += sales[ row ][ column ];
81           }
82
83           output += "\t" + productTotal;
84        }
85
86        output += "\nTotal";
87
88        for ( int column = 0; column < 4; column++ )
89           output += "\t" + salesPersonTotal[ column ];
90
91        outputArea.setText( output );
92
93     }  // end method actionPerformed
94
95  }  // end class Sales2
```



7.21 (*Turtle Graphics*) The Logo language made the concept of *turtle graphics* famous. Imagine a mechanical turtle that walks around the room under the control of a Java program. The turtle holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves; while the pen is up, the turtle moves about freely without writing anything. In this problem, you will simulate the operation of the turtle and create a computerized sketchpad.

Use a 20-by-20 array `floor` that is initialized to zeros. Read commands from an array that contains them. Keep track of the current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at position (0, 0) of the floor with its pen up. The set of turtle commands your program must process are shown in Fig. 7.23.

Command	Meaning
1	Pen up
2	Pen down
3	Turn right
4	Turn left
5,10	Move forward 10 spaces (replace 10 for a different number of spaces)
6	Print the 20-by-20 array
9	End of data (sentinel)

Fig. 7.23 Turtle graphics commands.

Suppose that the turtle is somewhere near the center of the floor. The following "program" would draw and print a 12-by-12 square, leaving the pen in the up position:

```
2
5,12
3
5,12
3
5,12
3
```

```
5,12
1
6
9
```

As the turtle moves with the pen down, set the appropriate elements of array `floor` to 1s. When the 6 command (print) is given, wherever there is a 1 in the array, display an asterisk or any character you choose. Wherever there is a 0, display a blank.

Write a Java applet to implement the turtle graphics capabilities discussed here. The applet should display the turtle graphics in a `JTextArea`, using Monospaced font. Write several turtle graphics programs to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

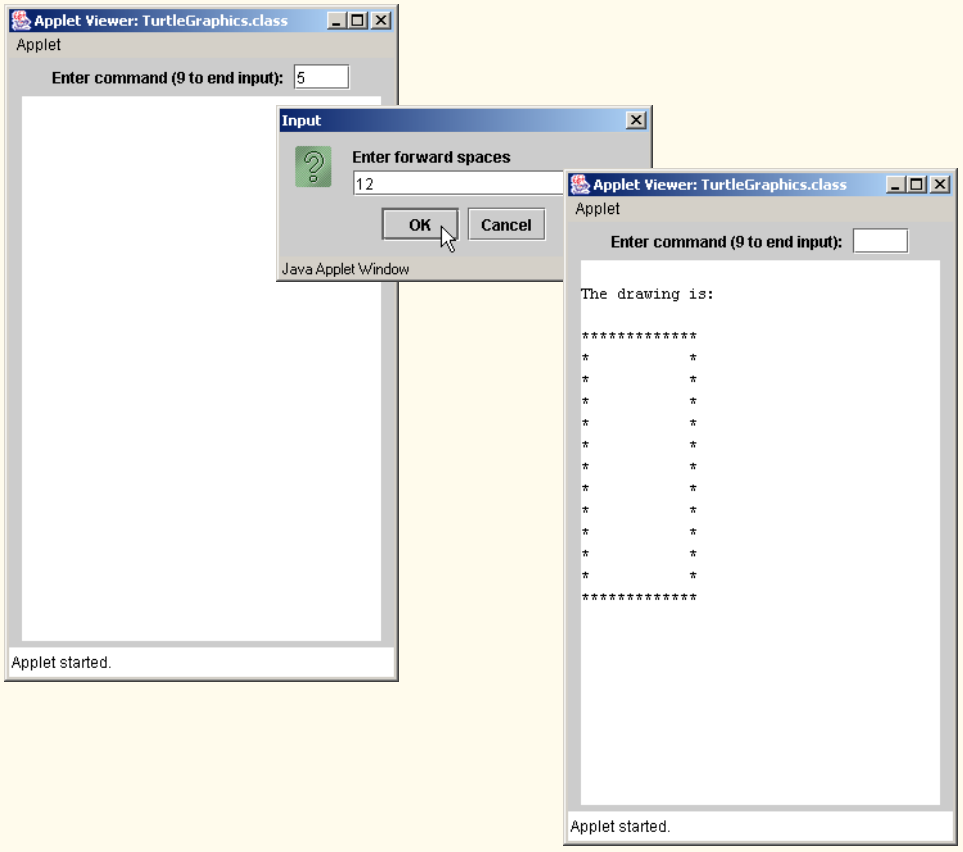
ANS:

```
1  // Exercise 7.21: TurtleGraphics.java
2  // Drawing turtle graphics based on turtle commands.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class TurtleGraphics extends JApplet implements ActionListener {
8     final int MAXCOMMANDS = 100;
9     final int SIZE = 20;
10    JLabel prompt;
11    JTextField input;
12    JTextArea output;
13    int floor[][];
14    int commandArray[][];
15    int command, distance, direction, count, xPos, yPos;
16    boolean penDown;
17
18    // set up GUI components and initialize instance variables
19    public void init()
20    {
21       prompt = new JLabel( "Enter command (9 to end input): " );
22       input = new JTextField( 4 );
23       input.addActionListener( this );
24
25       // set up JTextArea for displaying turtle graphics
26       output = new JTextArea( 25, 40 );
27       output.setFont( new Font( "Monospaced", Font.PLAIN, 12 ) );
28
29       Container container = getContentPane();
30       container.setLayout( new FlowLayout() );
31       container.add( prompt );
32       container.add( input );
33       container.add( output );
34
35       // initialize values
36       direction = 0;
37       count = 0;
38       xPos = 0;
39       yPos = 0;
40       penDown = false;
```

```
41
42          floor = new int[ SIZE ][ SIZE ];
43          commandArray = new int[ MAXCOMMANDS ][ 2 ];
44
45       }  // end method init
46
47       // perform appropriate action
48       public void actionPerformed( ActionEvent actionEvent )
49       {
50          // input field
51          if ( actionEvent.getSource() == input ) {
52             int inputCommand = Integer.parseInt( input.getText() );
53             input.setText( "" );
54
55             // if reach max commands
56             if ( count < MAXCOMMANDS ) {
57                commandArray[ count ][ 0 ] = inputCommand;
58
59                // prompt for forward spaces
60                if ( inputCommand == 5 ) {
61                   int spaces = Integer.parseInt( JOptionPane.showInputDialog(
62                      "Enter forward spaces" ) );
63                   commandArray[ count ][ 1 ] = spaces;
64                }
65             }
66
67             count++;
68
69             // execute commands if command is 9
70             if ( inputCommand == 9 || count == MAXCOMMANDS )
71                executeCommands();
72
73          } // end outer if
74
75       } // end method actionPerformed
76
77       public void executeCommands()
78       {
79          int commandNumber = 0;
80          command = commandArray[ commandNumber ][ 0 ];
81
82          // continue executing commands until either reach the end
83          // or reach the max commands
84          while ( command != 9 && commandNumber < MAXCOMMANDS ) {
85
86             // determine what command was entered
87             // and perform desired action
88             switch ( command ) {
89                case 1:
90                   penDown = false;
91                   break;
92                case 2:
93                   penDown = true;
94                   break;
```

```
95              case 3:
96                 direction = turnRight( direction );
97                 break;
98              case 4:
99                 direction = turnLeft( direction );
100                break;
101             case 5:
102                distance = commandArray[ commandNumber ][ 1 ];
103                movePen( penDown, floor, direction, distance );
104                break;
105             case 6:
106                output.append( "\nThe drawing is:\n\n" );
107                printArray( floor );
108                break;
109
110          }  // end switch
111
112          command = commandArray[ ++commandNumber ][ 0 ];
113
114       }  // end while
115
116    } // end method executeCommands
117
118    // method to turn turtle to the right
119    public int turnRight( int d )
120    {
121       return ++d > 3 ? 0 : d;
122    }
123
124    // method to turn turtle to the left
125    public int turnLeft( int d )
126    {
127       return --d < 0 ? 3 : d;
128    }
129
130    // method to move the pen
131    public void movePen(
132       boolean down, int a[][], int dir, int dist )
133    {
134       int j;  // looping variable
135
136       // determine which way to move pen
137       switch ( dir ) {
138
139          case 0:   // move to right
140             for ( j = 1; j <= dist && yPos + j < SIZE; ++j )
141                if ( down )
142                   a[ xPos ][ yPos + j ] = 1;
143
144             yPos += j - 1;
145             break;
146
147          case 1:   // move down
148             for ( j = 1; j <= dist && xPos + j < SIZE; ++j )
```

```
149                if ( down )
150                   a[ xPos + j ][ yPos ] = 1;
151
152             xPos += j - 1;
153             break;
154
155          case 2:   // move to left
156             for ( j = 1; j <= dist && yPos - j >= 0; ++j )
157                if ( down )
158                   a[ xPos ][ yPos - j ] = 1;
159
160             yPos -= j - 1;
161             break;
162
163          case 3:   // move up
164             for ( j = 1; j <= dist && xPos - j >= 0; ++j )
165                if ( down )
166                   a[ xPos - j ][ yPos ] = 1;
167
168             xPos -= j - 1;
169             break;
170
171       } // end switch
172
173    } // end method movePen
174
175    // method to print array drawing
176    public void printArray( int a[][] )
177    {
178       // display array
179       for ( int i = 0; i < SIZE; ++i ) {
180          for ( int j = 0; j < SIZE; ++j )
181             output.append( ( a[ i ][ j ] == 1 ? "*" : " " ) );
182
183          output.append( "\n" );
184       }
185    }
186
187 } // end class TurtleGraphics
```



7.22 (*Knight's Tour*) One of the more interesting puzzlers for chess buffs is the Knight's Tour problem, originally proposed by the mathematician Euler. Can the chess piece called the knight move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes only L-shaped moves (two spaces in one direction and one space in a perpendicular direction). Thus, as shown in Fig. 7.24, from a square near the middle of an empty chessboard, the knight (labeled K) can make eight different moves (numbered 0 through 7).

a) Draw an eight-by-eight chessboard on a sheet of paper, and attempt a Knight's Tour by hand. Put a 1 in the starting square, a 2 in the second square, a 3 in the third, etc. Before starting the tour, estimate how far you think you will get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let us develop an applet that will move the knight around a chessboard. The board is represented by an eight-by-eight two-dimensional array `board`. Each square is initialized to zero. We describe each of the eight possible moves in terms of both their horizontal and vertical components. For example, a move of type 0 as shown in Fig. 7.24 consists of moving two squares horizontally to the right and one square vertically upward. A move of type 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two one-dimensional arrays `horizontal` and `vertical` as follows:

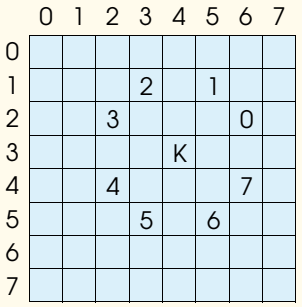


Fig. 7.24 The eight possible moves of the knight.

```
horizontal[ 0 ] = 2        vertical[ 0 ] = -1
horizontal[ 1 ] = 1        vertical[ 1 ] = -2
horizontal[ 2 ] = -1       vertical[ 2 ] = -2
horizontal[ 3 ] = -2       vertical[ 3 ] = -1
horizontal[ 4 ] = -2       vertical[ 4 ] = 1
horizontal[ 5 ] = -1       vertical[ 5 ] = 2
horizontal[ 6 ] = 1        vertical[ 6 ] = 2
horizontal[ 7 ] = 2        vertical[ 7 ] = 1
```

Let the variables `currentRow` and `currentColumn` indicate the row and column; respectively, of the knight's current position. To make a move of type `moveNumber`, where `moveNumber` is between 0 and 7, your program should use the statements

```
currentRow += vertical[ moveNumber ];
currentColumn += horizontal[ moveNumber ];
```

Write a program to move the knight around the chessboard. Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Test each potential move to see if the knight already visited that square. Test every potential move to ensure that the knight does not land off the chessboard. Run the program. How many moves did the knight make?

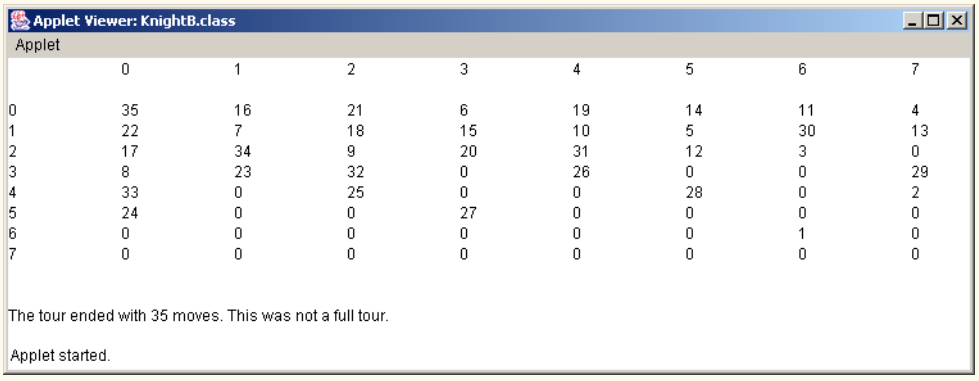
ANS:

```
1  // Exercise 7.22 Part b Solution: KnightB.java
2  // Knight's Tour
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class KnightB extends JApplet {
7
8     int currentRow, currentColumn, moveNumber;
9
10    int board[][];    // gameboard
11
12    int testRow, testColumn, moveType; // row and column positions
13
```

```
14       // moves
15       int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
16       int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
17
18       boolean done, goodMove;
19       JTextArea tour;
20
21       // initialize applet
22       public void init()
23       {
24          board = new int[ 8 ][ 8 ];
25
26          // randomize initial board position
27          currentRow = ( int ) ( Math.random() * 8 );
28          currentColumn = ( int ) ( Math.random() * 8 );
29
30          board[ currentRow ][ currentColumn ] = ++moveNumber;
31          done = false;
32
33          tour = new JTextArea( 20, 30 );
34          Container container = getContentPane();
35          container.add( tour );
36       }
37
38       // start touring
39       public void start()
40       {
41          // continue until knight can no longer move
42          while ( !done ) {
43             moveType = 0;
44             testRow = currentRow + vertical[ moveType ];
45             testColumn = currentColumn + horizontal[ moveType ];
46             goodMove = validMove( testRow, testColumn );
47
48             // test if desired move is valid
49             if ( goodMove ) {
50                currentRow = testRow;
51                currentColumn = testColumn;
52                board[ currentRow ][ currentColumn ] = ++moveNumber;
53             }
54             else { // if move is not legal try another move
55
56                for ( int count = 0; count < 7 && !goodMove; ++count ) {
57                   moveType = ++moveType % 8;
58                   testRow = currentRow + vertical[ moveType ];
59                   testColumn = currentColumn + horizontal[ moveType ];
60                   goodMove = validMove( testRow, testColumn );
61
62                   // test if new move is valid
63                   if ( goodMove ) {
64                      currentRow = testRow;
65                      currentColumn = testColumn;
66                      board[ currentRow ][ currentColumn ] = ++moveNumber;
67                   }
68                }
```

```
69
70             // if no valid moves, knight can no longer move
71             if ( !goodMove )
72                done = true;
73          }
74
75          // if 64 moves have been made, a full tour is complete
76          if ( moveNumber == 64 )
77             done = true;
78
79       } // end while
80
81       String statusBar = "The tour ended with " + moveNumber + " moves.";
82
83       if ( moveNumber == 64 )
84          statusBar += " This was a full tour!";
85       else
86          statusBar += " This was not a full tour.";
87
88       printTour();
89
90       tour.append( "\n\n" );
91       tour.append( statusBar );
92
93    }  // end method start
94
95    // checks for valid move
96    public boolean validMove( int row, int column )
97    {
98       // NOTE: This test stops as soon as it becomes false
99       return ( row >= 0 && row < 8 && column >= 0 && column < 8
100         && board[ row ][ column ] == 0 );
101   }
102
103   // display Knight's tour path
104   public void printTour()
105   {
106      String buildString = "\t";
107
108      // display numbers for column
109      for ( int k = 0; k < 8; k++ )
110         buildString += k + "\t";
111
112      buildString += "\n\n";
113      tour.append( buildString );
114      buildString = "";
115
116      for ( int row = 0; row < board.length; row++ ) {
117
118         buildString += row + "\t";
119
120         for ( int column = 0; column < board[ row ].length; column++ )
121            buildString += board[ row ][ column ] + "\t";
122
```

```
123            buildString += "\n";
124         }
125
126         tour.append( buildString );
127
128      }  // end method printTour
129
130   }  // end class KnightB
```



c) After attempting to write and run a Knight's Tour program, you have probably developed some valuable insights. We will use these insights to develop a *heuristic* (or "rule of thumb") for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome or inaccessible squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to, so that when the board gets congested near the end of the tour, there will be a greater chance of success.

We could develop an "accessibility heuristic" by classifying each of the squares according to how accessible it is and always moving the knight (using the knight's L-shaped moves) to the most inaccessible square. We label a two-dimensional array `accessibility` with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each of the 16 squares nearest the center is rated as 8; each corner square is rated as 2; and the other squares have accessibility numbers of 3, 4 or 6 as follows:

```
2  3  4  4  4  4  3  2
3  4  6  6  6  6  4  3
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
4  6  8  8  8  8  6  4
3  4  6  6  6  6  4  3
2  3  4  4  4  4  3  2
```

Write a new version of the Knight's Tour, using the accessibility heuristic. The knight should always move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in

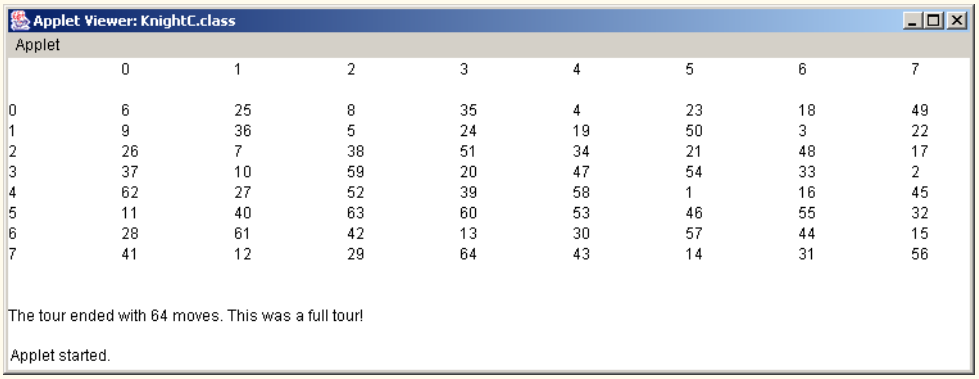
any of the four corners. [*Note*: As the knight moves around the chessboard, your program should reduce the accessibility numbers as more squares become occupied. In this way, at any given time during the tour, each available square's accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your program. Did you get a full tour? Modify the program to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

ANS:

```
1  // Exercise 7.22 Part C Solution: Knight.java
2  // Knight's Tour - heuristic version
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class KnightC extends JApplet {
7
8     int currentRow, currentColumn, moveNumber;
9
10    int board[][];    // gameboard
11
12    // accessibility values for each board position
13    int access[][] = { { 2, 3, 4, 4, 4, 4, 3, 2 },
14                       { 3, 4, 6, 6, 6, 6, 4, 3 },
15                       { 4, 6, 8, 8, 8, 8, 6, 4 },
16                       { 4, 6, 8, 8, 8, 8, 6, 4 },
17                       { 4, 6, 8, 8, 8, 8, 6, 4 },
18                       { 4, 6, 8, 8, 8, 8, 6, 4 },
19                       { 3, 4, 6, 6, 6, 6, 4, 3 },
20                       { 2, 3, 4, 4, 4, 4, 3, 2 } };
21
22    // row and column positions
23    int testRow, testColumn, count, minRow, minColumn;
24
25    int minAccess, accessNumber; // accessibility values
26
27    // moves
28    int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
29    int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
30
31    boolean done;
32    JTextArea tour;
33
34    // initialize applet
35    public void init()
36    {
37       minAccess = 9;
38       board = new int[ 8 ][ 8 ];
39
40       // randomize initial board position
41       currentRow = ( int ) ( Math.random() * 8 );
42       currentColumn = ( int ) ( Math.random() * 8 );
43
44       board[ currentRow ][ currentColumn ] = ++moveNumber;
45       done = false;
```

```
46
47          tour = new JTextArea( 20, 30 );
48          Container container = getContentPane();
49          container.add( tour );
50       }
51
52       // start touring
53       public void start()
54       {
55          // continue touring until finished traversing
56          while ( !done ) {
57
58             accessNumber = minAccess;
59
60             // try all possible moves
61             for ( int moveType = 0; moveType < board.length; moveType++ ) {
62
63                // new position of hypothetical moves
64                testRow = currentRow + vertical[ moveType ];
65                testColumn = currentColumn + horizontal[ moveType ];
66
67                if ( validMove( testRow, testColumn ) ) {
68
69                   // obtain access number
70                   if ( access[ testRow ][ testColumn ] < accessNumber ) {
71
72                      accessNumber = access[ testRow ][ testColumn ];
73
74                      minRow = testRow;
75                      minColumn = testColumn;
76                   }
77
78                   // position access number tried
79                   --access[ testRow ][ testColumn ];
80                }
81
82             }  // end if
83
84             // traversing done
85             if ( accessNumber == minAccess )
86                done = true;
87
88             else { // make move
89                currentRow = minRow;
90                currentColumn = minColumn;
91                board[ currentRow ][ currentColumn ] = ++moveNumber;
92             }
93
94          }  // end while
95
96          String statusBar = "The tour ended with " + moveNumber + " moves.";
97
98          if ( moveNumber == 64 )
99             statusBar += " This was a full tour!";
```

```
100         else
101            statusBar += " This was not a full tour.";
102
103         printTour();
104
105         tour.append( "\n\n" );
106         tour.append( statusBar );
107
108      }  // end method start
109
110      // checks for valid move
111      public boolean validMove( int row, int column )
112      {
113         // NOTE: This test stops as soon as it becomes false
114         return ( row >= 0 && row < 8 && column >= 0 && column < 8
115            && board[ row ][ column ] == 0 );
116      }
117
118      // display Knight's tour path
119      public void printTour()
120      {
121         String buildString = "\t";
122
123         // display numbers for column
124         for ( int k = 0; k < 8; k++ )
125            buildString += k + "\t";
126
127         buildString += "\n\n";
128         tour.append( buildString );
129         buildString = "";
130
131         for ( int row = 0; row < board.length; row++ ) {
132
133            buildString += row + "\t";
134
135            for ( int column = 0; column < board[ row ].length; column++ )
136               buildString += board[ row ][ column ] + "\t";
137
138            buildString += "\n";
139         }
140
141         tour.append( buildString );
142
143      }  // end method printTour
144
145   }  // end class KnightC
```



d) Write a version of the Knight's Tour program that, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the "tied" squares. Your program should move to the tied square for which the next move would arrive at a square with the lowest accessibility number.

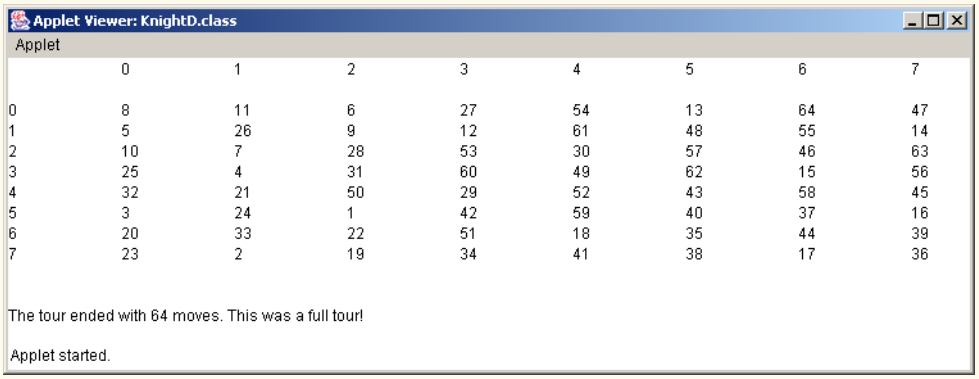
ANS:

```
1  // Exercise 7.22 Part D Solution: KnightD.java
2  // Knight's Tour - heuristic version
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class KnightD extends JApplet {
7
8     int currentRow, currentColumn, moveNumber;
9
10    int board[][];    // gameboard
11
12    // accessibility values for each board position
13    int access[][] = { { 2, 3, 4, 4, 4, 4, 3, 2 },
14                       { 3, 4, 6, 6, 6, 6, 4, 3 },
15                       { 4, 6, 8, 8, 8, 8, 6, 4 },
16                       { 4, 6, 8, 8, 8, 8, 6, 4 },
17                       { 4, 6, 8, 8, 8, 8, 6, 4 },
18                       { 4, 6, 8, 8, 8, 8, 6, 4 },
19                       { 3, 4, 6, 6, 6, 6, 4, 3 },
20                       { 2, 3, 4, 4, 4, 4, 3, 2 } };
21
22    // row and column positions
23    int testRow, testColumn, count, minRow, minColumn;
24
25    int minAccess, accessNumber; // accessibility values
26
27    // moves
28    int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
29    int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
30
31    boolean done;
```

```
32     JTextArea tour;
33
34     // initialize applet
35     public void init()
36     {
37        minAccess = 9;
38        board = new int[ 8 ][ 8 ];
39
40        // randomize initial board position
41        currentRow = ( int ) ( Math.random() * 8 );
42        currentColumn = ( int ) ( Math.random() * 8 );
43
44        board[ currentRow ][ currentColumn ] = ++moveNumber;
45        done = false;
46
47        tour = new JTextArea( 20, 30 );
48        Container container = getContentPane();
49        container.add( tour );
50     }
51
52     // start touring
53     public void start()
54     {
55        // continue touring until finished traversing
56        while ( !done ) {
57
58           accessNumber = minAccess;
59
60           // try all possible moves
61           for ( int moveType = 0; moveType < board.length; moveType++ ) {
62
63              // new position of hypothetical moves
64              testRow = currentRow + vertical[ moveType ];
65              testColumn = currentColumn + horizontal[ moveType ];
66
67              if ( validMove( testRow, testColumn ) ) {
68
69                 // obtain access number
70                 if ( access[ testRow ][ testColumn ] < accessNumber ) {
71
72                    accessNumber = access[ testRow ][ testColumn ];
73
74                    minRow = testRow;
75                    minColumn = testColumn;
76                 }
77
78                 if ( access[ testRow ][ testColumn ] == accessNumber ) {
79                    int lowestTest = nextMove( testRow, testColumn );
80                    int lowestMin = nextMove( minRow, minColumn );
81
82                    if ( lowestTest <= lowestMin ) {
83                       accessNumber = access[ testRow ][ testColumn ];
84
85                       minRow = testRow;
```

```
86                         minColumn = testColumn;
87                      }
88                   }
89
90                   // position access number tried
91                   --access[ testRow ][ testColumn ];
92                }
93
94             }  // end if
95
96             // traversing done
97             if ( accessNumber == minAccess )
98                done = true;
99
100            // make move
101            else {
102               currentRow = minRow;
103               currentColumn = minColumn;
104               board[ currentRow ][ currentColumn ] = ++moveNumber;
105            }
106
107         }  // end while
108
109         String statusBar = "The tour ended with " + moveNumber + " moves.";
110
111         if ( moveNumber == 64 )
112            statusBar += " This was a full tour!";
113         else
114            statusBar += " This was not a full tour.";
115
116         printTour();
117
118         tour.append( "\n\n" );
119         tour.append( statusBar );
120
121      }  // end method start
122
123      // checks for next move
124      public int nextMove( int row, int column )
125      {
126         int tempRow, tempColumn, tempMinRow, tempMinColumn;
127         int tempAccessNumber = accessNumber;
128         int tempAccess[][] = new int[ 8 ][ 8 ];
129
130         for ( int i = 0; i < access.length; i++ ) {
131
132            for ( int j = 0; j < access[ i ].length; j++ )
133               tempAccess[ i ][ j ] = access[ i ][ j ];
134         }
135
136         // try all possible moves
137         for ( int moveType = 0; moveType < board.length; moveType++ ) {
138
139            // new position of hypothetical moves
140            tempRow = row + vertical[ moveType ];
```

```
141          tempColumn = column + horizontal[ moveType ];
142
143          if ( validMove( tempRow, tempColumn ) ) {
144
145             // obtain access number
146             if ( access[ tempRow ][ tempColumn ] < tempAccessNumber )
147                tempAccessNumber = tempAccess[ tempRow ][ tempColumn ];
148
149             // position access number tried
150             --tempAccess[ tempRow ][ tempColumn ];
151          }
152
153       }  // end for
154
155       return tempAccessNumber;
156
157    }  // end method nextMove
158
159    // checks for valid move
160    public boolean validMove( int row, int column )
161    {
162       // NOTE: This test stops as soon as it becomes false
163       return ( row >= 0 && row < 8 && column >= 0 && column < 8
164          && board[ row ][ column ] == 0 );
165    }
166
167    // display Knight's tour path
168    public void printTour()
169    {
170       String buildString = "\t";
171
172       // display numbers for column
173       for ( int k = 0; k < 8; k++ )
174          buildString += k + "\t";
175
176       buildString += "\n\n";
177       tour.append( buildString );
178       buildString = "";
179
180       for ( int row = 0; row < board.length; row++ ) {
181
182          buildString += row + "\t";
183
184          for ( int column = 0; column < board[ row ].length; column++ )
185             buildString += board[ row ][ column ] + "\t";
186
187          buildString += "\n";
188       }
189
190       tour.append( buildString );
191
192    }  // end method printTour
193
194 }  // end class KnightD
```



7.23 (*Knight's Tour: Brute-Force Approaches*) In part (c) of Exercise 7.22, we developed a solution to the Knight's Tour problem. The approach used, called the "accessibility heuristic," generates many solutions and executes efficiently.

As computers continue to increase in power, we will be able to solve more problems with sheer computer power and relatively unsophisticated algorithms. Let us call this approach "brute-force" problem solving.

a) Use random-number generation to enable the knight to walk around the chessboard (in its legitimate L-shaped moves) at random. Your program should run one tour and print the final chessboard. How far did the knight get?

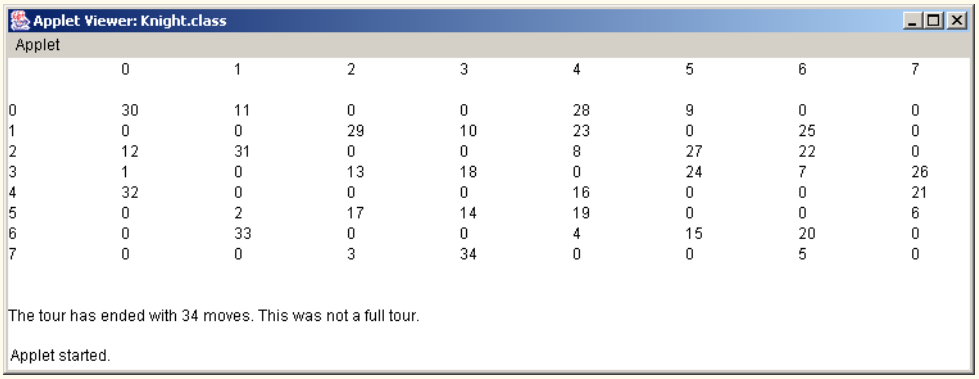
ANS:

```
1  // Exercise 7.23 Part A Solution: Knight.java
2  // Knights tour - Brute Force Approach. Uses random number
3  // generation to move around the board.
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class Knight extends JApplet {
8
9     int currentRow, currentColumn, moveType, moveNumber;
10
11    int testRow, testColumn, board[][]; // row and column positions
12
13    // moves
14    int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
15    int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
16
17    boolean done, goodMove;
18
19    JTextArea tour;
20
21    // initialize applet
22    public void init()
23    {
24       board = new int[ 8 ][ 8 ]; // chess board
25
```

```
26          // randomize initial board position
27          currentRow = ( int ) ( Math.random() * 8 );
28          currentColumn = ( int ) ( Math.random() * 8 );
29
30          board[ currentRow ][ currentColumn ] = ++moveNumber;
31
32          tour = new JTextArea( 20, 30 );
33          Container container = getContentPane();
34          container.add( tour );
35       }
36
37       // start applet
38       public void start()
39       {
40          // continue touring until finished traversing
41          while ( !done ) {
42
43             moveType = ( int ) ( Math.random() * 8 ); // random move type
44
45             // new position of hypothetical moves
46             testRow = currentRow + vertical[ moveType ];
47             testColumn = currentColumn + horizontal[ moveType ];
48
49             goodMove = validMove( testRow, testColumn );
50
51             // make move
52             if ( goodMove ) {
53
54                currentRow = testRow;
55                currentColumn = testColumn;
56                board[ currentRow ][ currentColumn ] = ++moveNumber;
57             }
58
59             // if first wasn't good, try other possible moves
60             else {
61
62                // test other 7 possible moves
63                for ( int count = 0; count < 7 && !goodMove; count++ ) {
64
65                   moveType = ++moveType % 8;
66
67                   testRow = currentRow + vertical[ moveType ];
68                   testColumn = currentColumn + horizontal[ moveType ];
69
70                   goodMove = validMove( testRow, testColumn );
71
72                   // make move
73                   if ( goodMove ) {
74
75                      currentRow = testRow;
76                      currentColumn = testColumn;
77                      board[ currentRow ][ currentColumn ] = ++moveNumber;
78                   }
79
80                } // end test for other moves
```

```
81
82             // no more valid moves available
83             if ( !goodMove )
84                done = true;
85
86          }  // end else
87
88          // tried all 64 positions
89          if ( moveNumber == 64 )
90             done = true;
91
92       }  // end while
93
94       String statusBar = "The tour has ended with " +
95          moveNumber + " moves.";
96
97       if ( moveNumber == 64 )
98          statusBar += " This was a full tour!";
99       else
100         statusBar += " This was not a full tour.";
101
102      printTour();
103      tour.append( "\n\n" + statusBar );
104
105   }  // end method start
106
107   // checks for valid move
108   public boolean validMove( int row, int column )
109   {
110      // NOTE: The tour stops as soon as this becomes false
111      return ( row >= 0 && row < 8 && column >= 0 &&
112         column < 8 && board[ row ][ column ] == 0 );
113   }
114
115   // display Knight's tour path
116   public void printTour()
117   {
118      String buildString = "\t";
119
120      // display numbers for column
121      for ( int colNum = 0; colNum < 8; colNum++ )
122         buildString += colNum + "\t";
123
124      buildString += "\n\n";
125
126      tour.append( buildString );
127      buildString = "";
128
129      for ( int row = 0; row < board.length; row++ ) {
130
131         buildString += row + "\t";
132
133         for ( int column = 0; column < board[ row ].length; column++ )
134            buildString += board[ row ][ column ] + "\t";
```

```
135
136           buildString += "\n";
137        }
138
139        tour.append( buildString );
140
141     }  // end method printTour
142
143  }  // end class Knight
```



b) Most likely, the program in part (a) produced a relatively short tour. Now modify your program to attempt 1000 tours. Use a one-dimensional array to keep track of the number of tours of each length. When your program finishes attempting the 1000 tours, it should print this information in neat tabular format. What was the best result?

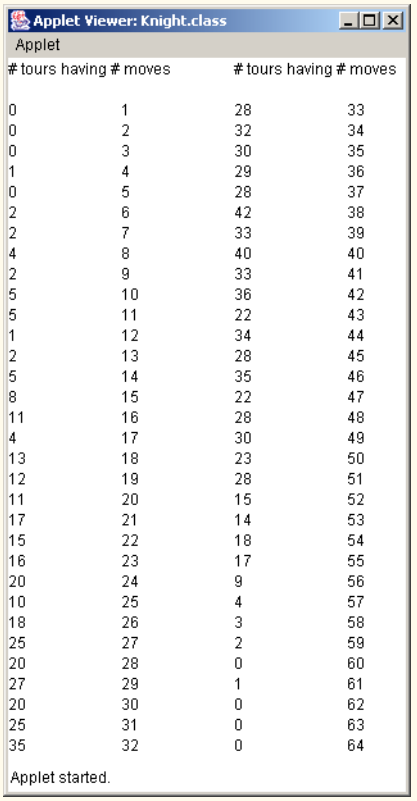
ANS:

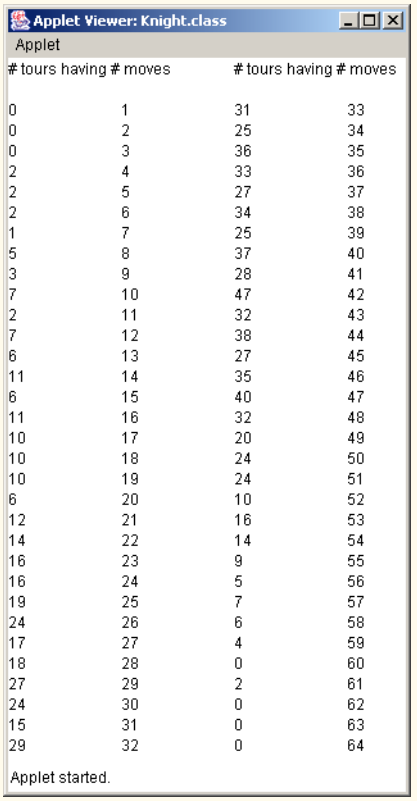
```
1  // Exercise 7.23 Part B Solution: Knight.java
2  // Knights tour program - Brute Force Approach. Use random
3  // number generation to traverse the board. ( 1000 tours )
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class Knight extends JApplet {
8
9     final int TOURS = 1000;     // number of tours
10
11    int currentRow, currentColumn, moveType, moveNumber;
12
13    int testRow, testColumn; // row and column positions
14    int board[][], moveTotal[];
15
16    // possible moves
17    int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
18    int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
19
20    boolean done, goodMove;
```

```
21    JTextArea tour;
22
23    // initialize applet
24    public void init()
25    {
26       moveTotal = new int[ 65 ]; // total number of tours per move
27
28       board = new int[ 8 ][ 8 ]; // new board
29
30       tour = new JTextArea( 20, 30 );
31       Container container = getContentPane();
32       container.add( tour );
33    }
34
35    // start applet
36    public void start()
37    {
38       for ( int k = 0; k < TOURS; k++ ) {
39
40          clearBoard();
41          moveNumber = 0;
42
43          // randomize initial board position
44          currentRow = ( int ) ( Math.random() * 8 );
45          currentColumn = ( int ) ( Math.random() * 8 );
46
47          board[ currentRow ][ currentColumn ] = ++moveNumber;
48          done = false;
49
50          // continue touring until finished traversing
51          while ( !done ) {
52
53             moveType = ( int ) ( Math.random() * 8 ); // random move type
54
55             // new position of hypothetical moves
56             testRow = currentRow + vertical[ moveType ];
57             testColumn = currentColumn + horizontal[ moveType ];
58
59             goodMove = validMove( testRow, testColumn );
60
61             // make move
62             if ( goodMove ) {
63
64                currentRow = testRow;
65                currentColumn = testColumn;
66                board[ currentRow ][ currentColumn ] = ++moveNumber;
67             }
68
69             // try other possible moves
70             else {
71
72                // test other 7 possible moves
73                for ( int count = 0; count < 7 && !goodMove; count++ ) {
74
```

```
75                      moveType = ++moveType % 8;
76
77                      testRow = currentRow + vertical[ moveType ];
78                      testColumn = currentColumn + horizontal[ moveType ];
79
80                      goodMove = validMove( testRow, testColumn );
81
82                      // make move
83                      if ( goodMove ) {
84
85                         currentRow = testRow;
86                         currentColumn = testColumn;
87                         board[ currentRow ][ currentColumn ] = ++moveNumber;
88                      }
89
90                   }  // end for
91
92                   // no more valid moves available for tour
93                   if ( !goodMove )
94                     done = true;
95
96                }  // end else
97
98                // tried all 64 positions
99                if ( moveNumber == 64 )
100                  done = true;
101
102            }  // end while
103
104            // increment number of tours having move number
105            ++moveTotal[ moveNumber ];
106
107         }  // end for
108
109         paintResults();
110
111      }  // end method start
112
113      // checks for valid move
114      public boolean validMove( int row, int column )
115      {
116         // NOTE: This test stops as soon as it becomes false
117         return ( row >= 0 && row < 8 && column >= 0 && column < 8
118            && board[ row ][ column ] == 0 );
119      }
120
121      // display results on applet window
122      public void paintResults()
123      {
124         String buildString = "# tours having # moves\t";
125         buildString += "# tours having # moves\n\n";
126
127         // display results in tabulated columns
128         for ( int row = 1; row < 33; row++ ) {
```

```
129
130          buildString +=  moveTotal[ row ] + "\t" + row + "\t";
131
132          buildString += moveTotal[ row + 32 ] + "\t" +
133             ( row + 32 ) + "\n";
134       }
135
136       tour.append( buildString );
137
138    }  // end method paintResults
139
140    // resets board
141    public void clearBoard()
142    {
143       for ( int j = 0; j < board.length; j++ )
144
145          for ( int k = 0; k < board[ j ].length; k++ )
146             board[ j ][ k ] = 0;
147    }
148
149 }  // end class Knight
```





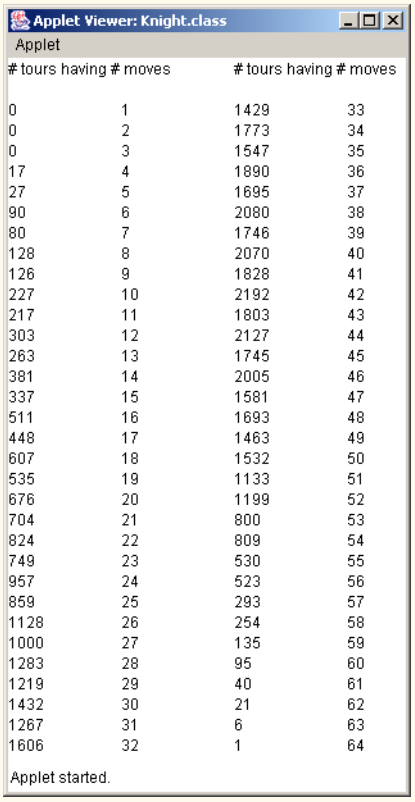
c) Most likely, the program in part (b) gave you some "respectable" tours, but no full tours. Now let your program run until it produces a full tour. (*Caution*: This version of the program could run for hours on a powerful computer.) Once again, keep a table of the number of tours of each length, and print this table when the first full tour is found. How many tours did your program attempt before producing a full tour? How much time did it take?

ANS:

```
1  // Exercise 7.23 Part C Solution: Knight.java
2  // Knights tour program - Brute Force Approach. Use random
3  // number generation to traverse the board. ( 1000 tours )
4  import java.awt.*;
5  import javax.swing.*;
6
7  public class Knight extends JApplet {
8
9     int currentRow, currentColumn, moveType, moveNumber;
10
11    boolean fullTour = false; // tour completion flag
12
13    int testRow, testColumn; // row and column positions
14    int board[][], moveTotal[];
15
16    // possible moves
17    int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
18    int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
19
20    boolean done, goodMove;
21    JTextArea tour;
22
23    // initialize applet
24    public void init()
25    {
26       moveTotal = new int[ 65 ]; // total number of tours per move
27
28       board = new int[ 8 ][ 8 ]; // new board
29
30       tour = new JTextArea( 20, 30 );
31       Container container = getContentPane();
32       container.add( tour );
33    }
34
35    // start applet
36    public void start()
37    {
38       while (!fullTour) {
39
40          clearBoard();
41          moveNumber = 0;
42
43          // randomize initial board position
44          currentRow = ( int ) ( Math.random() * 8 );
45          currentColumn = ( int ) ( Math.random() * 8 );
46
47          board[ currentRow ][ currentColumn ] = ++moveNumber;
```

```
48            done = false;
49
50            // continue touring until finished traversing
51            while ( !done ) {
52
53               moveType = ( int ) ( Math.random() * 8 ); // random move type
54
55               // new position of hypothetical moves
56               testRow = currentRow + vertical[ moveType ];
57               testColumn = currentColumn + horizontal[ moveType ];
58
59               goodMove = validMove( testRow, testColumn );
60
61               // make move
62               if ( goodMove ) {
63
64                  currentRow = testRow;
65                  currentColumn = testColumn;
66                  board[ currentRow ][ currentColumn ] = ++moveNumber;
67               }
68
69               // try other possible moves
70               else {
71
72                  // test other 7 possible moves
73                  for ( int count = 0; count < 7 && !goodMove; count++ ) {
74
75                     moveType = ++moveType % 8;
76
77                     testRow = currentRow + vertical[ moveType ];
78                     testColumn = currentColumn + horizontal[ moveType ];
79
80                     goodMove = validMove( testRow, testColumn );
81
82                     // make move
83                     if ( goodMove ) {
84
85                        currentRow = testRow;
86                        currentColumn = testColumn;
87                        board[ currentRow ][ currentColumn ] = ++moveNumber;
88                     }
89
90                  }  // end for
91
92                  // no more valid moves available for tour
93                  if ( !goodMove )
94                    done = true;
95
96               }  // end else
97
98               // tried all 64 positions
99               if ( moveNumber == 64 ) {
100                  done = true;
101                  fullTour = true;
102               }
```

```
103
104          }  // end while
105
106          // increment number of tours having move number
107          ++moveTotal[ moveNumber ];
108
109       }  // end while
110
111       paintResults();
112
113    }  // end method start
114
115    // checks for valid move
116    public boolean validMove( int row, int column )
117    {
118       // NOTE: This test stops as soon as it becomes false
119       return ( row >= 0 && row < 8 && column >= 0 && column < 8
120          && board[ row ][ column ] == 0 );
121    }
122
123    // display results on applet window
124    public void paintResults()
125    {
126       String buildString = "# tours having # moves\t";
127       buildString += "# tours having # moves\n\n";
128
129       // display results in tabulated columns
130       for ( int row = 1; row < 33; row++ ) {
131
132          buildString +=  moveTotal[ row ] + "\t" + row + "\t";
133
134          buildString += moveTotal[ row + 32 ] + "\t" +
135             ( row + 32 ) + "\n";
136       }
137
138       tour.append( buildString );
139
140    }  // end method paintResults
141
142    // resets board
143    public void clearBoard()
144    {
145       for ( int j = 0; j < board.length; j++ )
146
147          for ( int k = 0; k < board[ j ].length; k++ )
148             board[ j ][ k ] = 0;
149    }
150
151 }  // end class Knight
```



d) Compare the brute-force version of the Knight's Tour with the accessibility-heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility-heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute-force approach? Argue the pros and cons of brute-force problem solving in general.

7.24 (*Eight Queens*) Another puzzler for chess buffs is the Eight Queens problem, which asks the following: Is it possible to place eight queens on an empty chessboard so that no queen is "attacking" any other (i.e., no two queens are in the same row, in the same column or along the same diagonal)? Use the thinking developed in Exercise 7.22 to formulate a heuristic for solving the Eight Queens problem. Run your program. (*Hint*: It is possible to assign a value to each square of the chessboard to indicate how many squares of an empty chessboard are "eliminated" if a queen is placed in that square. Each of the corners would be assigned the value 22, as demonstrated by Fig. 7.25. Once these "elimination numbers" are placed in all 64 squares, an appropriate heuristic might be as follows: Place the next queen in the square with the smallest elimination number. Why is this strategy intuitively appealing?

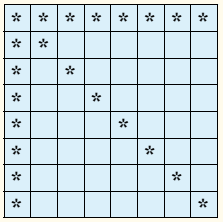


Fig. 7.25 The 22 squares eliminated by placing a queen in the upper left corner.

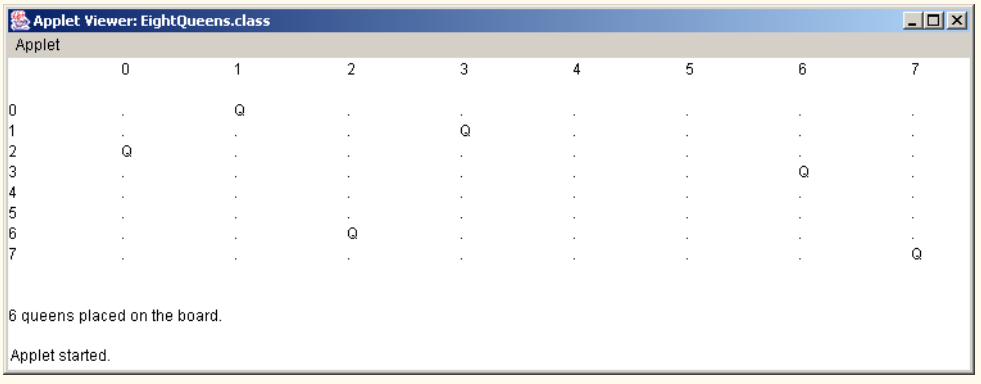
ANS:

```
1  // Exercise 7.24 Solution: EightQueens.java
2  // EightQueens - heuristic version
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class EightQueens extends JApplet {
7
8     int currentRow, currentColumn, queens;
9
10    char board[][];    // gameboard
11
12    // accessibility values for each board position
13    int access[][] = { { 22, 22, 22, 22, 22, 22, 22, 22 },
14                       { 22, 24, 24, 24, 24, 24, 24, 22 },
15                       { 22, 24, 26, 26, 26, 26, 24, 22 },
16                       { 22, 24, 26, 28, 28, 26, 24, 22 },
17                       { 22, 24, 26, 28, 28, 26, 24, 22 },
18                       { 22, 24, 26, 26, 26, 26, 24, 22 },
19                       { 22, 24, 24, 24, 24, 24, 24, 22 },
20                       { 22, 22, 22, 22, 22, 22, 22, 22 } };
21
22    int minAccess, accessNumber; // accessibility values
23
24    boolean done;
25    JTextArea output;
26    final char queenChar = 'Q';
27
28    // initialize applet
29    public void init()
30    {
31       minAccess = 50;
32       board = new char[ 8 ][ 8 ];
33
34       // initialize board to '.'
35       for ( int i = 0; i < board.length; i++ ) {
36
37          for ( int j = 0; j < board[ i ].length; j++ )
```

```
38                board[ i ][ j ] = '.';
39             }
40
41             // randomize initial first queen position
42             currentRow = ( int ) ( Math.random() * 8 );
43             currentColumn = ( int ) ( Math.random() * 8 );
44
45             board[ currentRow ][ currentColumn ] = queenChar;
46             queens = 1;
47
48             updateAccess( currentRow, currentColumn ); // update access
49
50             done = false;
51
52             output = new JTextArea( 20, 30 );
53             Container container = getContentPane();
54             container.add( output );
55          }
56
57          // start placing queens
58          public void start()
59          {
60             // continue until finished traversing
61             while ( !done ) {
62                accessNumber = minAccess;
63
64                // find square with the smallest elimination number
65                for ( int row = 0; row < board.length; row++ ) {
66
67                   for ( int col = 0; col < board.length; col++ ) {
68
69                      // obtain access number
70                      if ( access[ row ][ col ] < accessNumber ) {
71
72                         accessNumber = access[ row ][ col ];
73                         currentRow = row;
74                         currentColumn = col;
75                      }
76                   }
77                }
78
79                // traversing done
80                if ( accessNumber == minAccess )
81                   done = true;
82
83                // mark 'Q' in current location
84                else {
85                   board[ currentRow ][ currentColumn ] = queenChar;
86                   updateAccess( currentRow, currentColumn );
87                   queens++;
88                }
89
90             }  // end while
91
```

```
92          printBoard();
93
94       }  // end method start
95
96       // update access array
97       public void updateAccess( int row, int column )
98       {
99          for ( int i = 0; i < 8; i++ ) {
100
101            // set elimination numbers to 50 in the row occupied by the queen
102            access[ row ][ i ] = 50;
103
104            // set elimination numbers to 50
105            // in the column occupied by the queen
106            access[ i ][ column ] = 50;
107         }
108
109         // set elimination numbers to 50 in diagonals occupied by the queen
110         updateDiagonals( row, column );
111      }
112
113      // place * in diagonals of position in all 4 directions
114      public void updateDiagonals( int rowValue, int colValue )
115      {
116         int row = rowValue, column = colValue;
117
118         // upper left diagonal
119         for ( int diagonal = 0; diagonal < 8 &&
120            validMove( --row, --column ); diagonal++ )
121            access[ row ][ column ] = 50;
122
123         row = rowValue;
124         column = colValue;
125
126         // upper right diagonal
127         for ( int diagonal = 0; diagonal < 8 &&
128            validMove( --row, ++column ); diagonal++ )
129            access[ row ][ column ] = 50;
130
131         row = rowValue;
132         column = colValue;
133
134         // lower left diagonal
135         for ( int diagonal = 0; diagonal < 8 &&
136            validMove( ++row, --column ); diagonal++ )
137            access[ row ][ column ] = 50;
138
139         row = rowValue;
140         column = colValue;
141
142         // lower right diagonal
143         for ( int diagonal = 0; diagonal < 8 &&
144            validMove( ++row, ++column ); diagonal++ )
145            access[ row ][ column ] = 50;
```

```
146
147    }  // end method updateDiagonals
148
149    // check for valid move
150    public boolean validMove( int row, int column )
151    {
152       return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
153    }
154
155    // display the board
156    public void printBoard()
157    {
158       String buildString = "\t";
159
160       // display numbers for column
161       for ( int k = 0; k < 8; k++ )
162          buildString += k + "\t";
163
164       buildString += "\n\n";
165       output.append( buildString );
166       buildString = "";
167
168       for ( int row = 0; row < board.length; row++ ) {
169
170          buildString += row + "\t";
171
172          for ( int column = 0; column < board[ row ].length; column++ )
173             buildString += board[ row ][ column ] + "\t";
174
175          buildString += "\n";
176       }
177
178       buildString += "\n\n" + queens + " queens placed on the board.";
179
180       output.append( buildString );
181
182    }  // end method printBoard
183
184 }  // end class EightQueens
```



7.25 (*Eight Queens: Brute-Force Approaches*) In this exercise, you will develop several brute-force approaches to solving the Eight Queens problem introduced in Exercise 7.24.

a) Use the random brute-force technique developed in Exercise 7.23 to solve the Eight Queens problem.

ANS:

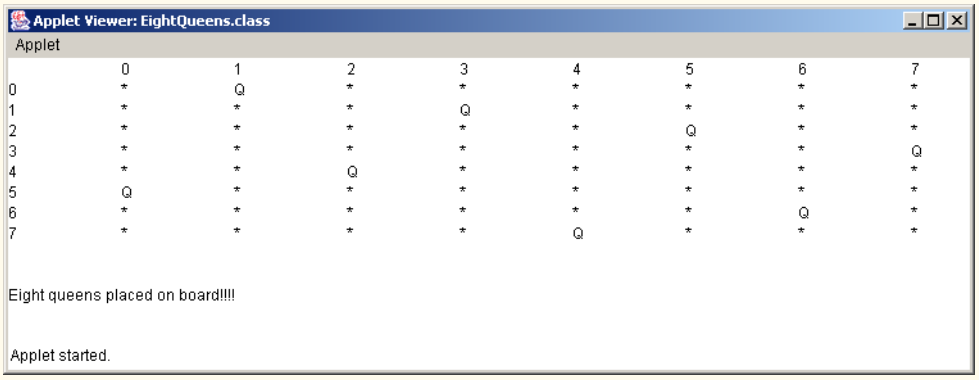
```
1  // Exercise 7.25 Solution: EightQueens.java
2  // Applet uses a brute force approach to solve the eight queens problem.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class EightQueens extends JApplet {
7
8     char board[][];   // chess board
9     int queens;       // number of queens placed
10
11    JTextArea output;
12
13    // initialize applet
14    public void init()
15    {
16       output = new JTextArea( 20, 30 );
17       Container container = getContentPane();
18       container.add( output );
19
20       // repeat until solved
21       while (queens < 8) {
22          board = new char[ 8 ][ 8 ];
23          placeQueens();
24       }
25       printBoard();
26    }
27
28    // display board
29    public void printBoard()
30    {
31       String buildString = "\t";
32
33       for ( int j = 0; j < 8; j++ )
34          buildString += j + "\t";
35
36       buildString += "\n";
37
38       for ( int row = 0; row < board.length; row++ ) {
39
40          buildString += row + "\t";
41
42          for ( int column = 0; column < board[ row ].length; column++ )
43             buildString += board[ row ][ column ] + "\t";
44
45          buildString += "\n";
46       }
47
48       buildString += "\n\n";
```

```
49          buildString += "Eight queens placed on board!!!!";
50
51          output.append( buildString );
52
53       }  // end method printBoard
54
55       // check for valid move
56       public boolean validMove( int row, int column )
57       {
58          return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
59       }
60
61       // check if any squares left
62       public boolean availableSquare()
63       {
64          for ( int row = 0; row < board.length; row++ )
65
66             for ( int col = 0; col < board[ row ].length; col++ )
67
68                if ( board[ row ][ col ] == '\0' )
69                   return true;  // at least one available square
70
71          return false;   // no available squares
72
73       }  // end method availableSquare
74
75       // place queens on board
76       public void placeQueens()
77       {
78          queens = 0;
79          char queenChar = 'Q';
80          int rowMove, colMove;   // board move
81          boolean done = false;   // indicates if all squares filled
82
83          // continue placing queens until no more squares
84          // or not all queens placed
85          while ( !done ) {
86
87             // randomize move
88             rowMove = ( int ) ( Math.random() * 8 );
89             colMove = ( int ) ( Math.random() * 8 );
90
91             // if valid move, place queen and mark off conflict squares
92             if ( queenCheck( rowMove, colMove ) == true ) {
93
94                board[ rowMove ][ colMove ] = queenChar;
95                xConflictSquares( rowMove, colMove );
96                ++queens;
97             }
98
99             // done when no more squares left
100            if ( !availableSquare() )
101               done = true;
102         }
```

```
103
104    }  // end method placeQueens
105
106    // conflicting square marked with *
107    public void xConflictSquares( int row, int col )
108    {
109       for ( int i = 0; i < 8; i++ ) {
110
111          // place a '*' in the row occupied by the queen
112          if ( board[ row ][ i ] == '\0' )
113             board[ row ][ i ] = '*';
114
115          // place a '*' in the col occupied by the queen
116          if ( board[ i ][ col ] == '\0' )
117             board[ i ][ col ] = '*';
118       }
119
120       // place a '*' in the diagonals occupied by the queen
121       xDiagonals( row, col );
122
123    }  // end method xConflictSquares
124
125    // check if queens can "attack" each other
126    public boolean queenCheck( int rowValue, int colValue )
127    {
128       int row = rowValue, column = colValue;
129
130       // check row and column for a queen
131       for ( int position = 0; position < 8; position++ )
132
133          if ( board[ row ][ position ] == 'Q' ||
134             board[ position ][ column ] == 'Q' )
135
136             return false;
137
138       // check upper left diagonal for a queen
139       for ( int square = 0; square < 8 &&
140          validMove( --row, --column ); square++ )
141          if ( board[ row ][ column ] == 'Q' )
142             return false;
143
144       row = rowValue;
145       column = colValue;
146
147       // check upper right diagonal for a queen
148       for ( int diagonal = 0; diagonal < 8 &&
149          validMove( --row, ++column ); diagonal++ )
150
151          if ( board[ row ][ column ] == 'Q' )
152             return false;
153
154       row = rowValue;
155       column = colValue;
156
```

```
157         // check lower left diagonal for a queen
158         for ( int diagonal = 0; diagonal < 8 &&
159            validMove( ++row, --column ); diagonal++ )
160
161            if ( board[ row ][ column ] == 'Q' )
162               return false;
163
164         row = rowValue;
165         column = colValue;
166
167         // check lower right diagonal for a queen
168         for ( int diagonal = 0; diagonal < 8 &&
169            validMove( ++row, ++column ); diagonal++ )
170
171            if ( board[ row ][ column ] == 'Q' )
172               return false;
173
174         return true;   // no queen in conflict
175
176      }  // end method queenCheck
177
178      // place * in diagonals of position in all 4 directions
179      public void xDiagonals( int rowValue, int colValue )
180      {
181         int row = rowValue, column = colValue;
182
183         // upper left diagonal
184         for ( int diagonal = 0; diagonal < 8 &&
185            validMove( --row, --column ); diagonal++ )
186            board[ row ][ column ] = '*';
187
188         row = rowValue;
189         column = colValue;
190
191         // upper right diagonal
192         for ( int diagonal = 0; diagonal < 8 &&
193            validMove( --row, ++column ); diagonal++ )
194            board[ row ][ column ] = '*';
195
196         row = rowValue;
197         column = colValue;
198
199         // lower left diagonal
200         for ( int diagonal = 0; diagonal < 8 &&
201            validMove( ++row, --column ); diagonal++ )
202            board[ row ][ column ] = '*';
203
204         row = rowValue;
205         column = colValue;
206
207         // lower right diagonal
208         for ( int diagonal = 0; diagonal < 8 &&
209            validMove( ++row, ++column ); diagonal++ )
210            board[ row ][ column ] = '*';
```

```
211
212     }  // end method xDiagonals
213
214  }  // end class EightQueens
```



b) Use an exhaustive technique (i.e., try all possible combinations of eight queens on the chessboard) to solve the Eight Queens problem.

ANS:

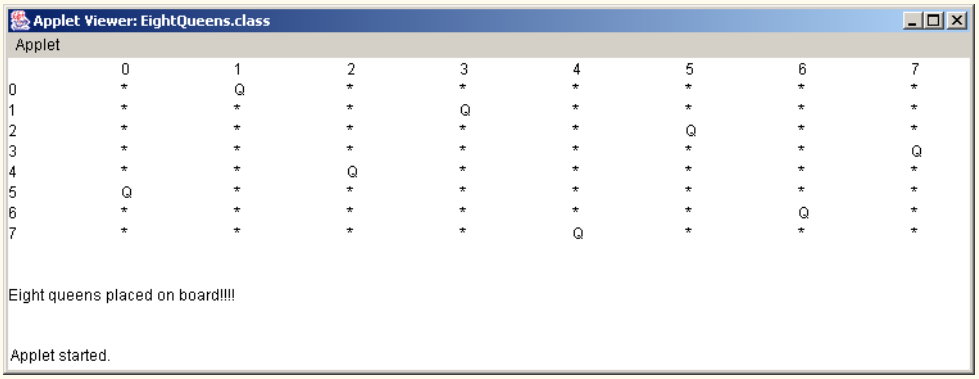
```
1  // Exercise 7.25 Solution: EightQueens.java
2  // Applet uses a brute force approach to solve the eight queens problem
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class EightQueens extends JApplet {
7
8     char board[][] = new char[ 8 ][ 8 ];   // chess board
9     int queens;       // number of queens placed
10
11    JTextArea output;
12
13    // initialize applet
14    public void init()
15    {
16       output = new JTextArea( 20, 30 );
17       Container container = getContentPane();
18       container.add( output );
19
20       placeQueens();
21
22       printBoard(); // display results
23    }
24
25    // display board
26    public void printBoard()
27    {
28       String buildString = "\t";
29
```

```
30          for ( int j = 0; j < 8; j++ )
31             buildString += j + "\t";
32
33          buildString += "\n";
34
35          for ( int row = 0; row < board.length; row++ ) {
36
37             buildString += row + "\t";
38
39             for ( int column = 0; column < board[ row ].length; column++ )
40                buildString += board[ row ][ column ] + "\t";
41
42             buildString += "\n";
43          }
44
45          buildString += "\n\n";
46          buildString += "Eight queens placed on board!!!!";
47
48          output.append( buildString );
49
50       }  // end method printBoard
51
52       // check for valid move
53       public boolean validMove( int row, int column )
54       {
55          return ( row >= 0 && row < 8 && column >= 0 && column < 8 );
56       }
57
58       // check if any squares left
59       public boolean availableSquare()
60       {
61          for ( int row = 0; row < board.length; row++ )
62
63             for ( int col = 0; col < board[ row ].length; col++ )
64
65                if ( board[ row ][ col ] == '\0' )
66                   return true;  // at least one available square
67
68          return false;   // no available squares
69
70       }  // end method availableSquare
71
72       // place queens on board
73       public void placeQueens()
74       {
75          char queenChar = 'Q'; // indicates queen marker
76
77          for ( int firstQueenRow = 0;
78                firstQueenRow < board[ 0 ].length && queens < 8;
79                firstQueenRow++ ) {
80             for ( int firstQueenCol = 0;
81                   firstQueenCol < board[ 0 ].length && queens < 8;
82                   firstQueenCol++ ) {
83
```

```
84             board = new char[ 8 ][ 8 ]; // start on new board
85             queens = 0;
86
87             // place first queen at starting position
88             board[ firstQueenRow ][ firstQueenCol ] = queenChar;
89             xConflictSquares( firstQueenRow, firstQueenCol );
90             ++queens;
91
92             // remaining queens will be placed in board
93
94             boolean done = false;   // indicates if all squares filled
95
96             // try all possible locations on board
97             for ( int rowMove = 0;
98                rowMove < board[ 0 ].length && !done; rowMove++ ) {
99                for ( int colMove = 0;
100                  colMove < board[0].length && !done; colMove++ ) {
101
102                  // if valid move, place queen
103                  // and mark off conflict squares
104                  if ( queenCheck( rowMove, colMove ) == true ) {
105
106                     board[ rowMove ][ colMove ] = queenChar;
107                     xConflictSquares( rowMove, colMove );
108                     ++queens;
109                  }
110
111                  // done when no more squares left
112                  if ( !availableSquare() ) {
113                     done = true;
114                  }
115
116               } // end for colMove
117
118            } // end for rowMove
119
120         } // end for firstQueenCol
121
122      } // end for firstQueenRow
123
124   }  // end method placeQueens
125
126   // conflicting square marked with *
127   public void xConflictSquares( int row, int col )
128   {
129      for ( int i = 0; i < 8; i++ ) {
130
131         // place a '*' in the row occupied by the queen
132         if ( board[ row ][ i ] == '\0' )
133            board[ row ][ i ] = '*';
134
135         // place a '*' in the col occupied by the queen
136         if ( board[ i ][ col ] == '\0' )
137            board[ i ][ col ] = '*';
138      }
```

```
139
140         // place a '*' in the diagonals occupied by the queen
141         xDiagonals( row, col );
142
143      }  // end method xConflictSquares
144
145      // check if queens can "attack" each other
146      public boolean queenCheck( int rowValue, int colValue )
147      {
148         int row = rowValue, column = colValue;
149
150         // check row and column for a queen
151         for ( int position = 0; position < 8; position++ )
152
153            if ( board[ row ][ position ] == 'Q' ||
154               board[ position ][ column ] == 'Q' )
155
156               return false;
157
158         // check upper left diagonal for a queen
159         for ( int square = 0; square < 8 &&
160            validMove( --row, --column ); square++ )
161            if ( board[ row ][ column ] == 'Q' )
162               return false;
163
164         row = rowValue;
165         column = colValue;
166
167         // check upper right diagonal for a queen
168         for ( int diagonal = 0; diagonal < 8 &&
169            validMove( --row, ++column ); diagonal++ )
170
171            if ( board[ row ][ column ] == 'Q' )
172               return false;
173
174         row = rowValue;
175         column = colValue;
176
177         // check lower left diagonal for a queen
178         for ( int diagonal = 0; diagonal < 8 &&
179            validMove( ++row, --column ); diagonal++ )
180
181            if ( board[ row ][ column ] == 'Q' )
182               return false;
183
184         row = rowValue;
185         column = colValue;
186
187         // check lower right diagonal for a queen
188         for ( int diagonal = 0; diagonal < 8 &&
189            validMove( ++row, ++column ); diagonal++ )
190
191            if ( board[ row ][ column ] == 'Q' )
192               return false;
```

```
193
194         return true;   // no queen in conflict
195
196      }  // end method queenCheck
197
198      // place * in diagonals of position in all 4 directions
199      public void xDiagonals( int rowValue, int colValue )
200      {
201         int row = rowValue, column = colValue;
202
203         // upper left diagonal
204         for ( int diagonal = 0; diagonal < 8 &&
205            validMove( --row, --column ); diagonal++ )
206            board[ row ][ column ] = '*';
207
208         row = rowValue;
209         column = colValue;
210
211         // upper right diagonal
212         for ( int diagonal = 0; diagonal < 8 &&
213            validMove( --row, ++column ); diagonal++ )
214            board[ row ][ column ] = '*';
215
216         row = rowValue;
217         column = colValue;
218
219         // lower left diagonal
220         for ( int diagonal = 0; diagonal < 8 &&
221            validMove( ++row, --column ); diagonal++ )
222            board[ row ][ column ] = '*';
223
224         row = rowValue;
225         column = colValue;
226
227         // lower right diagonal
228         for ( int diagonal = 0; diagonal < 8 &&
229            validMove( ++row, ++column ); diagonal++ )
230            board[ row ][ column ] = '*';
231
232      }  // end method xDiagonals
233
234   }  // end class EightQueens
```



c) Why might the exhaustive brute-force approach not be appropriate for solving the Knight's Tour problem?

d) Compare and contrast the random brute-force and exhaustive brute-force approaches.

7.26 (*Knight's Tour: Closed-Tour Test*) In the Knight's Tour (Exercise 7.22), a full tour occurs when the knight makes 64 moves, touching each square of the chessboard once and only once. A closed tour occurs when the 64th move is one move away from the square in which the knight started the tour. Modify the program you wrote in Exercise 7.22 to test for a closed tour if a full tour has occurred.

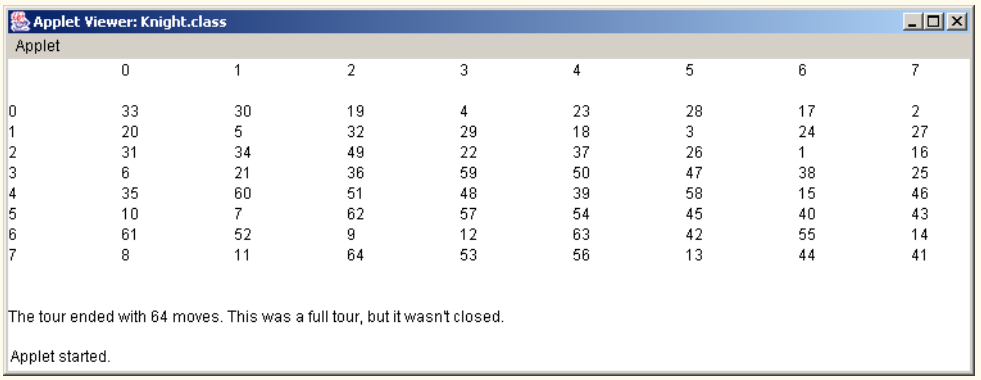
ANS:

```
1  // Exercise 7.26 Solution: Knight.java
2  // Knight's Tour - Closed Tour version
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Knight extends JApplet {
7
8     int currentRow, currentColumn, moveNumber, firstMoveRow,
9        firstMoveColumn;
10    boolean done = false, closed = false;
11    JTextArea tour;
12    int board[][];
13
14    // accessibility values for each board position
15    int access[][] = { { 2, 3, 4, 4, 4, 4, 3, 2 },
16                       { 3, 4, 6, 6, 6, 6, 4, 3 },
17                       { 4, 6, 8, 8, 8, 8, 6, 4 },
18                       { 4, 6, 8, 8, 8, 8, 6, 4 },
19                       { 4, 6, 8, 8, 8, 8, 6, 4 },
20                       { 4, 6, 8, 8, 8, 8, 6, 4 },
21                       { 3, 4, 6, 6, 6, 6, 4, 3 },
22                       { 2, 3, 4, 4, 4, 4, 3, 2 } };
23
24    // row and column positions
25    int testRow, testColumn, count, minRow, minColumn;
26
```

```
27      int minAccess, accessNumber; // accessibility values
28
29      // moves
30      int horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 };
31      int vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
32
33      // initialize applet
34      public void init()
35      {
36         minAccess = 9;
37         board = new int[ 8 ][ 8 ];
38
39         // randomize initial board position
40         currentRow = ( int ) ( Math.random() * 8 );
41         currentColumn = ( int ) ( Math.random() * 8 );
42
43         firstMoveRow = currentRow;
44         firstMoveColumn = currentColumn;
45
46         board[ currentRow ][ currentColumn ] = ++moveNumber;
47
48         tour = new JTextArea( 20, 30 );
49         Container container = getContentPane();
50         container.add( tour );
51      }
52
53      // start touring
54      public void start()
55      {
56         // continue touring until finished traversing
57         while ( !done ) {
58
59            accessNumber = minAccess;
60
61            // try all possible moves
62            for ( int moveType = 0; moveType < board.length; moveType++ ) {
63
64               // new position of hypothetical moves
65               testRow = currentRow + vertical[ moveType ];
66               testColumn = currentColumn + horizontal[ moveType ];
67
68               if ( validMove( testRow, testColumn ) ) {
69
70                  // obtain access number
71                  if ( access[ testRow ][ testColumn ] < accessNumber ) {
72
73                     accessNumber = access[ testRow ][ testColumn ];
74
75                     minRow = testRow;
76                     minColumn = testColumn;
77                  }
78
79                  // position access number tried
80                  --access[ testRow ][ testColumn ];
81               }
```

```
82
83             }  // end if
84
85             // traversing done
86             if ( accessNumber == minAccess )
87                done = true;
88
89             // make move
90             else {
91
92                currentRow = minRow;
93                currentColumn = minColumn;
94                board[ currentRow ][ currentColumn ] = ++moveNumber;
95             }
96
97          }  // end while
98
99          String statusBar = "The tour ended with " + moveNumber +
100            " moves.";
101
102         // if finished tour
103         if ( moveNumber == 64 )
104
105            // test all 8 possible moves to check if move
106            // would position knight on first move
107            for ( int moveType = 0; moveType < 8; moveType++ ) {
108
109               testRow = currentRow + vertical[ moveType ];
110               testColumn = currentColumn + horizontal[ moveType ];
111
112               // if one move away from initial move
113               if ( testRow == firstMoveRow &&
114                  testColumn == firstMoveColumn ) {
115
116                  closed = true;
117                  break;
118               }
119
120            }  // end for
121
122         if ( moveNumber == 64 && closed == true )
123            statusBar += "This was a CLOSED tour!";
124
125         else if ( moveNumber == 64 )
126            statusBar += " This was a full tour, but it wasn't closed.";
127
128         else
129            statusBar += " This was not a full tour.";
130
131         printTour();
132
133         tour.append( "\n\n" + statusBar );
134
135      }  // end method start
```

```
136
137    // checks for valid move
138    public boolean validMove( int row, int column )
139    {
140       // NOTE: This test stops as soon as it becomes false
141       return ( row >= 0 && row < 8 && column >= 0 && column < 8
142          && board[ row ][ column ] == 0 );
143    }
144
145    // display Knight's tour path
146    public void printTour()
147    {
148       String buildString = "\t";
149
150       // display numbers for column
151       for ( int rowNum = 0; rowNum < 8; rowNum++ )
152          buildString += rowNum + "\t";
153
154       buildString += "\n\n";
155       tour.append( buildString );
156       buildString = "";
157
158       for ( int row = 0; row < board.length; row++ ) {
159
160          buildString += row + "\t";
161
162          for ( int column = 0; column < board[ row ].length; column++ )
163             buildString += board[ row ][ column ] + "\t";
164
165          buildString += "\n";
166       }
167
168       tour.append( buildString );
169
170    }  // end method printTour
171
172 }  // end class Knight
```



7.27 (*The Sieve of Eratosthenes*) A prime number is any integer that is evenly divisible only by itself and one. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

a) Create a primitive type `boolean` array with all elements initialized to `true`. Array elements with prime indices will remain `true`. All other array elements will eventually be set to `false`.

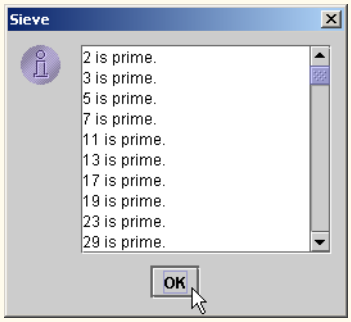
b) Starting with array index 2, determine whether a given element is `true`. If so, loop through the remainder of the array and set to `false` every element whose index is a multiple of the index for the element with value `true`. Then continue the process with the next element with value `true`. For array index 2, all elements beyond element 2 in the array that have indices which are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to `false`; for array index 3, all elements beyond element 3 in the array that have indices which are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to `false`; and so on.

When this process is complete, the array elements that are still `true` indicate that the index is a prime number. These indices can be displayed. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 2 and 999. Ignore elements 0 and 1 of the array.

ANS:

```
1  // Exercise 7.27 Solution: Sieve.java
2  // Sieve of Eratosthenes
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Sieve {
7
8     public static void main( String args[] )
9     {
10       int count = 0;
11       String result = "";
12       JTextArea output = new JTextArea( 10, 15 );
13       JScrollPane scroller = new JScrollPane( output );
14
15       int array[] = new int[ 1000 ];
16
17       // initialize all array values to 1
18       for ( int index = 0; index < array.length; index++ )
19          array[ index ] = 1;
20
21       // starting at the third value, cycle through the array and put 0
22       // as the value of any greater number that is a multiple
23       for ( int i = 2; i < array.length; i++ )
24
25          if ( array[ i ] == 1 )
26
27             for ( int j = i + 1; j < array.length; j++ )
28
29                if ( j % i == 0 )
30                   array[ j ] = 0;
31
32       // cycle through the array one last time to print all primes
33       for ( int index = 2; index < array.length; index++ )
34
35          if ( array[ index ] == 1 ) {
36             result += index + " is prime.\n";
```

```
37                ++count;
38             }
39
40          result += "\n" + count + " primes found.";
41
42          output.setText( result );
43
44          JOptionPane.showMessageDialog( null, scroller, "Sieve",
45             JOptionPane.INFORMATION_MESSAGE );
46
47          System.exit( 0 );
48
49       }  // end method main
50
51    }  // end class Sieve
```



7.28 (*Bucket Sort*) A bucket sort begins with a one-dimensional array of positive integers to be sorted and a two-dimensional array of integers with rows indexed from 0 to 9 and columns indexed from 0 to $n - 1$, where n is the number of values to be sorted. Each row of the two-dimensional array is referred to as a bucket. Write an applet containing a method called `bucketSort` that takes an integer array as an argument and performs as follows:

a) Place each value of the one-dimensional array into a row of the bucket array, based on the value's "ones" digit. For example, 97 is placed in row 7, 3 is placed in row 3 and 100 is placed in row 0. This procedure is called a "distribution pass."

b) Loop through the bucket array row by row, and copy the values back to the original array. This procedure is called a "gathering pass." The new order of the preceding values in the one-dimensional array is 100, 3 and 97.

c) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.).

On the second (tens digit) pass, 100 is placed in row 0, 3 is placed in row 0 (because 3 has no tens digit) and 97 is placed in row 9. After the gathering pass, the order of the values in the one-dimensional array is 100, 3 and 97. On the third (hundreds digit) pass, 100 is placed in row 1, 3 is placed in row 0 and 97 is placed in row 0 (after the 3). After this last gathering pass, the original array is in sorted order.

Note that the two-dimensional array of buckets is 10 times the length of the integer array being sorted. This sorting technique provides better performance than a bubble sort, but requires much more memory; the bubble sort requires space for only one additional element of data. This comparison is an example of the space–time trade-off: The bucket sort uses more memory than the bubble sort, but performs better. This version of the bucket sort requires copying all the data back to the

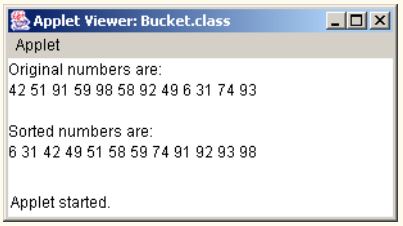
original array on each pass. Another possibility is to create a second two-dimensional bucket array and repeatedly swap the data between the two bucket arrays.

ANS:

```
1  // Exercise 7.28 Solution: Bucket.java
2  // Applet does a bucket sort
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Bucket extends JApplet {
7
8     static final int SIZE = 12;
9     JTextArea output;
10
11    // initialize applet
12    public void init()
13    {
14       output = new JTextArea( 5, 50 );
15       getContentPane().add( output );
16    }
17
18    // initialize for bucket sort
19    public void start()
20    {
21       int array[] = new int [ SIZE ]; // integer array to be sorted
22
23       // randomize integers
24       for ( int z = 0; z < array.length; z++ )
25          array[ z ] = ( int ) ( Math.random() * 100 );
26
27       String buildString = "Original numbers are:\n";
28
29       // display original numbers
30       for ( int z = 0; z < array.length; z++ )
31           buildString += array[ z ] + " ";
32
33       bucketSort( array );
34
35       buildString += "\n\nSorted numbers are:\n";
36
37       // display sorted numbers
38       for ( int z = 0; z < array.length; z++ )
39          buildString += array[ z ] + " ";
40
41       buildString += "\n";
42       output.append( buildString );
43
44    }  // end method start
45
46    // perform bucket sort algorithm on array
47    public void bucketSort( int array[] )
48    {
49       int totalDigits;
50
```

```
51          // bucket array where numbers will be placed according to digits
52          int pail[][] = new int [ 10 ][ SIZE ];
53
54          totalDigits = numberOfDigits( array );
55
56          // go through all digit places and sort each number
57          // according to digit place value
58          for ( int pass = 1; pass <= totalDigits; pass++ ) {
59
60             distributeElements( array, pail, pass ); // distribution pass
61
62             collectElements( array, pail ); // gathering pass
63
64             // set all bucket contents to 0
65             if ( pass != totalDigits )
66                emptyBucket( pail );
67          }
68       }
69
70       // determine number of digits in the largest number
71       public int numberOfDigits( int array[] )
72       {
73          int largest = array[ 0 ];
74          int digits = 0;
75
76          // obtain largest number
77          for ( int i = 1; i < SIZE; i++ )
78
79             if ( array[ i ] > largest )
80                largest = array[ i ];
81
82          // count number of digits in largest number
83          while ( largest != 0 ) {
84
85             ++digits;
86             largest /= 10;
87          }
88
89          return digits;
90
91       }  // end method numberOfDigits
92
93       // distribute elements into buckets based on specified digit
94       public void distributeElements( int array[], int pail[][], int digit )
95       {
96          int divisor = 10, bucketNumber, elementNumber;
97
98          // determine the divisor used to get specific digit
99          for ( int i = 1; i < digit; i++ )
100            divisor *= 10;
101
102         for ( int i = 0; i < SIZE; i++ ) {
103
104            // bucketNumber example for hundreds digit:
105            // ( 1234 % 1000 - 1234 % 100 ) / 100 --> 2
```

```
106            bucketNumber = ( array[ i ] % divisor - array[ i ] %
107               ( divisor / 10 ) ) / ( divisor / 10 );
108
109            // retrieve value in pail[ bucketNumber ][ 0 ] to
110            // determine which element of the row to store c[ i ]
111            elementNumber = ++pail[ bucketNumber ][ 0 ];
112            pail[ bucketNumber ][ elementNumber ] = array[ i ];
113         }
114
115      }  // end method distributeElements
116
117      // return elements to original array
118      public void collectElements( int array[], int pails[][] )
119      {
120         int subscript = 0;
121
122         for ( int i = 0; i < 10; i++ )
123
124            for ( int j = 1; j <= pails[ i ][ 0 ]; j++ )
125               array[ subscript++ ] = pails[ i ][ j ];
126      }
127
128      // set all buckets to zero
129      public void emptyBucket( int pails[][] )
130      {
131         for ( int i = 0; i < 10; i++ )
132
133            for ( int j = 0; j < SIZE; j++ )
134               pails[ i ][ j ] = 0;
135      }
136
137   }  // end class Bucket
```



7.29 (*Simulation: The Tortoise and the Hare*) In this problem, you will re-create the classic race of the tortoise and the hare. You will use random-number generation to develop a simulation of this memorable event.

Our contenders begin the race at "square 1" of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground.

A clock ticks once per second. With each tick of the clock, your applet should adjust the position of the animals according to the rules in Fig. 7.26. Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (the "starting gate"). If an animal slips left before square 1, move the animal back to square 1.

Generate the percentages in Fig. 7.26 by producing a random integer i in the range $1 \le i \le 10$. For the tortoise, perform a "fast plod" when $1 \le i \le 5$, a "slip" when $6 \le i \le 7$ or a "slow plod" when $8 \le i \le 10$. Use a similar technique to move the hare.

Animal	Move type	Percentage of the time	Actual move
Tortoise	Fast plod	50%	3 squares to the right
	Slip	20%	6 squares to the left
	Slow plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big hop	20%	9 squares to the right
	Big slip	10%	12 squares to the left
	Small hop	30%	1 square to the right
	Small slip	20%	2 squares to the left

Fig. 7.26 Rules for adjusting the positions of the tortoise and the hare.

Begin the race by printing

```
BANG !!!!!
AND THEY'RE OFF !!!!!
```

Then, for each tick of the clock (i.e., each repetition of a loop), print a 70-position line showing the letter `T` in the position of the tortoise and the letter `H` in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your program should print `OUCH!!!` beginning at that position. All print positions other than the `T`, the `H` or the `OUCH!!!` (in case of a tie) should be blank.

After each line is printed, test for whether either animal has reached or passed square 70. If so, print the winner and terminate the simulation. If the tortoise wins, print `TORTOISE WINS!!! YAY!!!` If the hare wins, print `Hare wins. Yuch.` If both animals win on the same tick of the clock, you may want to favor the tortoise (the "underdog"), or you may want to print `It's a tie`. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you are ready to run your program, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets!

Later in the book, we introduce a number of Java capabilities, such as graphics, images, animation, sound and multithreading. As you study those features, you might enjoy enhancing your tortoise-and-hare contest simulation.

ANS:

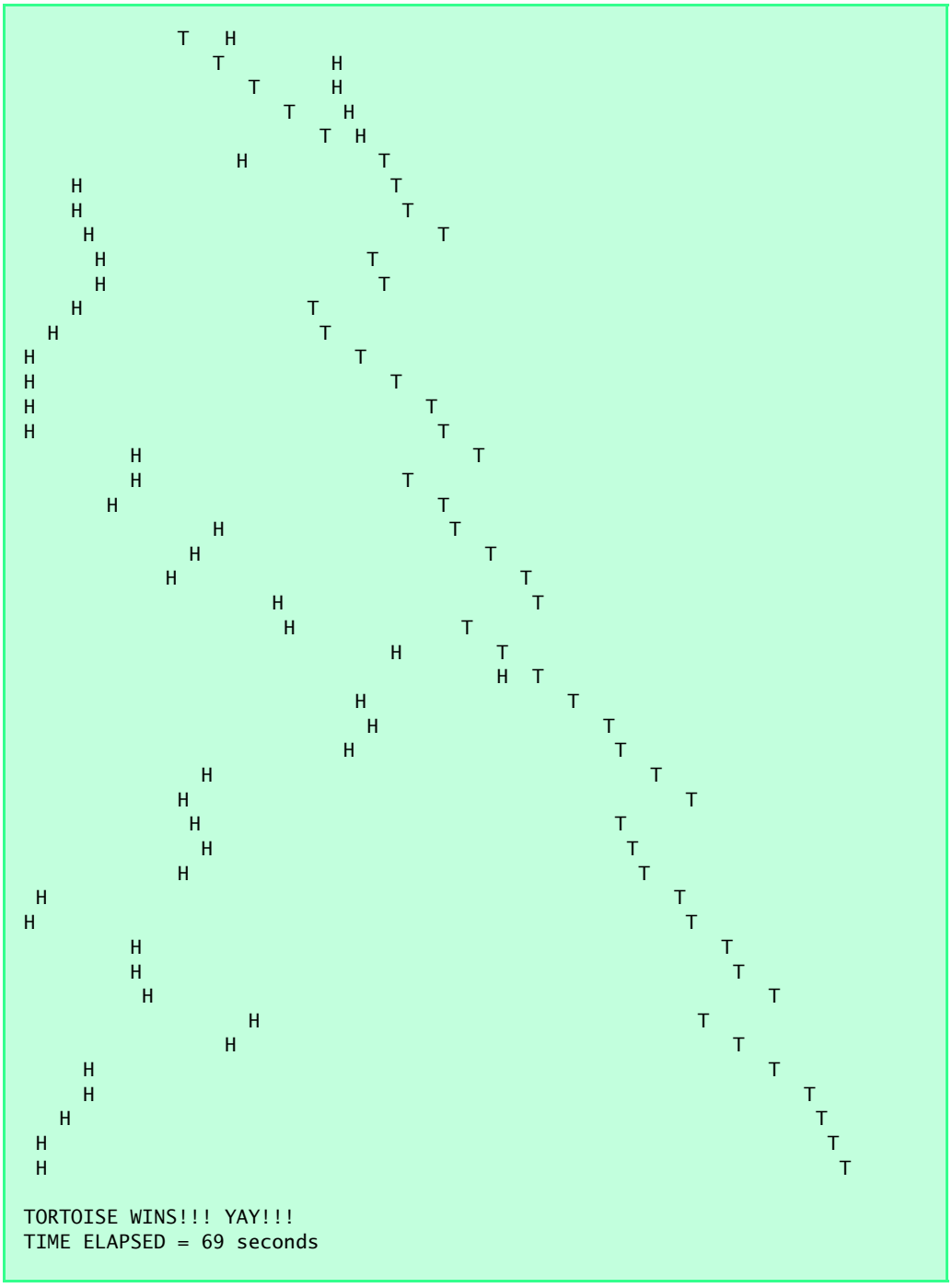
```
1  // Exercise 7.29 Solution: Race.java
2  // Program simulates the race between the tortoise and the hare
3
4  public class Race {
5
6     private static final int RACE_END = 70;  // final position
```

```
 7     private int tortoise = 1, hare = 1; // contestants' position
 8     private int timer;
 9
10     // run the race
11     public Race() {
12
13        timer = 0;
14
15        System.out.println( "ON YOUR MARK, GET SET" );
16        System.out.println( "BANG !!!!!" );
17        System.out.println( "AND THEY'RE OFF !!!!!" );
18
19        while ( tortoise != RACE_END && hare != RACE_END ) {
20           moveHare();
21           moveTortoise();
22           printCurrentPositions();
23
24           // slow down race
25           for ( int temp = 0; temp < 100000000; temp++ ) {
26              ;
27           }
28
29           ++timer;
30        }
31
32        // tortoise beats hare or a tie
33        if ( tortoise >= hare )
34           System.out.println( "\nTORTOISE WINS!!! YAY!!!" );
35
36        // hare beat tortoise
37        else
38           System.out.println( "Hare wins. Yuch!" );
39
40        System.out.println( "TIME ELAPSED = " + timer + " seconds" );
41     }
42
43     // move tortoise's position
44     public void moveTortoise()
45     {
46        // randomize move to choose
47        int percent = ( int ) ( 1 + Math.random() * 10 );
48
49        int tortoiseMoves[] = { 3, 6 }; // possible tortoise moves
50
51        // determine moves by percent in range in Fig 7.32
52        // fast plod
53        if ( percent >= 1 && percent <= 5 )
54           tortoise += tortoiseMoves[ 0 ];
55
56        // slip
57        else if ( percent == 6 || percent == 7 )
58           tortoise -= tortoiseMoves[ 1 ];
59
```

```
60          // slow plod
61          else
62             ++tortoise;
63
64          // ensure tortoise doesn't slip beyond start position
65          if ( tortoise < 1 )
66             tortoise = 1;
67
68          // ensure tortoise doesn't pass the finish
69          else if ( tortoise > RACE_END )
70             tortoise = RACE_END;
71
72       }  // end method move Tortoise
73
74       // move hare's position
75       public void moveHare()
76       {
77          // randomize move to choose
78          int percent = ( int ) ( 1 + Math.random() * 10 );
79
80          int hareMoves[] = { 9, 12, 2 };  // possible hare moves
81
82          // determine moves by percent in range in Fig 7.32
83          // big hop
84          if ( percent == 3 || percent == 4 )
85             hare += 9;
86
87          // big slip
88          else if ( percent == 5 )
89             hare -= 12;
90
91          // small hop
92          else if ( percent >= 6 && percent <= 8 )
93             ++hare;
94
95          // small slip
96          else if ( percent > 8 )
97             hare -= 2;
98
99          // ensure that hare doesn't slip beyond start position
100         if ( hare < 1 )
101            hare = 1;
102
103         // ensure hare doesn't pass the finish
104         else if ( hare > RACE_END )
105            hare = RACE_END;
106
107      }  // end method moveHare
108
109      // display positions of tortoise and hare
110      public void printCurrentPositions()
111      {
112         // goes through all 70 squares, printing H
113         // if hare on position and T for tortoise
```

```
114         for ( int count = 1; count <= RACE_END; count++ )
115
116            // tortoise and hare positions collide
117            if ( count == tortoise && count == hare )
118               System.out.print( "OUCH!!!" );
119
120            else if ( count == hare )
121               System.out.print( 'H' );
122
123            else if ( count == tortoise )
124               System.out.print( 'T' );
125
126            else
127               System.out.print( ' ' );
128
129         System.out.println();
130
131      }  // end printCurrentPositions
132
133      // execute application
134      public static void main( String args[] )
135      {
136         Race race = new Race();
137      }
138
139   }  // end class Race
```

```
ON YOUR MARK, GET SET
BANG !!!!!
AND THEY'RE OFF !!!!!
H  T
H   T
       T H
          OUCH!!!
H          T
 H   T
 H    T
       T  H
H       T
H        T
         HT
       H   T
       H      T
        T       H
           T              H
            T                      H
               T                    H
                  T    H
           H       T
         H            T
        H              T
        H          T
             T   H
                T         H
```



7.30 The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, …

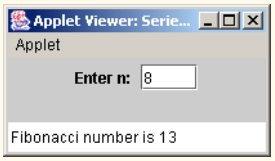
begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms.

a) Write a *nonrecursive* method `fibonacci( n )` that calculates the *n*th Fibonacci number. Incorporate this method into an applet that enables the user to enter the value of `n`.

ANS:

```
1  // Exercise 7.30 Part A Solution: Series.java
2  // Program calculates the Fibonacci series iteratively
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Series extends JApplet implements ActionListener {
8
9     JTextField input;
10    JLabel label;
11
12    // initialize applet
13    public void init()
14    {
15       input = new JTextField( 4 );
16       input.addActionListener( this );
17       label = new JLabel( "Enter n: " );
18
19       Container container = getContentPane();
20       container.setLayout( new FlowLayout() );
21
22       container.add( label );
23       container.add( input );
24    }
25
26    // get nth element input by user
27    public void actionPerformed( ActionEvent event )
28    {
29       int element;
30
31       element = Integer.parseInt( input.getText() );
32
33       if ( element > 0 ) {
34
35          int value = fibonacci( element );
36          showStatus( "Fibonacci number is " + value );
37       }
38
39       else
40          showStatus( "Invalid Value." );
41    }
42
```

```
43     // returns fibonacci number of nth element
44     public int fibonacci( int nElement )
45     {
46        int temp = 1;         // number to be added
47        int fibNumber = 0;    // fibonacci number
48
49        if ( nElement == 1 )
50           return 0;
51
52        // find nth element
53        for ( int n = 2; n <= nElement; n++ ) {
54
55           int last = fibNumber;
56           fibNumber += temp;
57
58           temp = last;
59        }
60
61        return fibNumber;
62     }
63
64  }  // end class Series
```



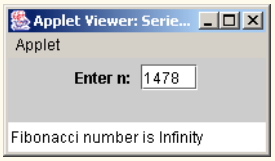
b) Determine the largest Fibonacci number that can be printed on your system.

c) Modify the program you wrote in part (a) to use `double` instead of `int` to calculate and return Fibonacci numbers, and use this modified program to repeat part (b).

ANS:

```
1  // Exercise 7.30 Part C Solution: Series.java
2  // Program calculates the Fibonacci series iteratively
3  // using double as the fibonacci series data type.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Series extends JApplet implements ActionListener {
9
10    JTextField input;
11    JLabel label;
12
13    // initialize applet
14    public void init()
15    {
16       input = new JTextField( 4 );
17       input.addActionListener( this );
18       label = new JLabel( "Enter n: " );
```

```
19
20          Container container = getContentPane();
21          container.setLayout( new FlowLayout() );
22
23          container.add( label );
24          container.add( input );
25       }
26
27       // get nth element input by user
28       public void actionPerformed( ActionEvent event )
29       {
30          double element = Double.parseDouble( input.getText() );
31
32          if ( element > 0.0 ) {
33
34             double value = fibonacci( element );
35             showStatus( "Fibonacci number is " + value );
36          }
37
38          else
39             showStatus( "Invalid Value." );
40       }
41
42       // returns fibonacci number of nth element
43       public double fibonacci( double nElement )
44       {
45          double temp = 1;         // number to be added
46          double fibNumber = 0;    // fibonacci number
47
48          if ( nElement == 1 )
49             return 0;
50
51          // find nth element
52          for ( int n = 2; n <= nElement; n++ ) {
53
54             double last = fibNumber;
55             fibNumber += temp;
56
57             temp = last;
58          }
59
60          return fibNumber;
61       }
62
63    }  // end class Series
```



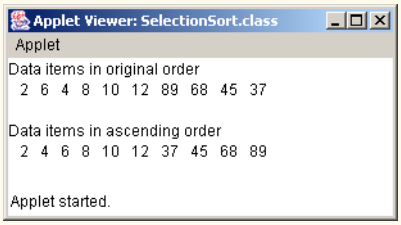
RECURSION EXERCISES

7.31 (*Selection Sort*) A selection sort searches an array for the smallest element in the array, and swaps that element with the first element of the array. The process is repeated for the subarray beginning with the second element. Each pass of the array places one element in its proper location. For an array of n elements, $n - 1$ passes must be made, and for each subarray, $n - 1$ comparisons must be made to find the smallest value. When the subarray being processed contains one element, the array is sorted. Write recursive method `selectionSort` to perform this algorithm.

ANS:

```
1  // Exercise 7.31: SelectionSort.java
2  // Sort an arry's values into ascending order.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class SelectionSort extends JApplet {
7
8     // initialize applet
9     public void init()
10    {
11       JTextArea outputArea = new JTextArea();
12       Container container = getContentPane();
13       container.add( outputArea );
14
15       int array[] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };
16
17       String output = "Data items in original order\n";
18
19       // append original array values to String output
20       for ( int counter = 0; counter < array.length; counter++ )
21          output += "   " + array[ counter ];
22
23       selectionSort( array, 0 ); // sort array
24
25       output += "\n\nData items in ascending order\n";
26
27       // append sorted array values to String output
28       for ( int counter = 0; counter < array.length; counter++ )
29          output += "   " + array[ counter ];
30
31       outputArea.setText( output );
32
33    } // end method init
34
35    // sort elements of array with selectionSort
36    public void selectionSort( int array2[], int start )
37    {
38      if ( start < array2.length ) {
39          int smallest = array2[ start ];
40          int smallestPosition = start;
41
42          // find the smallest element in the array
43          for ( int pass = start + 1; pass < array2.length; pass++ )
44
```

```
            if ( array2[ pass ] < smallest ) {
               smallest = array2[ pass ];
               smallestPosition = pass;
            }

         // swap the smallest with first element in subarray
         swap( array2, start, smallestPosition );

         // recursive calls to selectionSort
         selectionSort( array2, start + 1 );
      }
   }

   // swap two elements of an array
   public void swap( int array3[], int first, int second )
   {
      int hold;  // temporary holding area for swap

      hold = array3[ first ];
      array3[ first ] = array3[ second ];
      array3[ second ] = hold;
   }

} // end class SelectionSort
```



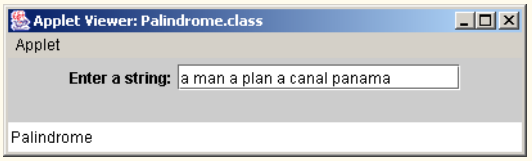
7.32 (*Palindromes*) A palindrome is a string that is spelled the same way forward and backward. Some examples of palindromes are "radar," "able was i ere i saw elba" and (if blanks are ignored) "a man a plan a canal panama." Write a recursive method `testPalindrome` that returns `boolean` value `true` if the string stored in the array is a palindrome and `false` otherwise. The method should ignore spaces and punctuation in the string. [*Hint:* Use `String` method `toCharArray`, which takes no arguments, to get a `char` array containing the characters in the `String`. Then pass the array to method `testPalindrome`.]

ANS:

```
// Exercise 7.32 Solution: Palindrome.java
// Program tests for a palindrome.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Palindrome extends JApplet implements ActionListener {

```

```
 9     JLabel prompt;
10     JTextField input;
11
12     // set up GUI components
13     public void init()
14     {
15        prompt = new JLabel( "Enter a string:" );
16        input = new JTextField( 20 );
17        input.addActionListener( this );
18        Container container = getContentPane();
19        container.setLayout( new FlowLayout() );
20        container.add( prompt );
21        container.add( input);
22
23     }  // end method init
24
25     // obtain user input and call method testPalindrome
26     public void actionPerformed( ActionEvent actionEvent )
27     {
28        String string = actionEvent.getActionCommand();
29        char[] copy = new char[ string.length() + 1 ];
30        int counter = 0;
31
32        for ( int index = 0; index < string.length(); index++ ) {
33
34           // method charAt returns a character at
35           // the specified subscript of a String
36           char character = string.charAt( index );
37
38           copy[ counter++ ] = character;
39        }
40
41        if ( testPalindrome( copy, 0, counter - 1 ) == 1 )
42           showStatus( "Palindrome" );
43
44        else
45           showStatus( "Not a palindrome" );
46
47     }  // end method actionPerformed
48
49     // recursively test if array is palindrome
50     public int testPalindrome( char array[], int left, int right )
51     {
52        while ( array[ left ] == ' ' || array[ left ] == '.' ||
53                array[ left ] == ';' || array[ left ] == ':' ||
54               array[ left ] == '?' || array[ left ] == '!' ||
55                array[ left ] == '-' || array[ left ] == ',' )
56
57           left++;
58
59        while ( array[ right ] == ' ' || array[ right ] == '.' ||
60                array[ right ] == ';' || array[ right ] == ':' ||
61                array[ right ] == '?' || array[ right ] == '!' ||
62                array[ right ] == '-' || array[ right ] == ',' )
```

```
63
64          right--;
65
66       if ( left == right || left > right )
67          return 1;
68
69       else if ( array[ left ] != array[ right ] )
70          return 0;
71
72       else
73          return testPalindrome( array, left + 1, right - 1 );
74
75    }  // end method testPalindrome
76
77 }  // end class Palindrome
```

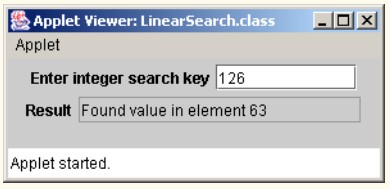


7.33 (*Linear Search*) Modify Fig. 7.11 to use recursive method `linearSearch` to perform a linear search of the array. The method should receive an integer array, the array's length and the search key as arguments. If the search key is found, return its index in the array; otherwise, return –1.

ANS:

```
1  // Exercise 7.33 Solution: LinearSearch.java
2  // Program recursively performs linear search of an array.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class LinearSearch extends JApplet implements ActionListener {
8
9     int array[];
10    JLabel enterLabel, resultLabel;
11    JTextField enterField, resultField;
12
13    // set up GUI components and initialize array
14    public void init()
15    {
16       enterLabel = new JLabel( "Enter integer search key" );
17       enterField = new JTextField( 10 );
18       enterField.addActionListener( this );
19       resultLabel = new JLabel( "Result" );
20       resultField = new JTextField( 20 );
21       resultField.setEditable( false );
22
23       Container container = getContentPane();
24       container.setLayout( new FlowLayout() );
25       container.add( enterLabel );
```

```
26          container.add( enterField );
27          container.add( resultLabel );
28          container.add( resultField );
29
30          array = new int[ 100 ];
31
32         // initialize array with even numbers 0 to 198
33          for ( int counter = 0; counter < array.length; counter++ )
34             array[ counter ] = 2 * counter;
35
36       }  // end method init
37
38       // obtain user input and call method linearSearch
39       public void actionPerformed( ActionEvent actionEvent )
40       {
41          String stringKey = actionEvent.getActionCommand();
42          int intKey = Integer.parseInt( stringKey );
43          int element = linearSearch( array, array.length, intKey );
44
45          if ( element != -1 )
46             resultField.setText( "Found value in element " + element );
47
48          else
49             resultField.setText( "Value not found" );
50
51       }  // end method actionPerformed
52
53       // recursively search for key within parameter array
54       public int linearSearch( int array2[], int size, int intKey )
55       {
56          if ( size == 0 )
57             return -1;
58
59          else if ( array2[ --size ] == intKey )
60             return size;
61
62          else
63             return linearSearch( array2, size, intKey );
64
65       }  // end method linearSearch
66
67    }  // end class LinearSearch
```



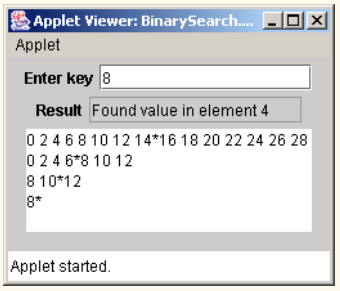
7.34 (*Binary Search*) Modify Fig. 7.12 to use recursive method `binarySearch` to perform a binary search of the array. The method should receive an integer array, the starting index and the ending index as arguments. If the search key is found, return its index in the array; otherwise, return –1.

ANS:

```
1  // Exercise 7.34 Solution: BinarySearch.java
2  // Binary search of an array using recursion
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class BinarySearch extends JApplet implements ActionListener {
8
9     int array[], element;
10    String searchKey, outString;
11
12    int xPosition;  // applet horizontal drawing position
13    int yPosition;  // applet vertical drawing position
14
15    JLabel enterLabel, resultLabel;
16    JTextField result, enter;
17    JTextArea output;
18
19    // initialize applet
20    public void init()
21    {
22       array = new int[ 15 ];
23
24       // create data
25       for ( int i = 0; i < array.length; i++ )
26          array[ i ] = 2 * i;
27
28       // create components
29       enterLabel = new JLabel( "Enter key" );
30       enter = new JTextField( 15 );
31       enter.addActionListener( this );
32
33       resultLabel = new JLabel( "Result" );
34       result = new JTextField( 15 );
35       result.setEditable( false );
36
37       output = new JTextArea( 5, 20 );
38
39       // add components to applet
40       Container container = getContentPane();
41       container.setLayout( new FlowLayout() );
42       container.add( enterLabel );
43       container.add( enter );
44       container.add( resultLabel );
45       container.add( result );
46       container.add( output);
47
48    }  // end method init
49
50    // search number entered by user
51    public void actionPerformed( ActionEvent event )
52    {
```

```
53          outString = "";
54
55          searchKey = event.getActionCommand(); // get user input
56
57          // perform binary search
58          element = binarySearch( Integer.parseInt( searchKey ),
59             array, 0, array.length - 1 );
60
61          // display search result
62          if ( element != -1 )
63             result.setText( "Found value in element " + element );
64
65          else
66             result.setText( "Value not found" );
67
68          // display search process
69          output.setText( outString );
70
71       }  // end method actionPerformed
72
73       // perform binary search on array
74       public int binarySearch( int key, int array[], int low, int high )
75       {
76          // if low subscript is less than high subscript
77          if ( low <= high ) {
78
79             // determine middle element subscript
80             int middle = ( low + high ) / 2;
81
82             printRow( low, middle, high );
83
84             // if key matches middle element, return middle location
85             if ( key == array[ middle ] )
86                return middle;
87
88             // search the left side
89             else if ( key < array[ middle ] )
90                return binarySearch( key, array, low, middle - 1 );
91
92             // search the right side
93             else
94                return binarySearch( key, array, middle + 1, high );
95          }
96
97          return -1; // searchKey not found
98
99       }  // end method binarySearch
100
101      // print one row of output showing the current
102      // part of the array being processed.
103      public void printRow( int low, int mid, int high )
104      {
105         // loop through array elements
106         for ( int i = 0; i < array.length; i++ ) {
```

```
107
108          // if counter outside current array subset
109          if ( i < low || i > high )
110             continue;
111
112          // if middle element, append element to String display
113          // followed by asterisk (*) to indicate middle element
114          else if ( i == mid )   // mark middle value
115             outString += array[ i ] + "*";
116
117          // append element to String display
118          else
119             outString += array[ i ] + " ";
120       }
121
122       outString += "\n";
123    }
124
125 }  // end class BinarySearch
```



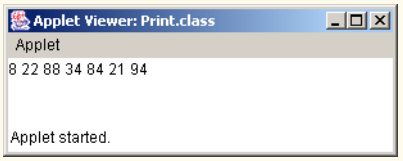
7.35 (*Eight Queens*) Modify the Eight Queens program you created in Exercise 7.24 to solve the problem recursively.

7.36 (*Print an array*) Write a recursive method `printArray` that takes an array of `int` values and the length of the array as arguments and returns nothing. The method should stop processing and return when it receives an array of length zero.

ANS:

```
1  // Exercise 7.36 Solution: Print.java
2  // Program prints a string.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Print extends JApplet {
7     int x;
8     String result;
9     JTextArea output;
10
11    //initializes values
12    public void init()
13    {
```

```
14          output = new JTextArea( 40, 40 );
15          Container container = getContentPane();
16          container.add( output );
17
18          x = 0;
19          result = "";
20          int array[] = { 8, 22, 88, 34, 84, 21, 94 };
21
22          printArray( array, array.length );
23          output.append( result );
24
25       }  // end method init
26
27       // recursively print array
28       public void printArray( int array2[], int size )
29       {
30          if ( size == 0 )
31             return;
32
33          else {
34
35             result += array2[ 0 ] + " ";
36
37             int array3[] = new int[ --size ];
38
39             for ( int counter = 0; counter < size; counter++ )
40                array3[ counter ] = array2[ counter + 1 ];
41
42             printArray( array3, size );
43          }
44
45       }  // end method printArray
46
47    }  // end class Print
```



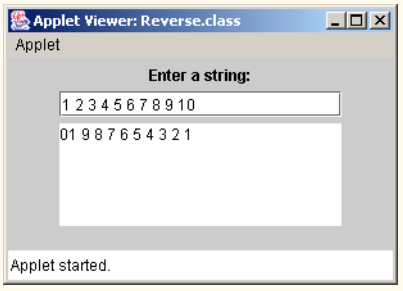
7.37 (*Print an array backward*) Write a recursive method `stringReverse` that takes a character array containing a string as an argument, prints the string backward and returns nothing. [*Hint:* Use `String` method `toCharArray`, which takes no arguments, to get a `char` array containing the characters in the `String`. Then, pass the array to method `stringReverse`.]

ANS:

```
1  // Exercise 7.37 Solution: Reverse.java
2  // Applet prints an input string backwards.
3  import java.awt.*;
4  import java.awt.event.*;
```

```
5  import javax.swing.*;
6
7  public class Reverse extends JApplet implements ActionListener {
8
9     String string; // input string
10
11    char copyChars[], reverseChars[]; // chars of string, reversed chars
12
13    JLabel label;
14    JTextField input;
15    JTextArea output;
16
17    int copied, reversed;
18
19    // initialize applet
20    public void init()
21    {
22       label = new JLabel( "Enter a string:" );
23
24       input = new JTextField( 20 );
25       input.addActionListener( this );
26
27       output = new JTextArea( 5, 20 );
28       output.setEditable( false );
29
30       Container container = getContentPane();
31       container.setLayout( new FlowLayout() );
32       container.add( label );
33       container.add( input);
34       container.add( output );
35    }
36
37    // takes user-input string and prints it reversed
38    public void actionPerformed( ActionEvent event )
39    {
40       copied = 0; // initialize counters for the arrays
41       reversed = 0;
42
43       string = input.getText();
44       copyChars = string.toCharArray(); // get char array from string
45
46       reverseChars = new char[ copyChars.length ];
47
48       stringReverse( copyChars ); // reverse char array
49
50       String outputString = "";
51
52       for ( int count = 0; count < reverseChars.length; count++ )
53          outputString += reverseChars[ count ];
54
55       output.setText( outputString );
56
57    }  // end method actionPerformed
58
```

```
59     // reverse an array of chars
60     public void stringReverse( char charArray[] )
61     {
62        // reached end of string
63        if ( copied >= charArray.length )
64           return;
65
66        else { // else keep on traversing the char array
67
68           copied++;
69           stringReverse( charArray );
70        }
71
72        // start reversing chars when reached end
73        reverseChars[ reversed++ ] = charArray[ --copied ];
74     }
75
76  }  // end class Reverse
```

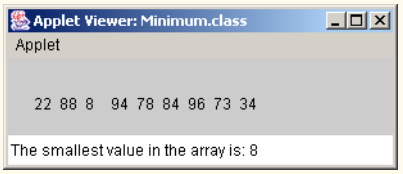


7.38 (*Find the minimum value in an array*) Write a recursive method `recursiveMinimum` that takes an integer array and the array's length as arguments and returns the smallest element of the array. The method should stop processing and return when it receives an array of one element.

ANS:

```
1  // Exercise 7.38 Solution: Minimum.java
2  // Program finds the minimum value in an array.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Minimum extends JApplet {
7     final int MAX = 100;
8     int smallest;
9
10    // initialize instance variable
11    public void init()
12    {
13       smallest = MAX;
14    }
15
```

```
16      // initialize array and draw array values
17      public void paint( Graphics g )
18      {
19         super.paint( g );
20         int array[] = { 22, 88, 8, 94, 78, 84, 96, 73, 34 };
21         int x = 0;
22
23         for ( int counter = 0; counter < array.length; counter++ )
24            g.drawString( "" + array[ counter ], x += 20, 40 );
25
26         showStatus( "The smallest value in the array is: " +
27            recursiveMinimum( array, array.length ) );
28
29      }  // end method paint
30
31      // recursively find minimum value in array
32      public int recursiveMinimum( int array2[], int size )
33      {
34         if ( size > 0 ) {
35
36            if ( array2[ --size ] < smallest )
37               smallest = array2[ size ];
38
39            recursiveMinimum( array2, size );
40         }
41
42         return smallest;
43
44      }  // end method recursiveMinimum
45
46   }  // end class Minimum
```



7.39 (*Quicksort*) The recursive sorting technique called *Quicksort* uses the following basic algorithm for a one-dimensional array of values:

a) *Partitioning Step*: Take the first element of the unsorted array and determine its final location in the sorted array (i.e., all values to the left of the element in the array are less than the element, and all values to the right of the element in the array are greater than the element). We now have one element in its proper location and two unsorted subarrays.

b) *Recursive Step*: Perform step 1 on each unsorted subarray.

Each time step 1 is performed on a subarray, another element is placed in its final location of the sorted array, and two unsorted subarrays are created. When a subarray consists of one element, that element is in its final location (because a one-element array already is sorted).

The basic algorithm seems simple enough, but how do we determine the final position of the first element of each subarray? As an example, consider the following set of values (the element in bold is the partitioning element; it will be placed in its final location in the sorted array):

37 2 6 4 89 8 10 12 68 45

a) Starting from the rightmost element of the array, compare each element with **37** until an element less than **37** is found; then swap **37** and that element. The first element less than **37** is 12, so **37** and 12 are swapped. The new array is

12 2 6 4 89 8 10 ***37*** 68 45

Element 12 is in italics to indicate that it was just swapped with **37**.

b) Starting from the left of the array, but beginning with the element after 12, compare each element with **37** until an element greater than **37** is found; then swap **37** and that element. The first element greater than **37** is 89, so **37** and 89 are swapped. The new array is

12 2 6 4 ***37*** 8 10 *89* 68 45

c) Starting from the right, but beginning with the element before 89, compare each element with **37** until an element less than **37** is found; then swap **37** and that element. The first element less than **37** is 10, so **37** and 10 are swapped. The new array is

12 2 6 4 *10* 8 ***37*** 89 68 45

d) Starting from the left, but beginning with the element after 10, compare each element with **37** until an element greater than **37** is found; then swap **37** and that element. There are no more elements greater than **37**, so when we compare **37** with itself we know that **37** has been placed in its final location of the sorted array.

Once the partition has been applied on the previous array, there are two unsorted subarrays. The subarray with values less than 37 contains 12, 2, 6, 4, 10 and 8. The subarray with values greater than 37 contains 89, 68 and 45. The sort continues with both subarrays being partitioned in the same manner as the original array.

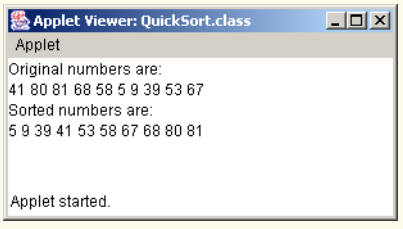
Based on the preceding discussion, write recursive method `quickSort` to sort a one-dimensional integer array. The method should receive as arguments an integer array, a starting index and an ending index. Method `partition` should be called by `quickSort` to perform the partitioning step.

ANS:

```
1  // Exercise 7.39 Solution: QuickSort.java
2  // Applet performs a Quicksort on an array of randomized numbers
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class QuickSort extends JApplet {
7
8     JTextArea output;
9
10    // initialize
11    public void init()
12    {
13       output = new JTextArea( 5, 20 );
14       getContentPane().add( output );
15    }
16
17    // initialize applet
18    public void start( )
19    {
20       int array[] = new int [ 10 ];
21
```

```
22          // randomize original numbers
23          for ( int z = 0; z < array.length; z++ )
24             array[ z ] = ( int ) ( Math.random() * 100 );
25
26          output.append( "Original numbers are:\n" );
27
28          // display original numbers
29          for ( int z = 0; z < array.length; z++ )
30             output.append( array[ z ] + " ");
31
32          quickSort( array, 0, 9 ); // quick sort array of numbers
33          output.append( "\nSorted numbers are:\n" );
34
35          // display sorted numbers
36          for ( int z = 0; z < array.length; z++ )
37             output.append( array[ z ] + " " );
38
39          output.append( "\n\n" );
40       }
41
42       // quick sort an array given first and last positions
43       public void quickSort( int numbers[], int first, int last )
44       {
45          int currentLocation;
46
47          if ( first >= last )
48             return;
49
50          // partitioning phase
51          currentLocation = partition( numbers, first, last );
52
53          // sort the left side
54          quickSort( numbers, first, currentLocation - 1 );
55
56          // sort the right side
57          quickSort( numbers, currentLocation + 1, last );
58       }
59
60       // place an element in its proper position
61       public int partition( int partArray[], int left, int right )
62       {
63          int position = left; // position of target element
64
65          // continue partition until element in correct position
66          while ( true ) {
67
68             // traverse array from right until number greater than target met
69             while ( partArray[ position ] <= partArray[ right ]
70                && position != right )
71
72                right--;
73
74             // element at correct position
75             if ( position == right )
76                return position;
```

```
77
78          // swap first element less than target
79          if ( partArray[ position ] > partArray[ right ] ) {
80
81             swap( partArray, position, right );
82             position = right;
83          }
84
85          // traverse array from left until number greater than target met
86          while ( partArray[ left ] <= partArray[ position] &&
87             left != position )
88
89                ++left;
90
91          // element at correct position
92          if ( position == left )
93             return position;
94
95          // swap first element greater than target
96          if ( partArray[ left ] > partArray[ position ] ) {
97
98             swap( partArray, position, left );
99             position = left;
100         }
101
102      }  // end while
103
104   }  // end method partition
105
106   // swap two elements in an array
107   public void swap( int array[], int position, int rightLeft )
108   {
109      int temp = array[ position ];
110      array[ position ] = array[ rightLeft ];
111      array[ rightLeft ] = temp;
112   }
113
114}  // end class QuickSort
```



7.40 (*Maze Traversal*) The following grid of #s and dots (.) is a two-dimensional array representation of a maze. The #s represent the walls of the maze, and the dots represent squares in the possible paths through the maze. Moves can be made only to a location in the array that contains a dot.

```
# # # # # # # # # # # #
# . . . # . . . . . . #
. . # . # . # # # # . #
# # # . # . . . . # . #
# . . . . # # # . # . .
# # # # . # . # . # . #
# . . # . # . # . # . #
# # . # . # . # . # . #
# . . . . . . . . # . #
# # # # # # . # # # . #
# . . . . . . # . . . #
# # # # # # # # # # # #
```

There is a simple algorithm for walking through a maze that guarantees finding the exit (assuming there is an exit). If there is not an exit, you will arrive at the starting location again. Place your right hand on the wall to your right and begin walking forward. Never remove your hand from the wall. If the maze turns to the right, follow the wall to the right. As long as you do not remove your hand from the wall, eventually you will arrive at the exit of the maze. There may be a shorter path than the one you have taken, but you are guaranteed to get out of the maze if you follow the algorithm.

Write recursive method `mazeTraverse` to walk through this maze. The method should receive as arguments a 12-by-12 character array representing the maze and the starting location of the maze. As `mazeTraverse` attempts to locate the exit, it should place the character `X` in each square in the path. The method should display the maze after each move so the user can watch as the maze is solved.

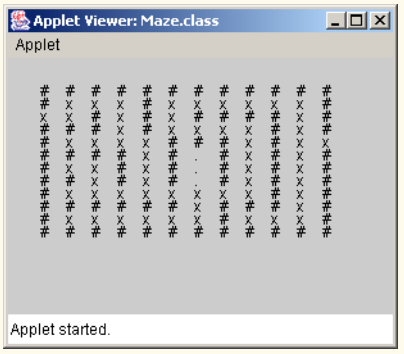
ANS:

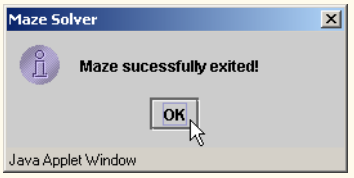
```
1  // Exercise 7.39 Solution: Maze.java
2  // Program traverses a maze.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Maze extends JApplet {
7     final int RIGHT = 1, LEFT = 3, UP = 2, DOWN = 0;
8     final int Y_START = 2, X_START = 0;
9     int move;
10    char maze[][] =
11       { { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' },
12         { '#', '.', '.', '.', '#', '.', '.', '.', '.', '.', '.', '#' },
13         { '.', '.', '#', '.', '#', '.', '#', '#', '#', '#', '.', '#' },
14         { '#', '#', '#', '.', '#', '.', '.', '.', '.', '#', '.', '#' },
15         { '#', '.', '.', '.', '.', '#', '#', '#', '.', '#', '.', '.' },
16         { '#', '#', '#', '#', '.', '#', '.', '#', '.', '#', '.', '#' },
17         { '#', '.', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#' },
18         { '#', '#', '.', '#', '.', '#', '.', '#', '.', '#', '.', '#' },
19         { '#', '.', '.', '.', '.', '.', '.', '.', '.', '#', '.', '#' },
20         { '#', '#', '#', '#', '#', '#', '.', '#', '#', '#', '.', '#' },
21         { '#', '.', '.', '.', '.', '.', '.', '#', '.', '.', '.', '#' },
22         { '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#' } };
23
24    // initialize applet
25    public void init()
26    {
27       move = 0;
28    }
```

```
29
30    // traverse maze recursively
31    public void mazeTraversal( char maze2[][], int y, int x,
32       int direction )
33    {
34       maze2[ y ][ x ] = 'x'; // place marker in maze
35       move++;
36       repaint();
37
38      // if returned to starting location
39       if ( y == Y_START && x == X_START && move > 1 ) {
40          JOptionPane.showMessageDialog( null,
41             "Returned to starting location!", "Maze Solver",
42             JOptionPane.INFORMATION_MESSAGE );
43          return;
44       }
45
46       // if maze exited
47       else if ( mazeExited( y, x ) && move > 1 ) {
48          JOptionPane.showMessageDialog( null, "Maze sucCessfully exited!",
49             "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
50       }
51
52       // make next move
53       else {
54          JOptionPane.showMessageDialog( null, "Next move?",
55             "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
56
57          // determine where next move should be made
58          for ( int move = direction, count = 0; count < 4;
59             ++count, ++move, move %= 4 )
60
61             // checks to see if the space to the right is free. If  so,
62             // moves there and continues maze traversal. If not,  breaks
63             // out of switch and tries new value of move in for loop.
64             switch ( move ) {
65
66                case DOWN:
67
68                   if ( validMove( y + 1, x ) ) {   // move down
69                      mazeTraversal( maze2, y + 1, x, LEFT );
70                      return;
71                   }
72
73                   break;
74
75                case RIGHT:
76
77                   if ( validMove( y, x + 1 ) ) {   // move right
78                      mazeTraversal( maze2, y, x + 1, DOWN );
79                      return;
80                   }
81
82                   break;
```

```
83
84                case UP:
85
86                   if ( validMove( y - 1, x ) ) {   // move up
87                      mazeTraversal( maze2, y - 1, x, RIGHT );
88                      return;
89                   }
90
91                   break;
92
93                case LEFT:
94
95                   if ( validMove( y, x - 1 ) ) {  // move left
96                      mazeTraversal( maze2, y, x - 1, UP );
97                      return;
98                   }
99
100             } // end switch statement
101
102          // if no valid moves available
103          JOptionPane.showMessageDialog( null, "Can't move!" ,
104             "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
105
106       }  // end else
107
108    }  // end method mazeTraversal
109
110    // check if move is valid
111    public boolean validMove( int row, int column )
112    {
113       return ( row >= 0 && row <= 11 && column >= 0 &&
114          column <= 11 && maze[ row ][ column ] != '#' );
115    }
116
117    // check if location is on edge of maze
118    public boolean mazeExited( int row, int column )
119    {
120       return ( row == 0 || row == 11 ||
121          column == 0 || column == 11 );
122    }
123
124    // draw maze
125    public void paint( Graphics g )
126    {
127       super.paint( g );
128       int x = 5, y = 30;
129
130       for ( int row = 0; row < maze.length; row++ ) {
131
132          for ( int column = 0; column < maze[ row ].length; column++ )
133             g.drawString( "" + maze[ row ][ column ], x += 20, y );
134
135          y += 10;
136          x = 5;
137       }
```

```
138
139       if ( move == 0 )
140          mazeTraversal( maze, Y_START, X_START, RIGHT );
141
142    }  // end method paint
143
144 }  // end class Maze
```





7.41 (*Generating Mazes Randomly*) Write a method `mazeGenerator` that takes as an argument a two-dimensional 12-by-12 character array and randomly produces a maze. The method should also provide the starting and ending locations of the maze. Try your method `mazeTraverse` from Exercise 7.40, using several randomly generated mazes.

ANS:

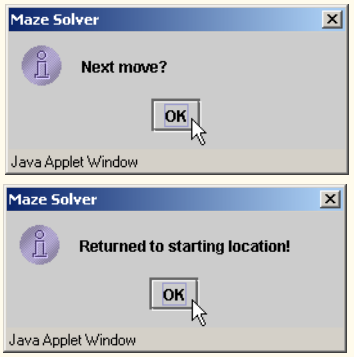
```
1  // Exercise 7.41 Solution: Maze.java
2  // Program randomly generates mazes, then traverses them
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Maze extends JApplet {
7     final int RIGHT = 1, LEFT = 3, UP = 2, DOWN = 0, MAX_DOTS = 100;
8     char maze[][];
9     int move = 0, flag, xStart = 0, yStart = 0;
10
11    // initialize values
12    public void init()
13    {
14       maze = new char[ 12 ][ 12 ];
15
16       // create block of '#' characters
17       for ( int j = 0; j < maze.length; j++ )
18          for ( int k = 0; k < maze[ j ].length; k++ )
19             maze[ j ][ k ] = '#';
20
21       mazeGenerator( maze );
22    }
23
```

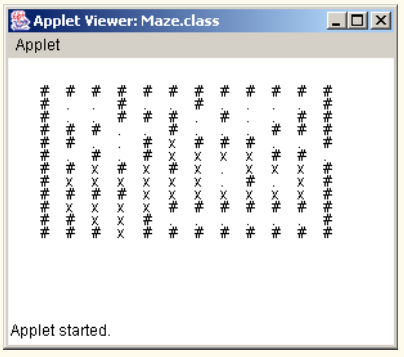
```
24    // display maze
25    public void paint( Graphics g )
26    {
27       int x = 5, y = 30;
28
29       for ( int row = 0; row < maze.length; row++ ) {
30
31          for ( int column = 0; column < maze[ row ].length; column++ )
32             g.drawString( "" + maze[ row ][ column ], x += 20, y );
33
34          y += 10;
35          x = 5;
36       }
37
38       if ( move == 0 )
39          mazeTraversal( maze, yStart, xStart, RIGHT );
40    }
41
42    // traverse maze recursively
43    public void mazeTraversal( char maze2[][], int y, int x,
44       int direction )
45    {
46       maze2[ y ][ x ] = 'x'; // place marker in maze
47       move++;
48       repaint();
49
50      // if returned to starting location
51       if ( y == yStart && x == xStart && move > 1 ) {
52          JOptionPane.showMessageDialog( null,
53             "Returned to starting location!", "Maze Solver",
54             JOptionPane.INFORMATION_MESSAGE );
55          return;
56       }
57
58       // if maze exited
59       else if ( mazeExited( y, x ) && move > 1 ) {
60          JOptionPane.showMessageDialog( null, "Maze successfully exited!",
61             "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
62       }
63
64       // make next move
65       else {
66          JOptionPane.showMessageDialog( null, "Next move?",
67             "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
68
69          // determine where next move should be made
70          for ( int move = direction, count = 0; count < 4;
71             ++count, ++move, move %= 4 )
72
73             // checks to see if the space to the right is free. If so,
74             // moves there and continues maze traversal. If not, breaks
75             // out of switch and tries new value of move in for loop.
76             switch ( move ) {
77
```

```
78                case DOWN:
79
80                   if ( validMove( y + 1, x ) ) {   // move down
81                      mazeTraversal( maze2, y + 1, x, LEFT );
82                      return;
83                   }
84
85                   break;
86
87                case RIGHT:
88
89                   if ( validMove( y, x + 1 ) ) {   // move right
90                      mazeTraversal( maze2, y, x + 1, DOWN );
91                      return;
92                   }
93
94                   break;
95
96                case UP:
97
98                   if ( validMove( y - 1, x ) ) {   // move up
99                      mazeTraversal( maze2, y - 1, x, RIGHT );
100                     return;
101                  }
102
103                  break;
104
105               case LEFT:
106
107                  if ( validMove( y, x - 1 ) ) {  // move left
108                     mazeTraversal( maze2, y, x - 1, UP );
109                     return;
110                  }
111
112            } // end switch statement
113
114         // if no valid moves available
115         JOptionPane.showMessageDialog( null, "Can't move!" ,
116            "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
117
118      }  // end else
119
120   }  // end method mazeTraversal
121
122   // check if move is valid
123   public boolean validMove( int row, int column )
124   {
125      return ( row >= 0 && row <= 11 && column >= 0 &&
126         column <= 11 && maze[ row ][ column ] != '#' );
127   }
128
129   // check if location is on edge of maze
130   public boolean mazeExited( int row, int column )
131   {
```

```
132       return ( row == 0 || row == 11 || column == 0 || column == 11 );
133    }
134
135    // create a new maze
136    public void mazeGenerator( char board[][] )
137    {
138       int exitValue, xValue, yValue, entry, exit;
139
140       xStart = 0; // create start location
141       yStart = 0;
142
143       do {
144          entry = ( int ) ( 1 + Math.random() * 4 );
145          exit = ( int ) ( 1 + Math.random() * 4 );
146       } while ( entry == exit );
147
148       // determine entry position
149       if ( entry == 0 ) {
150
151          xStart = ( int ) ( 1 + Math.random() * 10 ); // avoid corners
152          yStart = 0;
153       }
154
155       else if ( entry == 1 ) {
156          xStart = 0;
157          yStart = ( int ) ( 1 + Math.random() * 10 );
158       }
159
160       else if ( entry == 2 ) {
161          xStart = ( int ) ( 1 + Math.random() * 10 );
162          yStart = 11;
163       }
164
165       else {
166          xStart = 11;
167          yStart = ( int ) ( 1 + Math.random() * 10 );
168       }
169
170       board[ yStart ][ xStart ] = '.';
171
172
173       // determine exit location
174       if ( exit == 0 ) {
175          exitValue = ( int ) ( 1 + Math.random() * 10 );
176          board[ 0 ][ exitValue ] = '.';
177       }
178
179       else if ( exit == 1 ) {
180          exitValue = ( int ) ( 1 + Math.random() * 10 );
181          board[ exitValue ][ 0 ] = '.';
182       }
183
184       else if ( exit == 2 ) {
185          exitValue = ( int ) ( 1 + Math.random() * 10 );
```

```
186            board[ 11 ][ exitValue ] = '.';
187         }
188
189         else {
190            exitValue = ( int ) ( 1 + Math.random() * 10 );
191            board[ exitValue ][ 11 ] = '.';
192         }
193
194         // create paths by adding dots randomly
195         // Note: maze doesn't necessarily have a solution
196         for ( int loop = 1; loop < MAX_DOTS; loop++ ) {
197            xValue = ( int ) ( 1 + Math.random() * 10 );
198            yValue = ( int ) ( 1 + Math.random() * 10 );
199            board[ yValue ][ xValue ] = '.';
200         }
201
202      }  // end method MazeGenerator
203
204   }  // end class Maze
```





7.42 (*Mazes of Any Size*) Generalize methods `mazeTraverse` and `mazeGenerator` of Exercise 7.40 and Exercise 7.41 to process mazes of any width and height.

ANS:

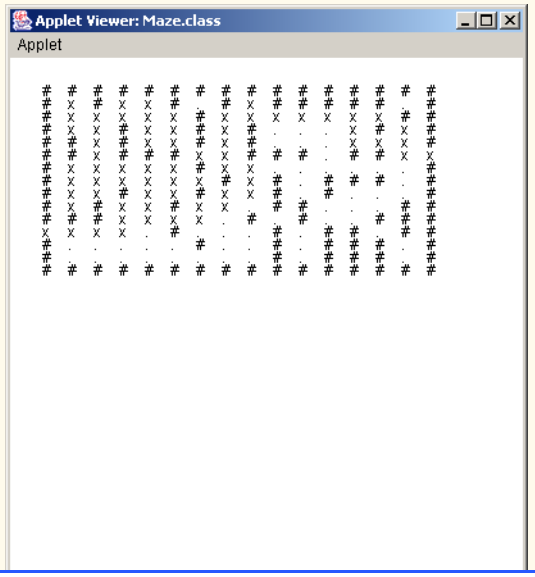
```
1  // Exercise 7.41 Solution: Maze.java
2  // Program randomly generates mazes of different sizes and traverses them
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Maze extends JApplet {
7     final int RIGHT = 1, LEFT = 3, UP = 2, DOWN = 0,
8        MAX_ROWS = 30, MAX_COLS = 30, MIN_ROWS = 5, MIN_COLS = 5;
9     char maze[][];
10    int move = 0, flag, xStart = 0, yStart = 0, rows, cols, numDots;
11
12    // initialize values
13    public void init()
14    {
```

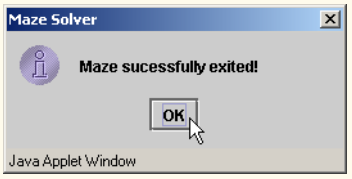
```
15          rows = ( int ) ( MIN_ROWS + Math.random() *
16             ( MAX_ROWS - MIN_ROWS ) );
17          cols = ( int ) ( MIN_COLS + Math.random() *
18             ( MAX_COLS - MIN_COLS ) );
19          maze = new char[ rows ][ cols ];
20          numDots = ( rows * cols ) / 4 * 3;
21
22          for ( int j = 0; j < maze.length; j++ )
23             for ( int k = 0; k < maze[ j ].length; k++ )
24                maze[ j ][ k ] = '#';
25
26          mazeGenerator( maze );
27       }
28
29       // display maze
30       public void paint( Graphics g )
31       {
32          int x = 5, y = 30;
33
34          for ( int row = 0; row < maze.length; row++ ) {
35
36             for ( int column = 0; column < maze[ row ].length; column++ )
37                g.drawString( "" + maze[ row ][ column ], x += 20, y );
38
39             y += 10;
40             x = 5;
41          }
42
43          if ( move == 0 )
44             mazeTraversal( maze, yStart, xStart, RIGHT );
45       }
46
47       // traverse maze recursively
48       public void mazeTraversal( char maze2[][], int y, int x,
49          int direction )
50       {
51          maze2[ y ][ x ] = 'x'; // place marker in maze
52          move++;
53          repaint();
54
55         // if returned to starting location
56          if ( y == yStart && x == xStart && move > 1 ) {
57             JOptionPane.showMessageDialog( null,
58                "Returned to starting location!", "Maze Solver",
59                JOptionPane.INFORMATION_MESSAGE );
60             return;
61          }
62
63          // if maze exited
64          else if ( mazeExited( y, x ) && move > 1 ) {
65             JOptionPane.showMessageDialog( null, "Maze successfully exited!",
66                "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
67          }
68
```

```
69          // make next move
70          else {
71             JOptionPane.showMessageDialog( null, "Next move?",
72                "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
73
74             // determine where next move should be made
75             for ( int move = direction, count = 0; count < 4;
76                ++count, ++move, move %= 4 )
77
78                // checks to see if the space to the right is free.  If so,
79                // moves there and continues maze traversal. If not, breaks
80                // out of switch and tries new value of move in for loop.
81                switch ( move ) {
82
83                   case DOWN:
84
85                      if ( validMove( y + 1, x ) ) {
86                         mazeTraversal( maze2, y + 1, x, LEFT ); // move down
87                         return;
88                      }
89
90                      break;
91
92                   case RIGHT:
93
94                      if ( validMove( y, x + 1 ) ) {
95                         mazeTraversal( maze2, y, x + 1, DOWN ); // move right
96                         return;
97                      }
98
99                      break;
100
101                  case UP:
102
103                     if ( validMove( y - 1, x ) ) {
104                        mazeTraversal( maze2, y - 1, x, RIGHT ); // move up
105                        return;
106                     }
107
108                     break;
109
110                  case LEFT:
111
112                     if ( validMove( y, x - 1 ) ) {
113                        mazeTraversal( maze2, y, x - 1, UP ); // move left
114                        return;
115                     }
116
117               } // end switch statement
118
119            // if no valid moves available
120            JOptionPane.showMessageDialog( null, "Can't move!" ,
121               "Maze Solver", JOptionPane.INFORMATION_MESSAGE );
122
```

```
123         }  // end else
124
125      }  // end method mazeTraversal
126
127      // check if move is valid
128      public boolean validMove( int row, int column )
129      {
130         return ( row >= 0 && row < rows && column >= 0 &&
131            column < cols && maze[ row ][ column ] != '#' );
132      }
133
134      // check if location is on edge of maze
135      public boolean mazeExited( int row, int column )
136      {
137         return ( row == 0 || row == rows - 1 ||
138            column == 0 || column == cols - 1 );
139      }
140
141      // create a new maze
142      public void mazeGenerator( char board[][] )
143      {
144         int exitX, exitY, entry, exit;
145
146         do {
147            entry = ( int ) ( 1 + Math.random() * 4 );
148            exit = ( int ) ( 1 + Math.random() * 4 );
149         } while ( entry == exit );
150
151         // determine entry position
152         if ( entry == 0 ) {
153
154            // avoid corners
155            xStart = ( int ) ( 1 + Math.random() * ( cols - 2 ) );
156            yStart = 0;
157         }
158
159         else if ( entry == 1 ) {
160            xStart = 0;
161            yStart = ( int ) ( 1 + Math.random() * ( rows - 2 ) );
162         }
163
164         else if ( entry == 2 ) {
165            xStart = ( int ) ( 1 + Math.random() * ( cols - 2 ) );
166            yStart = rows - 1;
167         }
168
169         else {
170            xStart = cols - 1;
171            yStart = ( int ) ( 1 + Math.random() * ( rows - 2 ) );
172         }
173
174         board[ yStart ][ xStart ] = '.';
175
176
```

```
177         // determine exit location
178         if ( exit == 0 ) {
179            exitX = ( int ) ( 1 + Math.random() * ( cols - 2 ) );
180            exitY = 0;
181         }
182
183         else if ( exit == 1 ) {
184            exitX = 0;
185            exitY = ( int ) ( 1 + Math.random() * ( rows - 2 ) );
186         }
187
188         else if ( exit == 2 ) {
189            exitX = ( int ) ( 1 + Math.random() * ( cols - 2 ) );
190            exitY = rows - 1;
191         }
192
193         else {
194            exitX = cols - 1;
195            exitY = ( int ) ( 1 + Math.random() * ( rows - 2 ) );
196         }
197
198         board[ exitY ][ exitX ] = '.';
199
200         // add dots randomly
201         for ( int loop = 1; loop < numDots; loop++ ) {
202            int xValue = ( int ) ( 1 + Math.random() * ( cols - 2 ) );
203            int yValue = ( int ) ( 1 + Math.random() * ( rows - 2 ) );
204            board[ yValue ][ xValue ] = '.';
205         }
206
207      }  // end method MazeGenerator
208
209   }  // end class Maze
```





SPECIAL SECTION: BUILDING YOUR OWN COMPUTER

In the next several problems, we take a temporary diversion from the world of high-level language programming. To "peel open" a computer and look at its internal structure. We introduce machine-language programming and write several machine-language programs. To make this an especially valuable experience, we then build a computer (through the technique of software-based *simulation*) on which you can execute your machine-language programs!

7.43 (*Machine-Language Programming*) Let us create a computer called the Simpletron. As its name implies, it is a simple, but powerful, machine. The Simpletron runs programs written in the only language it directly understands: Simpletron Machine Language, or SML for short.

The Simpletron contains an *accumulator*—a "special register" in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All information in the Simpletron is handled in terms of *words*. A word is a signed four-digit decimal number such as `+3364`, `-1293`, `+0007` and `-0001`. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers `00`, `01`, ..., `99`.

Before running an SML program, we must *load*, or place, the program into memory. The first instruction (or statement) of every SML program is always placed in location `00`. The simulator will start executing at this location.

Each instruction written in SML occupies one word of the Simpletron's memory (and hence instructions are signed four-digit decimal numbers). We shall assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron's memory may contain an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code* specifying the operation to be performed. SML operation codes are summarized in Fig. 7.27.

The last two digits of an SML instruction are the *operand*—the address of the memory location containing the word to which the operation applies. Let's consider several simple SML programs.

Operation code	Meaning
Input/output operations:	
`final int READ = 10;`	Read a word from the keyboard into a specific location in memory.
`final int WRITE = 11;`	Write a word from a specific location in memory to the screen.
Load/store operations:	
`final int LOAD = 20;`	Load a word from a specific location in memory into the accumulator.
`final int STORE = 21;`	Store a word from the accumulator into a specific location in memory.
Arithmetic operations:	
`final int ADD = 30;`	Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).

Fig. 7.27 Simpletron Machine Language (SML) operation codes. (Part 1 of 2.)

Operation code	Meaning
`final int SUBTRACT = 31;`	Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator).
`final int DIVIDE = 32;`	Divide a word from a specific location in memory into the word in the accumulator (leave result in the accumulator).
`final int MULTIPLY = 33;`	Multiply a word from a specific location in memory by the word in the accumulator (leave the result in the accumulator).
Transfer of control operations:	
`final int BRANCH = 40;`	Branch to a specific location in memory.
`final int BRANCHNEG = 41;`	Branch to a specific location in memory if the accumulator is negative.
`final int BRANCHZERO = 42;`	Branch to a specific location in memory if the accumulator is zero.
`final int HALT = 43;`	Halt. The program has completed its task.

Fig. 7.27 Simpletron Machine Language (SML) operation codes. (Part 2 of 2.)

The first SML program (Fig. 7.28) reads two numbers from the keyboard and computes and prints their sum. The instruction +1007 reads the first number from the keyboard and places it into location 07 (which has been initialized to 0). Then instruction +1008 reads the next number into location 08. The *load* instruction, +2007, puts the first number into the accumulator, and the *add* instruction, +3008, adds the second number to the number in the accumulator. *All SML arithmetic instructions leave their results in the accumulator.* The *store* instruction, +2109, places the result back into memory location 09, from which the *write* instruction, +1109, takes the number and prints it (as a signed four-digit decimal number). The *halt* instruction, +4300, terminates execution.

Location	Number	Instruction
00	+1007	(Read A)
01	+1008	(Read B)
02	+2007	(Load A)
03	+3008	(Add B)
04	+2109	(Store C)
05	+1109	(Write C)
06	+4300	(Halt)
07	+0000	(Variable A)
08	+0000	(Variable B)
09	+0000	(Result C)

Fig. 7.28 SML program that reads two integers and computes their sum.

The second SML program (Fig. 7.29) reads two numbers from the keyboard and determines and prints the larger value. Note the use of the instruction +4107 as a conditional transfer of control, much the same as Java's `if` statement.

Location	Number	Instruction
00	+1009	(Read A)
01	+1010	(Read B)
02	+2009	(Load A)
03	+3110	(Subtract B)
04	+4107	(Branch negative to 07)
05	+1109	(Write A)
06	+4300	(Halt)
07	+1110	(Write B)
08	+4300	(Halt)
09	+0000	(Variable A)
10	+0000	(Variable B)

Fig. 7.29 SML program that reads two integers and determines which is larger.

Now write SML programs to accomplish each of the following tasks:

a) Use a sentinel-controlled loop to read 10 positive numbers. Compute and print their sum.
b) Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and print their average.
c) Read a series of numbers, and determine and print the largest number. The first number read indicates how many numbers should be processed.

7.44 (*A Computer Simulator*) In this problem, you are going to build your own computer. No, you will not be soldering components together. Rather, you will use the powerful technique of *software-based simulation* to create an object-oriented *software model* of the Simpletron of Exercise 7.43. Your Simpletron simulator will turn the computer you are using into a Simpletron, and you will actually be able to run, test and debug the SML programs you wrote in Exercise 7.43. Your Simpletron will be an event-driven applet: You will click a button to execute each SML instruction, and you will be able to see the instruction "in action."

When you run your Simpletron simulator, it should begin by displaying:

```
*** Welcome to Simpletron! ***
*** Please enter your program one instruction  ***
*** (or data word) at a time into the input    ***
*** text field. I will display the location    ***
*** number and a question mark (?). You then   ***
*** type the word for that location. Press the ***
*** Done button to stop entering your program. ***
```

The program should display an `input JTextField` in which the user will type each instruction one at a time and a `Done` button for the user to click when the complete SML program has been entered. Simulate the memory of the Simpletron with a one-dimensional array `memory` that has 100 elements.

Now assume that the simulator is running, and let us examine the dialog as we enter the program of Fig. 7.29 (Exercise 7.43):

```
00 ? +1009
01 ? +1010
02 ? +2009
03 ? +3110
04 ? +4107
05 ? +1109
06 ? +4300
07 ? +1110
08 ? +4300
09 ? +0000
10 ? +0000
```

Your program should use a `JTextField` to display the memory location followed by a question mark. Each of the values to the right of a question mark is typed by the user into the `input JTextField`. When the `Done` button is clicked, the program should display the following:

```
*** Program loading completed ***
*** Program execution begins  ***
```

The SML program has now been placed (or loaded) in array `memory`. The Simpletron should provide an "`Execute next instruction`" button the user can click to execute each instruction in the SML program. Execution begins with the instruction in location `00` and, as in Java, continues sequentially, unless directed to some other part of the program by a transfer of control.

Use the variable `accumulator` to represent the accumulator register. Use the variable `instructionCounter` to keep track of the location in memory that contains the instruction being performed. Use the variable `operationCode` to indicate the operation currently being performed (i.e., the left two digits of the instruction word). Use the variable `operand` to indicate the memory location on which the current instruction operates. Thus, `operand` is the rightmost two digits of the instruction currently being performed. Do not execute instructions directly from memory. Rather, transfer the next instruction to be performed from memory to a variable called `instructionRegister`. Then "pick off" the left two digits and place them in `operationCode`, and "pick off" the right two digits and place them in `operand`. Each of the preceding registers should have a corresponding `JTextField` in which its current value is displayed at all times. When the Simpletron begins execution, the special registers are all initialized to zero.

Now, let us "walk through" execution of the first SML instruction, `+1009` in memory location `00`. This procedure is called an *instruction execution cycle.*

The `instructionCounter` tells us the location of the next instruction to be performed. We *fetch* the contents of that location from `memory` by using the Java statement

```
instructionRegister = memory[ instructionCounter ];
```

The operation code and the operand are extracted from the instruction register by the statements

```
operationCode = instructionRegister / 100;
operand = instructionRegister % 100;
```

Now the Simpletron must determine that the operation code is actually a *read* (versus a *write*, a *load*, etc.). A `switch` differentiates among the 12 operations of SML. In the `switch` statement, the behavior of various SML instructions is simulated as shown in Fig. 7.30. We discuss branch instructions shortly and leave the others to you.

Instruction	Description
read:	Display an input dialog with the prompt "`Enter an integer`." Convert the input value to an integer and store it in location `memory[ operand ]`.
load:	`accumulator = memory[ operand ];`
add:	`accumulator += memory[ operand ];`

Fig. 7.30 Behavior of several SML instructions in the Simpletron.

When the SML program completes execution, the name and contents of each register as well as the complete contents of memory should be displayed. Such a printout is often called a *computer dump* (no, a computer dump is not a place where old computers go). To help you program your dump method, a sample dump format is shown in Fig. 7.31. Note that a dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated. The sample dump assumes that the output will be sent to the display screen with a series of `System.out.print` and `System.out.println` method calls. However, we encourage you to experiment with a version that can be displayed on the applet using a `JTextArea` or an array of `JTextField` objects.

```
REGISTERS:
accumulator          +0000
instructionCounter      00
instructionRegister  +0000
operationCode           00
operand                 00

MEMORY:
       0     1     2     3     4     5     6     7     8     9
 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000
```

Fig. 7.31 A sample dump.

Let us proceed with the execution of our program's first instruction—namely, the +1009 in location 00. As we have indicated, the `switch` statement simulates this task by prompting the user to enter a value into the input dialog, reading the value, converting the value to an integer and storing it in memory location `memory[ operand ]`. Since your Simpletron is event driven, it waits for the user to type a value into the `input JTextField` and press the *Enter* key. The value is then read into location 09.

At this point, simulation of the first instruction is completed. All that remains is to prepare the Simpletron to execute the next instruction. Since the instruction just performed was not a transfer of control, we need merely increment the instruction-counter register as follows:

```
++instructionCounter;
```

This action completes the simulated execution of the first instruction. When the user clicks the `Execute next instruction` button, the entire process (i.e., the instruction execution cycle) begins again with the fetch of the next instruction to be executed.

Now let us consider how the branching instructions—the transfers of control—are simulated. All we need to do is adjust the value in the instruction counter appropriately. Therefore, the unconditional branch instruction (40) is simulated within the `switch` as

```
instructionCounter = operand;
```

The conditional "branch if accumulator is zero" instruction is simulated as

```
if ( accumulator == 0 )
   instructionCounter = operand;
```

At this point, you should implement your Simpletron simulator and run each of the SML programs you wrote in Exercise 7.43. If you desire, you may embellish SML with additional features and provide for these features in your simulator.

Your simulator should check for various types of errors. During the program-loading phase, for example, each number the user types into the Simpletron's `memory` must be in the range `-9999` to `+9999`. Your simulator should test that each number entered is in this range and, if not, keep prompting the user to reenter the number until the user enters a correct number.

During the execution phase, your simulator should check for various serious errors, such as attempts to divide by zero, attempts to execute invalid operation codes, and accumulator overflows (i.e., arithmetic operations resulting in values larger than `+9999` or smaller than `-9999`). Such serious errors are called *fatal errors.* When a fatal error is detected, your simulator should print an error message such as

```
*** Attempt to divide by zero ***
*** Simpletron execution abnormally terminated ***
```

and should print a full computer dump in the format we discussed previously. This treatment will help the user locate the error in the program.

ANS:

```
1  // Exercise 7.43 Solution: Simulator.java
2  // A computer simulator
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Simulator extends JApplet implements ActionListener {
8     static final int READ = 10, WRITE = 11, LOAD = 20, STORE = 21,
9        ADD = 30, SUBTRACT = 31, MULTIPLY = 32, DIVIDE = 33, BRANCH = 40,
10       BRANCH_NEG = 41, BRANCH_ZERO = 42, HALT = 43;
11
12    JTextField input, prompt, accum, counter, operandField,
13       operCode, register;
```

```
14    JLabel accumLabel, counterLabel, operandLabel,
15       operCodeLabel, registerLabel;
16    JButton doneButton, executeButton;
17    boolean readFlag;
18    int memory[], accumulator, instructionCounter, index,
19       operand, operationCode, instructionRegister;
20
21    public void init()
22    {
23       initializeRegisters();
24       instructions();
25
26       // create GUI components
27       doneButton = new JButton( "Done" );
28       doneButton.addActionListener( this );
29       executeButton = new JButton( "Execute next instruction" );
30       executeButton.setEnabled( false );
31       executeButton.addActionListener( this );
32       input = new JTextField( 5 );
33       input.addActionListener( this );
34       prompt = new JTextField( 15 );
35
36       register = new JTextField( 4 );
37       counter = new JTextField( 2 );
38       operCode = new JTextField( 2 );
39       operandField = new JTextField( 2 );
40       accum = new JTextField( 4 );
41
42       prompt.setEditable( false );
43       prompt.setText( "0" + index + " ? " );
44       register.setEditable( false );
45       counter.setEditable( false );
46       operCode.setEditable( false );
47       operandField.setEditable( false );
48
49       accumLabel = new JLabel( "Accumulator:" );
50       counterLabel = new JLabel( "InstructionCounter:" );
51       registerLabel = new JLabel( "InstructionRegister:" );
52       operCodeLabel = new JLabel( "OperationCode:" );
53       operandLabel = new JLabel( "Operand:" );
54
55       // add components to applet
56       Container container = getContentPane();
57       container.setLayout( new FlowLayout() );
58
59       container.add( accumLabel );
60       container.add( accum );
61       container.add( counterLabel );
62       container.add( counter );
63       container.add( registerLabel );
64       container.add( register );
65       container.add( operCodeLabel );
66       container.add( operCode );
67       container.add( operandLabel );
```

```
68          container.add( operandField );
69          container.add( prompt );
70          container.add( input );
71          container.add( doneButton );
72          container.add( executeButton );
73
74          displayRegisters(); // place correct values in fields
75       }
76
77       // set values held by text fields
78       public void displayRegisters()
79       {
80          operandField.setText( "" + operand );
81          counter.setText( "" + instructionCounter );
82          accum.setText( "" + accumulator );
83          operCode.setText( "" + operationCode );
84          register.setText( "" + instructionRegister );
85       }
86
87       // set all registers to the correct start value
88       public void initializeRegisters()
89       {
90          memory = new int[ 100 ];
91          accumulator = 0;
92          instructionCounter = 0;
93          instructionRegister = 0;
94          operand = 0;
95          operationCode = 0;
96
97          index = 0;
98          readFlag = false;
99
100         for ( int k = 0; k < memory.length; k++ )
101            memory[ k ] = 0;
102      }
103
104      // ensure value is within range
105      public boolean validation( int value )
106      {
107         if ( value < 9999 && value > -9999 )
108            return false;
109
110         else
111            return true;
112      }
113
114      // ensure that accumulator has not overflowed
115      public boolean testOverflow()
116      {
117         if ( validation( accumulator ) ) {
118            System.out.println( "*** Fatal error." +
119               "Accumulator overflow. ***" );
120            executeButton.setEnabled( false );
121
```

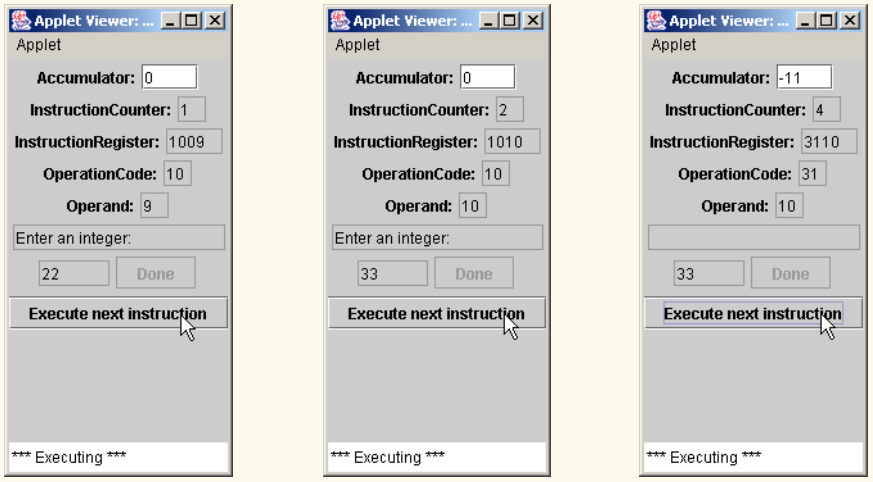
```
122            return true;
123         }
124
125         return false;
126      }
127
128      // perform all simulator functions
129      public void execute()
130      {
131         showStatus( "*** Executing ***" );
132         prompt.setText( "" );
133
134         instructionRegister = memory[ instructionCounter ];
135         operationCode = instructionRegister / 100;
136         operand = instructionRegister % 100;
137
138         switch( operationCode ) {
139            case READ:
140
141               // actual value is assigned in actionPeformed
142               readFlag = true;
143               input.setText( "" );
144               input.setEditable( true );
145               prompt.setText( "Enter an integer:" );
146               instructionCounter++;
147               break;
148
149            case WRITE:
150               System.out.println( "Contents of " + operand +
151                  " is " + memory[ operand ] );
152               instructionCounter++;
153               break;
154
155            case LOAD:
156               accumulator = memory[ operand ];
157               instructionCounter++;
158               break;
159
160            case STORE:
161               memory[ operand ] = accumulator;
162               instructionCounter++;
163               break;
164
165            case ADD:
166               accumulator += memory[ operand ];
167
168               if ( testOverflow() == true )
169                  return;
170
171               instructionCounter++;
172               break;
173
174            case SUBTRACT:
175               accumulator -= memory[ operand ];
```

```
176
177            if ( testOverflow() == true )
178               return;
179
180            instructionCounter++;
181            break;
182
183         case MULTIPLY:
184            accumulator *= memory[ operand ];
185
186            if ( testOverflow() == true )
187               return;
188
189            instructionCounter++;
190            break;
191
192         case DIVIDE:
193
194            if ( memory[ operand ] == 0 ) {
195               System.out.println( "*** Fatal error." +
196                  "Attempt to divide by zero. ***" );
197
198               executeButton.setEnabled( false );
199               return;
200            }
201
202            accumulator /= memory[ operand ];
203            instructionCounter++;
204            break;
205
206         case BRANCH:
207            instructionCounter = operand;
208            break;
209
210         case BRANCH_NEG:
211
212            if ( accumulator < 0 )
213               instructionCounter = operand;
214
215            else
216               instructionCounter++;
217
218            break;
219         case BRANCH_ZERO:
220
221            if ( accumulator == 0 )
222               instructionCounter = operand;
223            else
224
225               instructionCounter++;
226
227            break;
228
229         case HALT:
230            showStatus( "*** Simpletron execution terminated ***" );
```

```
231             executeButton.setEnabled( false );
232             dump();
233             break;
234
235          default:
236             executeButton.setEnabled( false );
237             System.out.println( "*** Fatal error. " +
238                " Invalid operation code. ***" );
239       }
240
241       // update the register textfields
242       displayRegisters();
243    }
244
245    // read in user input, test it, perform operations
246    public void actionPerformed( ActionEvent e )
247    {
248       if ( e.getSource() == input && index < 100 ) {
249          int temp = Integer.parseInt( input.getText() );
250
251          // not a read instruction
252          if ( !readFlag ) {
253
254             // allow user to reenter values
255             if ( validation( temp ) == false ) {
256                memory[ index++ ] = temp;
257
258                // clear text field
259                input.setText( "" );
260
261                if ( index < 10 )
262                   prompt.setText( "0" + index + " ? " );
263
264                else
265                   prompt.setText( index + " ? " );
266             }
267
268             else
269                input.setText( "" );
270
271             showStatus( "*** Loading ***" );
272          }
273
274          // read instruction
275          else {
276             if ( validation( temp ) == false ) {
277                memory[ operand ] = temp;
278                input.setEditable( false );
279                readFlag = false;
280                executeButton.setEnabled( true );
281             }
282
283             else {
284                input.setText( "" );
```

```
285                showStatus( "Invalid input!" );
286                executeButton.setEnabled( false );
287             }
288          }
289       }
290
291       else if ( e.getSource() == doneButton ) {
292          input.setEditable( false );
293          executeButton.setEnabled( true );
294          doneButton.setEnabled( false );
295          index = 0;
296          execute();   // execute first instruction
297       }
298
299       else if ( e.getSource() == executeButton ) {
300          index++;
301          execute();
302       }
303    }
304
305    // print out user instructions
306    public void instructions()
307    {
308       System.out.println( "*** Welcome to Simpletron! ***\n" +
309          "*** Please enter your program one instruction ***\n" +
310          "*** ( or data word ) at a time into the input ***\n" +
311          "*** text field. I will display the location ***\n" +
312          "*** number and a question mark (?). You then ***\n" +
313          "*** type the word for that location. Press the ***\n" +
314          "*** Done button to stop entering your program ***\n" );
315    }
316
317    // create a string version of the number to output
318    public String prepareNumber( int number )
319    {
320       int count = 0, factor = 1000;
321       String output;
322
323       // account for sign of number
324       if ( number < 0 ) {
325          number *= -1;
326          output = "-";
327       }
328
329       else
330          output = "+";
331
332       // build String representation of number
333       while ( factor >= 1 ) {
334          output += number / factor;
335          number %= factor;
336          factor /= 10;
337       }
338
```

```
339         return output;
340      }
341
342      // output information in registers
343      public void dump()
344      {
345         System.out.println( "REGISTERS:" );
346         System.out.print( "accumulator            " );
347         System.out.println( prepareNumber( accumulator ) );
348         System.out.print( "instructionCounter        " );
349         System.out.print( ( instructionCounter / 10 ) );
350         System.out.println( ( instructionCounter % 10 ) );
351         System.out.print( "instructionRegister     " );
352         System.out.println( prepareNumber( instructionRegister ) );
353         System.out.print( "operationCode             " );
354         System.out.print( ( operationCode / 10 ) );
355         System.out.println( ( operationCode % 10 ) );
356         System.out.print( "operand                   " );
357         System.out.print( ( operand / 10 ) );
358         System.out.println( ( operand % 10 ) );
359         System.out.println( "\nMEMORY:" );
360         System.out.println( "         0      1      2      3" +
361            "      4      5      6      7      8      9" );
362
363         for ( int k = 0; k < 10; k++ ) {
364
365            if ( k == 0 )
366               System.out.print( " " + k + "   " );
367            else
368               System.out.print( ( k * 10 ) + "   " );
369
370            for ( int i = 0; i < 10; i++ )
371               System.out.print( prepareNumber(
372                  memory[ k * 10 + i ] ) + "   " );
373
374            System.out.println();
375         }
376      }
377
378   }  // end class Simulator
```




```
*** Welcome to Simpletron! ***
*** Please enter your program one instruction ***
*** ( or data word ) at a time into the input ***
*** text field. I will display the location ***
*** number and a question mark (?). You then ***
*** type the word for that location. Press the ***
*** Done button to stop entering your program ***

Contents of 10 is 33
REGISTERS:
accumulator            -0011
instructionCounter        08
instructionRegister    +4300
operationCode             43
operand                   00

MEMORY:
        0      1      2      3      4      5      6      7      8      9
 0  +1009  +1010  +2009  +3110  +4107  +1109  +4300  +1110  +4300  +0022
10  +0033  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
20  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
30  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
40  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
50  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
60  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
70  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
80  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
90  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000  +0000
```

7.45 (*Modifications to the Simpletron Simulator*) In Exercise 7.44, you wrote a software simulation of a computer that executes programs written in Simpletron Machine Language (SML). In this exercise, we propose several modifications and enhancements to the Simpletron Simulator. In Exercise 20.26 and Exercise 20.27, we propose building a compiler that converts programs written in

a high-level programming language (a variation of Basic) to Simpletron Machine Language. Some of the following modifications and enhancements may be required to execute the programs produced by the compiler:

a) Extend the Simpletron Simulator's memory to contain 1000 memory locations to enable the Simpletron to handle larger programs.
b) Allow the simulator to perform remainder calculations. This modification requires an additional Simpletron Machine Language instruction.
c) Allow the simulator to perform exponentiation calculations. This modification requires an additional Simpletron Machine Language instruction.
d) Modify the simulator to use hexadecimal values rather than integer values to represent Simpletron Machine Language instructions.
e) Modify the simulator to allow output of a newline. This modification requires an additional Simpletron Machine Language instruction.
f) Modify the simulator to process floating-point values in addition to integer values.
g) Modify the simulator to handle string input. [*Hint*: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII decimal equivalent of a character. Add a machine-language instruction that will input a string and store the string, beginning at a specific Simpletron memory location. The first half of the word at that location will be a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction converts each character into its ASCII equivalent and assigns it to a half-word.]
h) Modify the simulator to handle output of strings stored in the format of part (g). [*Hint*: Add a machine-language instruction that will print a string, beginning at a certain Simpletron memory location. The first half of the word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.]

8

Object-Based Programming

Objectives

- To understand encapsulation and data hiding.
- To understand the notions of data abstraction and abstract data types (ADTs).
- To create Java ADTs—namely, classes.
- To be able to create and use objects.
- To be able to control access to instance variables and methods.
- To understand the use of the `this` reference.
- To be able to use class variables and methods.
- To appreciate the value of object orientation.

My object all sublime
I shall achieve in time.
W. S. Gilbert

Is it a world to hide virtues in?
William Shakespeare

Your public servants serve you right.
Adlai Stevenson

But what, to serve our private ends,
Forbids the cheating of our friends?
Charles Churchill

This above all: to thine own self be true.
William Shakespeare

Have no friends not equal to yourself.
Confucius

SELF-REVIEW EXERCISES

8.1 Fill in the blanks in each of the following statements:

a) Members of a class specified as _____ are accessible only to methods of the class.
ANS: `private`

b) A _____ is used to initialize the instance variables of a class.
ANS: constructor

c) A _____ method is used to assign values to `private` instance variables of a class.
ANS: *set* (or mutator)

d) Methods of a class are normally made _____, and instance variables of a class are normally made _____.
ANS: `public`, `private`

e) A _____ method is used to retrieve values of `private` data of a class.
ANS: *get* (or accessor)

f) The keyword _____ introduces a class declaration.
ANS: `class`

g) Members of a class specified as _____ are accessible anywhere an object of the class is in scope.
ANS: `public`

h) A _____ creates an object of a specified type and returns a _____ to that object.
ANS: class-instance creation expression, reference

i) A _____ variable represents class-wide information.
ANS: `static` (or class)

j) The keyword _____ specifies that a variable is not modifiable after it is initialized.
ANS: `final`

k) A method declared `static` cannot access _____ class members directly.
ANS: non-static

EXERCISES

8.2 Create a class called `Complex` for performing arithmetic with complex numbers. Complex numbers have the form

realPart + *imaginaryPart* * *i*

where *i* is

$\sqrt{-1}$

Write a program to test your class. Use floating-point variables to represent the `private` data of the class. Provide a constructor that enables an object of this class to be initialized when it is declared. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform the following operations:

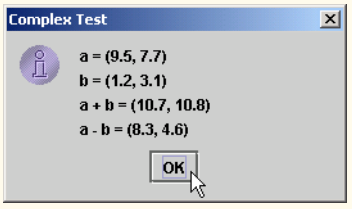
a) Add two `Complex` numbers: The real parts are added together and the imaginary parts are added together.

b) Subtract two `Complex` numbers: The real part of the right operand is subtracted from the real part of the left operand, and the imaginary part of the right operand is subtracted from the imaginary part of the left operand.

c) Print `Complex` numbers in the form `(a, b)`, where `a` is the real part and `b` is the imaginary part.

ANS:

```
// Exercise 8.2: ComplexTest.java
// Test the Complex number class
import javax.swing.*;

public class ComplexTest {

   public static void main( String args[] )
   {
      Complex a, b;
      a = new Complex( 9.5, 7.7 );
      b = new Complex( 1.2, 3.1 );

      String result = "a = " + a.toComplexString();
      result += "\nb = " + b.toComplexString();
      result += "\na + b = " + a.add( b ).toComplexString();
      result += "\na - b = " + a.subtract( b ).toComplexString();

      JOptionPane.showMessageDialog( null, result, "Complex Test",
         JOptionPane.INFORMATION_MESSAGE );
      System.exit( 0 );
   }

} // end class ComplexTest
```



ANS:

```
// Exercise 8.2 Solution: Complex.java
// Definition of class Complex

public class Complex {
   private double real;
   private double imaginary;

   // Initialize both parts to 0
   public Complex()
   {
      this( 0.0, 0.0 );
   }

   // Initialize real part to r and imaginary part to i
   public Complex( double r, double i )
   {
```

```
17          real = r;
18          imaginary = i;
19       }
20
21       // Add two Complex numbers
22       public Complex add( Complex right )
23       {
24          return new Complex( real + right.real,
25             imaginary + right.imaginary );
26       }
27
28       // Subtract two Complex numbers
29       public Complex subtract( Complex right )
30       {
31          return new Complex( real - right.real,
32             imaginary - right.imaginary );
33       }
34
35       // Return String representation of a Complex number
36       public String toComplexString()
37       {
38          return "(" + real + ", " + imaginary + ")";
39       }
40
41    } // end class Complex
```

8.3 Create a class called `Rational` for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the `private` instance variables of the class—the `numerator` and the `denominator`. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form—the fraction

```
2/4
```

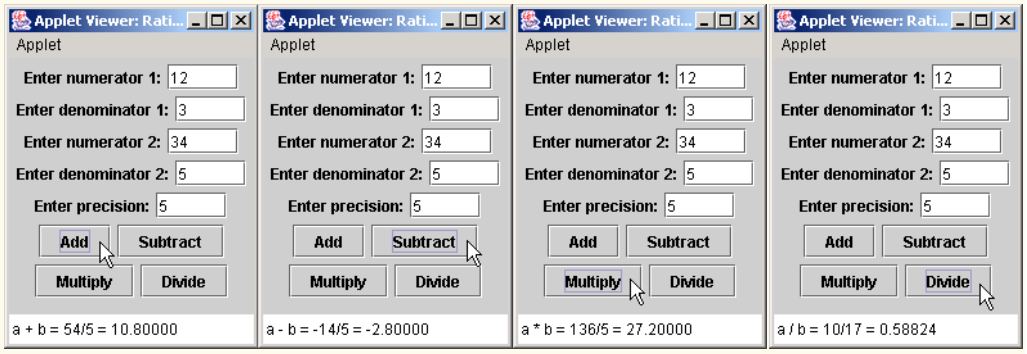
is equivalent to `1/2` and would be stored in the object as 1 in the `numerator` and 2 in the `denominator`. Provide a no-argument constructor with default values in case no initializers are provided. Provide `public` methods that perform each of the following operations:

a) Add two `Rational` numbers: The result of the addition should be stored in reduced form.
b) Subtract two `Rational` numbers: The result of the subtraction should be stored in reduced form.
c) Multiply two `Rational` numbers: The result of the multiplication should be stored in reduced form.
d) Divide of two `Rational` numbers: The result of the division should be stored in reduced form.
e) Print `Rational` numbers in the form `a/b`, where `a` is the `numerator` and `b` is the `denominator`.
f) Print `Rational` numbers in floating-point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

ANS:)

```
1  // Exercise 8.3 Solution: RationalTest.java
2  // Program tests class Rational.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class RationalTest extends JApplet implements ActionListener {
8     private JLabel numeratorlabel1, numeratorlabel2,
9        denominatorlabel1, denominatorlabel2, digitsLabel;
10    private JTextField numeratorField1, numeratorField2,
11       denominatorField1, denominatorField2, digitsField;
12    private JButton add, subtract, multiply, divide;
13
14    // set up GUI components
15    public void init()
16    {
17       // text fields for user input
18       numeratorlabel1 = new JLabel( "Enter numerator 1:" );
19       numeratorlabel2 = new JLabel( "Enter numerator 2:" );
20       denominatorlabel1 = new JLabel( "Enter denominator 1:" );
21       denominatorlabel2 = new JLabel( "Enter denominator 2:" );
22       digitsLabel = new JLabel( "Enter precision:" );
23
24       numeratorField1 = new JTextField( 5 );
25       numeratorField2 = new JTextField( 5 );
26       denominatorField1 = new JTextField( 5 );
27       denominatorField2 = new JTextField( 5 );
28       digitsField = new JTextField( 5 );
29
30       // buttons to manipulate the data
31       add = new JButton( "Add" );
32       subtract = new JButton( "Subtract" );
33       multiply = new JButton( "Multiply" );
34       divide = new JButton( "Divide" );
35
36       add.addActionListener( this );
37       subtract.addActionListener( this );
38       multiply.addActionListener( this );
39       divide.addActionListener( this );
40
41       Container container = getContentPane();
42       container.setLayout(new FlowLayout());
43
44       // add all components to GUI
45       container.add( numeratorlabel1 );
46       container.add( numeratorField1 );
47       container.add( denominatorlabel1 );
48       container.add( denominatorField1 );
49       container.add( numeratorlabel2 );
50       container.add( numeratorField2 );
51       container.add( denominatorlabel2 );
52       container.add( denominatorField2 );
```

```
53          container.add( digitsLabel );
54          container.add( digitsField );
55          container.add( add );
56          container.add( subtract );
57          container.add( multiply );
58          container.add( divide );
59
60       }  // end method init
61
62       // perform calculations
63       public void actionPerformed( ActionEvent actionEvent )
64       {
65          // read in the two rational numbers as data
66          Rational rational1, rational2, result;
67          rational1 = new Rational(
68             Integer.parseInt( numeratorField1.getText() ),
69             Integer.parseInt( denominatorField1.getText() ) );
70          rational2 = new Rational(
71             Integer.parseInt( numeratorField2.getText() ),
72             Integer.parseInt( denominatorField2.getText() ) );
73
74          // read in digits precision
75          int digits = Integer.parseInt( digitsField.getText() );
76
77          // recognize and perform the appropriate action on the data,
78          // then output the result as a floating point number
79          if ( actionEvent.getSource() == add ) {
80             result = rational1.sum( rational2 );
81             showStatus( "a + b = " + result.toRationalString() +
82                " = " + result.toFloatString( digits ) );
83          }
84
85          else if ( actionEvent.getSource() == subtract ) {
86             result = rational1.subtract( rational2 );
87             showStatus( "a - b = " + result.toRationalString() +
88                " = " + result.toFloatString( digits ) );
89          }
90
91          else if ( actionEvent.getSource() == multiply ) {
92             result = rational1.multiply( rational2 );
93             showStatus( "a * b = " + result.toRationalString() +
94                " = " + result.toFloatString( digits ) );
95          }
96
97          else if ( actionEvent.getSource() == divide ) {
98             result = rational1.divide( rational2 );
99             showStatus( "a / b = " + result.toRationalString() +
100               " = " + result.toFloatString( digits ) );
101         }
102
103      }  // end method actionPerformed
104
105   }  // end class RationalTest
```




```
1  // Exercise 8.3 Solution: Rational.java
2  // Rational class definition.
3  import java.text.DecimalFormat;
4
5  public class Rational {
6     private int numerator, denominator;
7
8     // no-argument constructor
9     public Rational()
10    {
11       numerator = 1;
12       denominator = 1;
13    }
14
15    // initialize numerator part to n and denominator part to d
16    public Rational( int theNumerator, int theDenominator )
17    {
18       numerator = theNumerator;
19       denominator = theDenominator;
20       reduce();
21    }
22
23    // add two Rational numbers
24    public Rational sum( Rational right )
25    {
26       int resultDenominator = denominator * right.denominator;
27       int resultNumerator = numerator * right.denominator +
28          right.numerator * denominator;
29
30       return new Rational( resultNumerator, resultDenominator );
31    }
32
33    // subtract two Rational numbers
34    public Rational subtract( Rational right )
35    {
36       int resultDenominator = denominator * right.denominator;
37       int resultNumerator = numerator * right.denominator -
38          right.numerator * denominator;
39
```

```
40          return new Rational( resultNumerator, resultDenominator );
41       }
42
43       // multiply two Rational numbers
44       public Rational multiply( Rational right )
45       {
46          return new Rational( numerator * right.numerator,
47             denominator * right.denominator );
48       }
49
50       // divide two Rational numbers
51       public Rational divide( Rational right )
52       {
53          return new Rational( numerator * right.denominator,
54             denominator * right.numerator );
55       }
56
57       // reduce the fraction
58       private void reduce()
59       {
60          int gcd = 0;
61          int smaller;
62
63          if ( numerator < denominator )
64             smaller = numerator;
65          else
66             smaller = denominator;
67
68          for ( int divisor = smaller; divisor >= 2; divisor-- ) {
69
70             if ( numerator % divisor == 0 && denominator % divisor == 0 ) {
71                gcd = divisor;
72                break;
73             }
74          }
75
76          if ( gcd != 0 ) {
77             numerator /= gcd;
78             denominator /= gcd;
79          }
80       }
81
82       // return String representation of a Rational number
83       public String toRationalString()
84       {
85         return numerator + "/" + denominator;
86       }
87
88       // return floating-point String representation of
89       // a Rational number
90       public String toFloatString( int digits )
91       {
92          String format = "0.";
93
```

```
94          // get format string
95          for ( int i = 0; i < digits; i++ )
96             format += "0";
97
98          DecimalFormat twoDigits = new DecimalFormat( format );
99
100         return twoDigits.format(
101            ( double )numerator / denominator ).toString();
102      }
103
104   }  // end class Rational
```

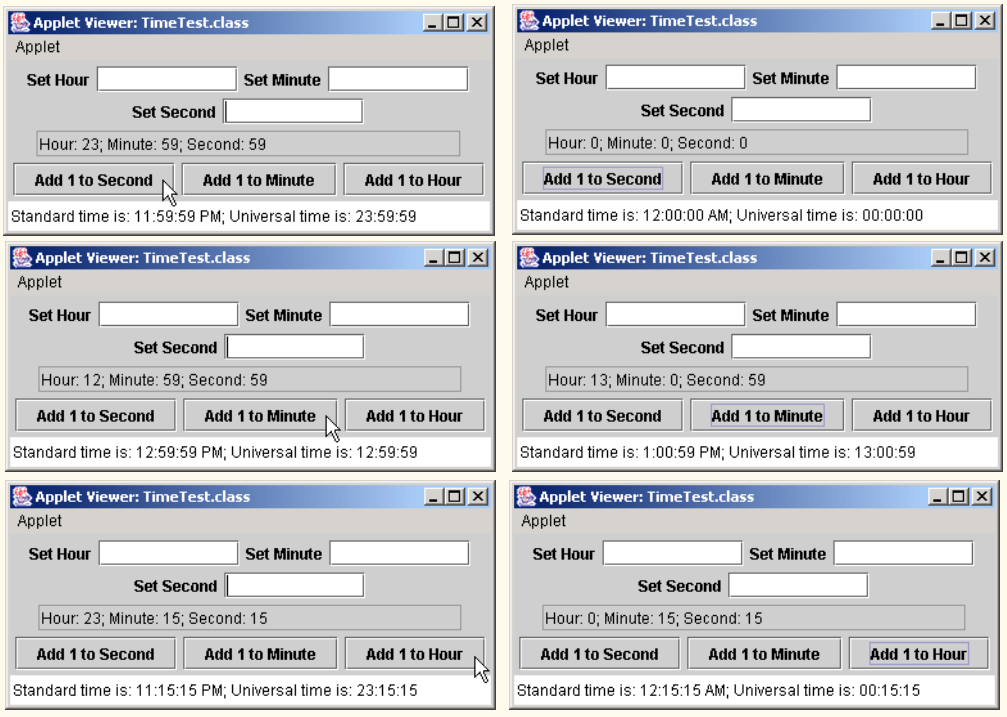
8.4 Modify class `Time3` of Fig. 8.7 to include the `tick` method that increments the time stored in a `Time3` object by one second. Also provide method `incrementMinute` to increment the minute and method `incrementHour` to increment the hour. The `Time3` object should always remain in a consistent state. Write a program that tests the `tick` method, the `incrementMinute` method and the `incrementHour` method to ensure that they work correctly. Be sure to test the following cases:

a) incrementing into the next minute.
b) incrementing into the next hour.
c) incrementing into the next day (i.e., 11:59:59 PM to 12:00:00 AM).

ANS:

```
1  // Exercise 8.4 Solution: TimeTest.java
2  // Demonstrating the Time class set and get methods
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class TimeTest extends JApplet implements ActionListener {
8     private Time3 time;
9     private JLabel hrLabel, minLabel, secLabel;
10    private JTextField hrField, minField, secField, display;
11    private JButton tickButton, minuteButton, hourButton;
12
13    public void init()
14    {
15       time = new Time3();
16
17       hrLabel = new JLabel( "Set Hour" );
18       hrField = new JTextField( 10 );
19       hrField.addActionListener( this );
20       minLabel = new JLabel( "Set Minute" );
21       minField = new JTextField( 10 );
22       minField.addActionListener( this );
23       secLabel = new JLabel( "Set Second" );
24       secField = new JTextField( 10 );
25       secField.addActionListener( this );
26       display = new JTextField( 30 );
27       display.setEditable( false );
28       tickButton = new JButton( "Add 1 to Second" );
29       tickButton.addActionListener( this );
30       minuteButton = new JButton( "Add 1 to Minute" );
```

```
31          minuteButton.addActionListener( this );
32          hourButton = new JButton( "Add 1 to Hour" );
33          hourButton.addActionListener( this );
34
35          Container container = getContentPane();
36          container.setLayout( new FlowLayout() );
37          container.add( hrLabel );
38          container.add( hrField );
39          container.add( minLabel );
40          container.add( minField );
41          container.add( secLabel );
42          container.add( secField );
43          container.add( display );
44          container.add( tickButton );
45          container.add( minuteButton );
46          container.add( hourButton );
47          updateDisplay();
48
49       } // end method init
50
51       public void actionPerformed( ActionEvent event )
52       {
53          if ( event.getSource() == tickButton )
54             time.tick();
55          else if ( event.getSource() == minuteButton )
56             time.incrementMinute();
57          else if ( event.getSource() == hourButton )
58             time.incrementHour();
59          else if ( event.getSource() == hrField ) {
60             time.setHour( Integer.parseInt( event.getActionCommand() ) );
61             hrField.setText( "" );
62          }
63          else if ( event.getSource() == minField ) {
64             time.setMinute( Integer.parseInt( event.getActionCommand() ) );
65             minField.setText( "" );
66          }
67          else if ( event.getSource() == secField ) {
68             time.setSecond( Integer.parseInt( event.getActionCommand() ) );
69             secField.setText( "" );
70          }
71
72          updateDisplay();
73
74       } // end method actionPerformed
75
76       public void updateDisplay()
77       {
78          display.setText( "Hour: " + time.getHour() + "; Minute: " +
79             time.getMinute() + "; Second: " + time.getSecond() );
80          showStatus( "Standard time is: " + time.toStandardString()+
81             "; Universal time is: " + time.toUniversalString() );
82       }
83
84    } // end class TimeTest
```




```
1  // Exercise 8.4 solution: Time3.java
2  // Time3 class definition with methods tick,
3  // incrementMinute and incrementHour.
4  import java.text.DecimalFormat;
5
6  public class Time3 extends Object {
7     private int hour;     // 0 - 23
8     private int minute;   // 0 - 59
9     private int second;   // 0 - 59
10
11    // Time3 constructor initializes each instance variable to zero.
12    // Ensures that Time object starts in a consistent state.
13    public Time3()
14    {
15       setTime( 0, 0, 0 );
16    }
17
18    // Time3 constructor: hour supplied, minute and second
19    // default to 0
20    public Time3( int h )
21    {
22       setTime( h, 0, 0 );
23    }
24
25    // Time3 constructor: hour and minute supplied, second
26    // defaulted to 0
```

```
27    public Time3( int h, int m )
28    {
29       setTime( h, m, 0 );
30    }
31
32    // Time3 constructor: hour, minute and second supplied
33    public Time3( int h, int m, int s )
34    {
35       setTime( h, m, s );
36    }
37
38    // Time3 constructor: another Time3 object supplied
39    public Time3( Time3 time )
40    {
41       setTime( time.getHour(), time.getMinute(),
42          time.getSecond() );
43    }
44
45    // Set Methods
46    // Set a new time value using universal time. Perform
47    // validity checks on data. Set invalid values to zero.
48    public void setTime( int h, int m, int s )
49    {
50       setHour( h );    // set the hour
51       setMinute( m );  // set the minute
52       setSecond( s );  // set the second
53    }
54
55    // validate and set hour
56    public void setHour( int h )
57    {
58       hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
59    }
60
61    // validate and set minute
62    public void setMinute( int m )
63    {
64       minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
65    }
66
67    // validate and set second
68    public void setSecond( int s )
69    {
70       second = ( ( s >= 0 && s < 60 ) ? s : 0 );
71    }
72
73    // Get Methods
74    // get hour value
75    public int getHour()
76    {
77       return hour;
78    }
79
```

```
80    // get minute value
81    public int getMinute()
82    {
83       return minute;
84    }
85
86    // get second value
87    public int getSecond()
88    {
89       return second;
90    }
91
92    // convert to String in universal-time format
93    public String toUniversalString()
94    {
95       DecimalFormat twoDigits = new DecimalFormat( "00" );
96
97       return twoDigits.format( getHour() ) + ":" +
98          twoDigits.format( getMinute() ) + ":" +
99          twoDigits.format( getSecond() );
100   }
101
102   // convert to String in standard-time format
103   public String toStandardString()
104   {
105      DecimalFormat twoDigits = new DecimalFormat( "00" );
106
107      return ( ( getHour() == 12 || getHour() == 0 ) ?
108         12 : getHour() % 12 ) + ":" + twoDigits.format( getMinute() ) +
109         ":" + twoDigits.format( getSecond() ) +
110         ( getHour() < 12 ? " AM" : " PM" );
111   }
112
113   // Tick the time by one second
114   public void tick()
115   {
116      setSecond( second + 1 );
117
118      if ( second == 0 )
119         incrementMinute();
120   }
121
122   // Increment the minute
123   public void incrementMinute()
124   {
125      setMinute( minute + 1 );
126
127      if ( minute == 0 )
128         incrementHour();
129   }
130
131   // Increment the hour
132   public void incrementHour()
133   {
```

```
134         setHour( hour + 1 );
135      }
136
137   }  // end class Time3
```

8.5 Modify class `Date` of Fig. 8.9 to perform error-checking on the initializer values for instance variables `month`, `day` and `year` (currently it validates only the month and day). Also, provide a method `nextDay` to increment the day by one. The `Date` object should always remain in a consistent state. Write a program that tests the `nextDay` method in a loop that prints the date during each iteration of the loop to illustrate that the `nextDay` method works correctly. Test the following cases:

a) incrementing into the next month.
b) incrementing into the next year.

ANS:

```
1   // Exercise 8.5 Solution: DateTest
2   // Program tests Date class.
3
4   public class DateTest {
5
6      // method main begins execution of Java application
7      public static void main( String args[] )
8      {
9         System.out.println( "Checking increment" );
10        Date testDate = new Date( 11, 27, 1988 );
11
12        // test incrementing of day, month and year
13        for ( int counter = 0; counter < 40; counter++ ) {
14           testDate.nextDay();
15           System.out.println( "Incremented Date:" +
16              testDate.toDateString() );
17        }
18     }
19
20  }  // end class DateTest
```

```
Checking increment
Date object constructor for date 11/27/1988
Incremented Date:11/28/1988
Incremented Date:11/29/1988
Incremented Date:11/30/1988
Day 31 invalid. Set to day 1.
Incremented Date:12/1/1988
Incremented Date:12/2/1988
...
Incremented Date:12/30/1988
Incremented Date:12/31/1988
Day 32 invalid. Set to day 1.
Incremented Date:1/1/1989
Incremented Date:1/2/1989
Incremented Date:1/3/1989
Incremented Date:1/4/1989
Incremented Date:1/5/1989
Incremented Date:1/6/1989
```

```
1  // Exercise 8.5 Solution: Date.java
2  // Date class definition.
3
4  public class Date {
5     private int month;  // 1-12
6     private int day;    // 1-31 based on month
7     private int year;   // any year
8
9     // constructor: confirm proper value for month;
10    // call method function checkDay to confirm proper value for day.
11    public Date( int theMonth, int theDay, int theYear )
12    {
13       if ( theMonth > 0 && theMonth <= 12 )   // validate month
14          month = theMonth;
15       else {
16          month = 1;
17          System.out.println( "Month " + theMonth +
18             " invalid. Set to month 1." );
19       }
20
21       if ( theYear > 0 )                      // validate year
22          year = theYear;
23       else {
24          year = 1;
25          System.out.println( "Year " + theYear +
26             " invalid. Set to Year 1." );
27       }
28
29       day = checkDay ( theDay );              // validate day
30
31       System.out.println(
32          "Date object constructor for date " + toDateString() );
33    }
34
35    // utility method to confirm proper day value
36    // based on month and year.
37    public int checkDay( int testDay )
38    {
39       int daysPerMonth[] =
40          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
41
42       // check if day in range for month
43       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
44          return testDay;
45
46       // check for leap year
47       if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
48          ( year % 4 == 0 && year % 100 != 0 ) ) )
49          return testDay;
50
51       System.out.println( "Day " + testDay + " invalid. Set to day 1." );
52
53       return 1;   // leave object in consistent state
54    }
```

```
55
56    // increment the day and check if doing so will change the month
57    public void nextDay()
58    {
59       int testDay = day + 1;
60
61       if ( checkDay( testDay ) == testDay )
62          day = testDay;
63       else {
64          day = 1;
65          nextMonth();
66       }
67    }
68
69    // increment the month and check if doing so will change the year
70    public void nextMonth()
71    {
72       if ( 12 == month )
73         year++;
74
75       month = month % 12 + 1;
76    }
77
78    // create a String of the form month/day/year
79    public String toDateString()
80    {
81       return month + "/" + day + "/" + year;
82    }
83
84 }  // end class Date
```

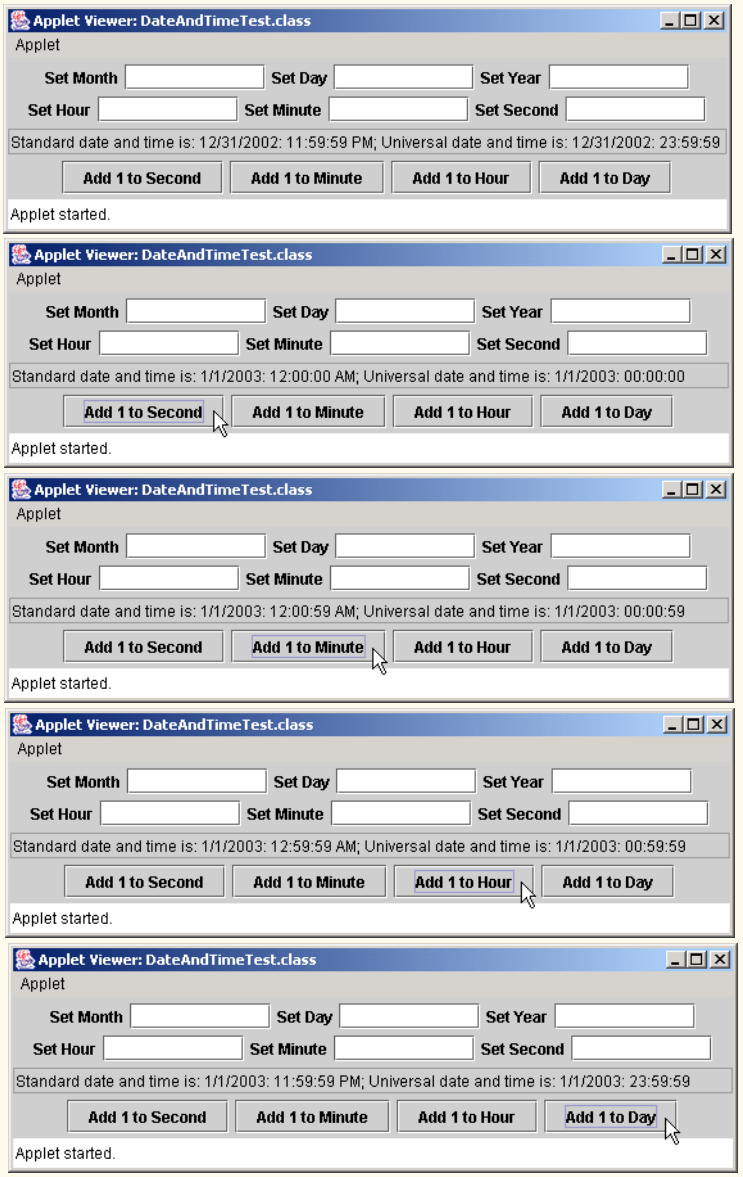
8.6 Create class `DateAndTime` that combines the modified `Time3` class of Exercise 8.4 and the modified `Date` class of Exercise 8.5. Modify method `incrementHour` to call method `nextDay` if the time is incremented into the next day. Modify methods `toStandardString` and `toUniversalString` to output the date in addition to the time. Write a program to test the new class `DateAndTime`. Specifically, test incrementing the time to the next day.

ANS:

```
1  // Exercise 8.6 Solution: DateAndTimeTest.java
2  // Demonstrating the DateAndTime class methods
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class DateAndTimeTest extends JApplet implements ActionListener {
8     private DateAndTime dateTime;
9     private JLabel hrLabel, minLabel, secLabel;
10    private JLabel yearLabel, monthLabel, dayLabel;
11    private JTextField hrField, minField, secField;
12    private JTextField yearField, monthField, dayField, display;
13    private JButton tickButton, minuteButton, hourButton, dayButton;
14
```

```
15     public void init()
16     {
17        dateTime = new DateAndTime();
18
19        // input area for time part
20        hrLabel = new JLabel( "Set Hour" );
21        hrField = new JTextField( 10 );
22        hrField.addActionListener( this );
23        minLabel = new JLabel( "Set Minute" );
24        minField = new JTextField( 10 );
25        minField.addActionListener( this );
26        secLabel = new JLabel( "Set Second" );
27        secField = new JTextField( 10 );
28        secField.addActionListener( this );
29
30        // input area for date part
31        monthLabel = new JLabel( "Set Month" );
32        monthField = new JTextField( 10 );
33        dayLabel = new JLabel( "Set Day" );
34        dayField = new JTextField( 10 );
35        yearLabel = new JLabel( "Set Year" );
36        yearField = new JTextField( 10 );
37        yearField.addActionListener( this );
38
39        // display area
40        display = new JTextField( 51 );
41        display.setEditable( false );
42
43        // buttons for time part
44        tickButton = new JButton( "Add 1 to Second" );
45        tickButton.addActionListener( this );
46        minuteButton = new JButton( "Add 1 to Minute" );
47        minuteButton.addActionListener( this );
48        hourButton = new JButton( "Add 1 to Hour" );
49        hourButton.addActionListener( this );
50
51        // buttons for date part
52        dayButton = new JButton( "Add 1 to Day" );
53        dayButton.addActionListener( this );
54
55        Container container = getContentPane();
56        container.setLayout( new FlowLayout() );
57        container.add( monthLabel );
58        container.add( monthField );
59        container.add( dayLabel );
60        container.add( dayField );
61        container.add( yearLabel );
62        container.add( yearField );
63        container.add( hrLabel );
64        container.add( hrField );
65        container.add( minLabel );
66        container.add( minField );
67        container.add( secLabel );
68        container.add( secField );
```

```
69          container.add( display );
70          container.add( tickButton );
71          container.add( minuteButton );
72          container.add( hourButton );
73          container.add( dayButton );
74          updateDisplay();
75
76       } // end method init
77
78       public void actionPerformed( ActionEvent event )
79       {
80          if ( event.getSource() == tickButton )
81             dateTime.tick();
82          else if ( event.getSource() == minuteButton )
83             dateTime.incrementMinute();
84          else if ( event.getSource() == hourButton )
85             dateTime.incrementHour();
86          else if ( event.getSource() == dayButton )
87             dateTime.nextDay();
88          else if ( event.getSource() == yearField ) {
89             dateTime.setDate( Integer.parseInt( monthField.getText() ),
90                Integer.parseInt( dayField.getText() ),
91                Integer.parseInt( yearField.getText() ) );
92             monthField.setText( "" );
93             dayField.setText( "" );
94             yearField.setText( "" );
95          }
96          else if ( event.getSource() == hrField ) {
97             dateTime.setHour( Integer.parseInt( event.getActionCommand() ) );
98             hrField.setText( "" );
99          }
100         else if ( event.getSource() == minField ) {
101            dateTime.setMinute( Integer.parseInt(
102               event.getActionCommand() ) );
103            minField.setText( "" );
104         }
105         else if ( event.getSource() == secField ) {
106            dateTime.setSecond( Integer.parseInt(
107               event.getActionCommand() ) );
108            secField.setText( "" );
109         }
110
111         updateDisplay();
112
113      } // end method actionPerformed
114
115      public void updateDisplay()
116      {
117         display.setText( "Standard date and time is: " +
118            dateTime.toStandardString() + "; Universal date and time is: " +
119            dateTime.toUniversalString() );
120      }
121
122   } // end class DateAndTimeTest
```




```
1  // Exercise 8.6 Solution: DateAndTime.java
2  // DateAndTime class definition.
3  import java.text.DecimalFormat;
4
5  public class DateAndTime {
6     private int month;  // 1-12
7     private int day;    // 1-31 based on month
8     private int year;   // any year
```

```
 9     private int hour;     // 0 - 23
10     private int minute;   // 0 - 59
11     private int second;   // 0 - 59
12
13     // no argument constructor
14     public DateAndTime()
15     {
16        setDate( 1, 1, 2000 );
17        setTime( 0, 0, 0 );
18     }
19
20     // constructor
21     public DateAndTime( int theMonth, int theDay, int theYear,
22        int theHour, int theMinute, int theSecond )
23     {
24        setDate( theMonth, theDay, theYear );
25        setTime( theHour, theMinute, theSecond );
26     }
27
28     // Set a new date value. Perform
29     // validity checks on data. Set invalid values to one.
30     public void setDate( int theMonth, int theDay, int theYear )
31     {
32        if ( theMonth > 0 && theMonth <= 12 )   // validate month
33           month = theMonth;
34        else
35           month = 1;
36
37        if ( theYear > 0 )                     // validate year
38           year = theYear;
39        else
40           year = 1;
41
42        day = checkDay ( theDay );             // validate day
43
44     } // end method setDate
45
46     // Set a new time value using universal time. Perform
47     // validity checks on data. Set invalid values to zero.
48     public void setTime( int h, int m, int s )
49     {
50        setHour( h );    // set the hour
51        setMinute( m );  // set the minute
52        setSecond( s );  // set the second
53     }
54
55     // validate and set hour
56     public void setHour( int h )
57     {
58        hour = ( ( h >= 0 && h < 24 ) ? h : 0 );
59     }
60
61     // validate and set minute
62     public void setMinute( int m )
63     {
```

```
64          minute = ( ( m >= 0 && m < 60 ) ? m : 0 );
65       }
66
67       // validate and set second
68       public void setSecond( int s )
69       {
70          second = ( ( s >= 0 && s < 60 ) ? s : 0 );
71       }
72
73       // Get Methods
74       // get hour value
75       public int getHour()
76       {
77          return hour;
78       }
79
80       // get minute value
81       public int getMinute()
82       {
83          return minute;
84       }
85
86       // get second value
87       public int getSecond()
88       {
89          return second;
90       }
91
92       // Tick the time by one second
93       public void tick()
94       {
95          setSecond( second + 1 );
96
97          if ( second == 0 )
98             incrementMinute();
99       }
100
101      // Increment the minute
102      public void incrementMinute()
103      {
104         setMinute( minute + 1 );
105
106         if ( minute == 0 )
107            incrementHour();
108      }
109
110      // Increment the hour
111      public void incrementHour()
112      {
113         setHour( hour + 1 );
114
115         if ( hour == 0 )
116            nextDay();
117      }
```

```
118
119    // utility method to confirm proper day value
120    // based on month and year.
121    public int checkDay( int testDay )
122    {
123       int daysPerMonth[] =
124          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
125
126       // check if day in range for month
127       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
128          return testDay;
129
130       // check for leap year
131       if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
132          ( year % 4 == 0 && year % 100 != 0 ) ) )
133          return testDay;
134
135       return 1;   // leave object in consistent state
136    }
137
138    // increment the day and check if doing so will change the month
139    public void nextDay()
140    {
141       int testDay = day + 1;
142
143       if ( checkDay( testDay ) == testDay )
144          day = testDay;
145       else {
146          day = 1;
147          nextMonth();
148       }
149    }
150
151    // increment the month and check if doing so will change the year
152    public void nextMonth()
153    {
154       if ( 12 == month )
155         year++;
156
157       month = month % 12 + 1;
158    }
159
160    // convert to String in universal-time format
161    public String toUniversalString()
162    {
163       DecimalFormat twoDigits = new DecimalFormat( "00" );
164
165       return month + "/" + day + "/" + year + ": " +
166          twoDigits.format( getHour() ) + ":" +
167          twoDigits.format( getMinute() ) + ":" +
168          twoDigits.format( getSecond() );
169    }
170
```

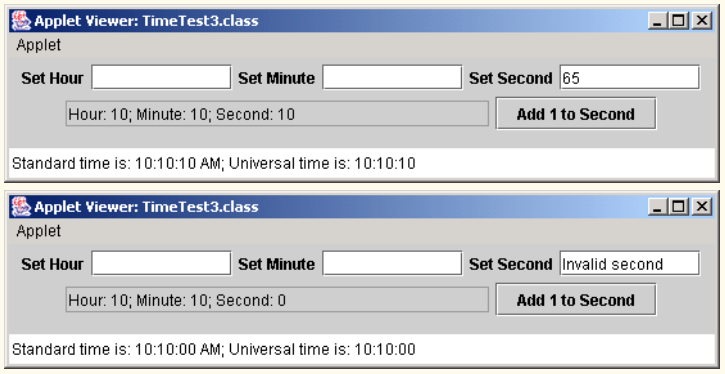
```
171    // convert to String in standard-time format
172    public String toStandardString()
173    {
174       DecimalFormat twoDigits = new DecimalFormat( "00" );
175
176       return month + "/" + day + "/" + year + ": " +
177          ( ( getHour() == 12 || getHour() == 0 ) ? 12 : getHour() % 12 ) +
178          ":" + twoDigits.format( getMinute() ) + ":" +
179          twoDigits.format( getSecond() ) +
180          ( getHour() < 12 ? " AM" : " PM" );
181    }
182
183 }  // end class DateAndTime
```

8.7 Modify the *set* methods in class `Time3` of Fig. 8.7 to return appropriate error values if an attempt is made to set one of the instance variables `hour`, `minute` or `second` of an object of class `Time` to an invalid value. (*Hint:* Use `boolean` return types on each method.) Write a program that tests these new *set* methods and outputs error messages when incorrect values are supplied.

ANS:

```
1  // Exercise 8.7 Solution: TimeTest3.java
2  // Program adds validation to Fig. 8.7 example
3  import java.awt.*;
4  import javax.swing.*;
5  import java.awt.event.*;
6
7  public class TimeTest3 extends JApplet implements ActionListener {
8     private Time3 time;
9     private JLabel hrLabel, minLabel, secLabel;
10    private JTextField hrField, minField, secField, display;
11    private JButton tickButton;
12
13    public void init()
14    {
15       time = new Time3();
16
17       hrLabel = new JLabel( "Set Hour" );
18       hrField = new JTextField( 10 );
19       hrField.addActionListener( this );
20       minLabel = new JLabel( "Set Minute" );
21       minField = new JTextField( 10 );
22       minField.addActionListener( this );
23       secLabel = new JLabel( "Set Second" );
24       secField = new JTextField( 10 );
25       secField.addActionListener( this );
26       display = new JTextField( 30 );
27       display.setEditable( false );
28       tickButton = new JButton( "Add 1 to Second" );
29       tickButton.addActionListener( this );
30
31       Container c = getContentPane();
32       c.setLayout( new FlowLayout() );
```

```
33          c.add( hrLabel );
34          c.add( hrField );
35          c.add( minLabel );
36          c.add( minField );
37          c.add( secLabel );
38          c.add( secField );
39          c.add( display );
40          c.add( tickButton );
41          updateDisplay();
42       }
43
44       public void actionPerformed( ActionEvent event )
45       {
46          if ( event.getSource() == tickButton )
47             time.tick();
48          else if ( event.getSource() == hrField ) {
49             if( time.setHour(
50                Integer.parseInt( event.getActionCommand() ) ) )
51                hrField.setText( "" );
52             else
53                hrField.setText( "Invalid hour" );
54          }
55          else if ( event.getSource() == minField ) {
56             if ( time.setMinute(
57                Integer.parseInt( event.getActionCommand() ) ) )
58                minField.setText( "" );
59             else
60                minField.setText( "Invalid minute" );
61          }
62          else if ( event.getSource() == secField ) {
63             if ( time.setSecond(
64                Integer.parseInt( event.getActionCommand() ) ) )
65                secField.setText( "" );
66             else
67                secField.setText( "Invalid second" );
68          }
69
70          showStatus( "" );
71          updateDisplay();
72       }
73
74       public void updateDisplay()
75       {
76          display.setText( "Hour: " + time.getHour() + "; Minute: " +
77             time.getMinute() + "; Second: " + time.getSecond() );
78          showStatus( "Standard time is: " + time.toStandardString()+
79             "; Universal time is: " + time.toUniversalString() );
80       }
81
82    } // end class TimeTest3
```




```
1  // Exercise 8.7 solution: Time3.java
2  // Time3 class definition with set and get methods
3  import java.text.DecimalFormat;
4
5  public class Time3 extends Object {
6     private int hour;     // 0 - 23
7     private int minute;   // 0 - 59
8     private int second;   // 0 - 59
9
10    // Time3 constructor initializes each instance variable
11    // to zero. Ensures that Time object starts in a
12    // consistent state.
13    public Time3()
14    {
15       setTime( 0, 0, 0 );
16    }
17
18    // Time3 constructor: hour supplied, minute and second
19    // defaulted to 0
20    public Time3( int h )
21    {
22       setTime( h, 0, 0 );
23    }
24
25    // Time3 constructor: hour and minute supplied, second
26    // defaulted to 0
27    public Time3( int h, int m )
28    {
29       setTime( h, m, 0 );
30    }
31
32    // Time3 constructor: hour, minute and second supplied
33    public Time3( int h, int m, int s )
34    {
35       setTime( h, m, s );
36    }
37
```

```
38    // Time3 constructor: another Time3 object supplied
39    public Time3( Time3 time )
40    {
41       setTime( time.getHour(), time.getMinute(),
42          time.getSecond() );
43    }
44
45    // Set Methods
46    // Set a new time value using universal time. Perform
47    // validity checks on data. Set invalid values to zero.
48    public boolean setTime( int h, int m, int s )
49    {
50       boolean hourValid = setHour( h );    // set the hour
51       boolean minuteValid = setMinute( m );  // set the minute
52       boolean secondValid = setSecond( s );  // set the second
53
54       return ( hourValid && minuteValid && secondValid );
55
56    }
57
58    // validate and set hour
59    public boolean setHour( int h )
60    {
61       if ( h >= 0 && h < 24 ) {
62          hour = h;
63          return true;
64       }
65       else {
66          hour = 0;
67          return false;
68       }
69    }
70
71    // validate and set minute
72    public boolean setMinute( int m )
73    {
74       if ( m >= 0 && m < 60 ) {
75          minute = m;
76          return true;
77       }
78       else {
79          minute = 0;
80          return false;
81       }
82    }
83
84    // validate and set second
85    public boolean setSecond( int s )
86    {
87       if ( s >= 0 && s < 60 ) {
88          second = s;
89          return true;
90       }
91       else {
```

```
92             second = 0;
93             return false;
94          }
95       }
96
97       // Get Methods
98       // get hour value
99       public int getHour()
100      {
101         return hour;
102      }
103
104      // get minute value
105      public int getMinute()
106      {
107         return minute;
108      }
109
110      // get second value
111      public int getSecond()
112      {
113         return second;
114      }
115
116      // convert to String in universal-time format
117      public String toUniversalString()
118      {
119         DecimalFormat twoDigits = new DecimalFormat( "00" );
120
121         return twoDigits.format( getHour() ) + ":" +
122            twoDigits.format( getMinute() ) + ":" +
123            twoDigits.format( getSecond() );
124      }
125
126      // convert to String in standard-time format
127      public String toStandardString()
128      {
129         DecimalFormat twoDigits = new DecimalFormat( "00" );
130
131         return ( ( getHour() == 12 || getHour() == 0 ) ?
132            12 : getHour() % 12 ) + ":" + twoDigits.format( getMinute() ) +
133            ":" + twoDigits.format( getSecond() ) +
134            ( getHour() < 12 ? " AM" : " PM" );
135      }
136
137      // Tick the time by one second
138      public void tick()
139      {
140         setSecond( second + 1 );
141
142         if ( second == 0 )
143            incrementMinute();
144      }
145
```

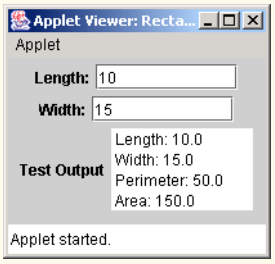
```
146    // Increment the minute
147    public void incrementMinute()
148    {
149       setMinute( minute + 1 );
150
151       if ( minute == 0 )
152          incrementHour();
153    }
154
155    // Increment the hour
156    public void incrementHour()
157    {
158       setHour( hour + 1 );
159    }
160
161 }  // end class Time3
```

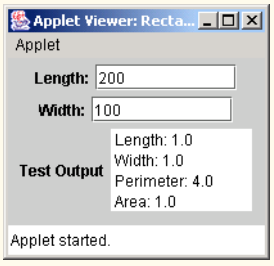
8.8 Create a class `Rectangle`. The class has attributes `length` and `width`, each of which defaults to 1. It has methods that calculate the `perimeter` and the `area` of the rectangle. It has *set* and *get* methods for both `length` and `width`. The *set* methods should verify that `length` and `width` are each floating-point numbers larger than 0.0 and less than 20.0. Write a program to test class `Rectangle`.

ANS:

```
1  // Exercise 8.8 Solution: RectangleTest.java
2  // Program tests class Rectangle.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class RectangleTest extends JApplet implements ActionListener {
8     private JLabel prompt1, prompt2;
9     private JTextField inputField1, inputField2;
10    private JLabel outputLabel;
11    private JTextArea outputArea;
12    private Rectangle rectangle;
13
14    // set up GUI components and instantiate new Rectangle
15    public void init()
16    {
17       prompt1 = new JLabel( "Length:" );
18       prompt2 = new JLabel( "Width:" );
19       inputField1 = new JTextField( 10 );
20       inputField2 = new JTextField( 10 );
21       inputField2.addActionListener( this );
22
23       outputLabel = new JLabel( "Test Output" );
24       outputArea = new JTextArea( 4, 10 );
25       outputArea.setEditable( false );
26
27       // add components to GUI
28       Container container = getContentPane();
```

```
29          container.setLayout( new FlowLayout() );
30          container.add( prompt1 );
31          container.add( inputField1 );
32          container.add( prompt2 );
33          container.add( inputField2 );
34          container.add( outputLabel );
35          container.add( outputArea );
36
37          // create a new Rectangle with no initial values
38          rectangle = new Rectangle();
39       }
40
41       // create rectangle with user input
42       public void actionPerformed( ActionEvent actionEvent )
43       {
44          double double1, double2;
45
46          double1 = Double.parseDouble( inputField1.getText() );
47          double2 = Double.parseDouble( inputField2.getText() );
48
49          rectangle.setLength( double1 );
50          rectangle.setWidth( double2 );
51
52          // see the results of the test
53          outputArea.setText( rectangle.toRectangleString() );
54       }
55
56    } // end class RectangleTest
```






```
1   // Exercise 8.8 Solution: Rectangle.java
2   // Definition of class Rectangle
3
4   public class Rectangle {
5      private double length, width;
6
7      // constructor without paramters
8      public Rectangle()
9      {
10        setLength( 1.0 );
11        setWidth( 1.0 );
12     }
13
```

```
14    // constructor with length and width supplied
15    public Rectangle( double theLength, double theWidth )
16    {
17       setLength( theLength );
18       setWidth( theWidth );
19    }
20
21    // validate and set length
22    public void setLength( double theLength )
23    {
24       length = ( theLength > 0.0 && theLength < 20.0 ? theLength : 1.0 );
25    }
26
27    // validate and set width
28    public void setWidth( double theWidth )
29    {
30       width = ( theWidth > 0 && theWidth < 20.0 ? theWidth : 1.0 );
31    }
32
33    // get value of length
34    public double getLength()
35    {
36       return length;
37    }
38
39    // get value of width
40    public double getWidth()
41    {
42       return width;
43    }
44
45    // calculate rectangle's perimeter
46    public double perimeter()
47    {
48       return 2 * length + 2 * width;
49    }
50
51    // calculate rectangle's area
52    public double area()
53    {
54       return length * width;
55    }
56
57    // convert to String
58    public String toRectangleString ()
59    {
60        return ( " Length: " + length + "\n" + " Width: " + width  + "\n" +
61           " Perimeter: " + perimeter() + "\n" + " Area: " + area() );
62
63    }
64
65 } // end class Rectangle
```

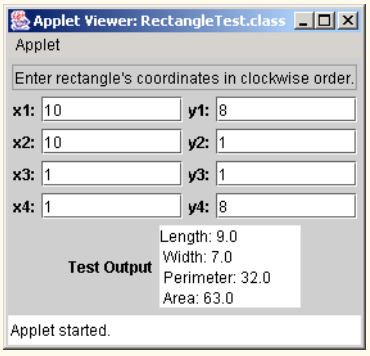
8.9 Create a more sophisticated `Rectangle` class than the one you created in Exercise 8.8. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a *set* method that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single *x*- or *y*-coordinate larger than 20.0. The *set* method also verifies that the supplied coordinates specify a rectangle. Provide methods to calculate the `length`, `width`, `perimeter` and `area`. The length is the larger of the two dimensions. Include a predicate method `isSquare` which determines whether the rectangle is a square. Write a program to test class `Rectangle`.

ANS:

```
1  // Exercise 8.9 Solution: RectangleTest.java
2  // Program tests class Rectangle.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class RectangleTest extends JApplet implements ActionListener {
8     private JLabel x1Prompt, y1Prompt, x2Prompt, y2Prompt,
9        x3Prompt, y3Prompt, x4Prompt, y4Prompt;
10    private JTextField directions, x1Field, y1Field, x2Field,
11       y2Field, x3Field, y3Field, x4Field, y4Field;
12    private JLabel outputLabel;
13    private JTextArea outputArea;
14
15    // set up GUI components and instantiate new Rectangle
16    public void init()
17    {
18       x1Prompt = new JLabel( "x1:" );
19       y1Prompt = new JLabel( "y1:" );
20       x2Prompt = new JLabel( "x2:" );
21       y2Prompt = new JLabel( "y2:" );
22       x3Prompt = new JLabel( "x3:" );
23       y3Prompt = new JLabel( "y3:" );
24       x4Prompt = new JLabel( "x4:" );
25       y4Prompt = new JLabel( "y4:" );
26
27       directions = new JTextField(
28          "Enter rectangle's coordinates in clockwise order.");
29       directions.setEditable( false );
30
31       // text fields for the four points' x and y values
32       x1Field = new JTextField( 10 );
33       y1Field = new JTextField( 10 );
34       x2Field = new JTextField( 10 );
35       y2Field = new JTextField( 10 );
36       x3Field = new JTextField( 10 );
37       y3Field = new JTextField( 10 );
38       x4Field = new JTextField( 10 );
39       y4Field = new JTextField( 10 );
40       y4Field.addActionListener( this );
41
42       // output area for rectangle information
43       outputLabel = new JLabel( "Test Output" );
44       outputArea = new JTextArea( 4, 10 );
```

```
45          outputArea.setEditable( false );
46
47          // add components to GUI
48          Container container = getContentPane();
49          container.setLayout( new FlowLayout() );
50          container.add( directions );
51          container.add( x1Prompt );
52          container.add( x1Field );
53          container.add( y1Prompt );
54          container.add( y1Field );
55          container.add( x2Prompt );
56          container.add( x2Field );
57          container.add( y2Prompt );
58          container.add( y2Field );
59          container.add( x3Prompt );
60          container.add( x3Field );
61          container.add( y3Prompt );
62          container.add( y3Field );
63          container.add( x4Prompt );
64          container.add( x4Field );
65          container.add( y4Prompt );
66          container.add( y4Field );
67          container.add( outputLabel );
68          container.add( outputArea );
69
70       }  // end method init
71
72       // create rectangle with user input
73       public void actionPerformed( ActionEvent actionEvent )
74       {
75          double x1, x2, x3, x4, y1, y2, y3, y4;
76
77          x1 = Double.parseDouble( x1Field.getText() );
78          x2 = Double.parseDouble( x2Field.getText() );
79          x3 = Double.parseDouble( x3Field.getText() );
80          x4 = Double.parseDouble( x4Field.getText() );
81          y1 = Double.parseDouble( y1Field.getText() );
82          y2 = Double.parseDouble( y2Field.getText() );
83          y3 = Double.parseDouble( y3Field.getText() );
84          y4 = Double.parseDouble( y4Field.getText() );
85
86          // create a new shape with the points and determine whether
87          // it is actually a rectangle.
88          Rectangle rectangle =
89             new Rectangle( x1, y1, x2, y2, x3, y3, x4, y4 );
90
91          if ( rectangle.isRectangle() )
92             outputArea.setText( rectangle.toRectangleString() );
93
94          else
95             outputArea.setText( "");
96
97       }  // end method actionPerformed
98
```

```
99 } // end class RectangleTest
```




```
1  // Exercise 8.9 Solution: Rectangle.java
2  // Definition of class Rectangle
3  import javax.swing.JOptionPane;
4
5  public class Rectangle {
6     private double x1, x2, x3, x4, y1, y2, y3, y4;
7
8     // no-argument constructor
9     public Rectangle()
10    {
11       setCoordinates( 1, 1, 1, 1, 1, 1, 1, 1 );
12    }
13
14    // constructor
15    public Rectangle( double x1, double y1, double x2,
16       double y2, double x3, double y3, double x4, double y4 )
17    {
18       setCoordinates( x1, y1, x2, y2, x3, y3, x4, y4 );
19    }
20
21    // check if coordinates are valid
22    public void setCoordinates( double xInput1, double yInput1,
23       double xInput2, double yInput2, double xInput3,
24       double yInput3, double xInput4, double yInput4 )
25    {
26       x1 = ( xInput1 >= 0.0 && xInput1 <= 20.0 ? xInput1 : 1 );
27       x2 = ( xInput2 >= 0.0 && xInput2 <= 20.0 ? xInput2 : 1 );
28       x3 = ( xInput3 >= 0.0 && xInput3 <= 20.0 ? xInput3 : 1 );
29       x4 = ( xInput4 >= 0.0 && xInput4 <= 20.0 ? xInput4 : 1 );
30       y1 = ( yInput1 >= 0.0 && yInput1 <= 20.0 ? yInput1 : 1 );
31       y2 = ( yInput2 >= 0.0 && yInput2 <= 20.0 ? yInput2 : 1 );
32       y3 = ( yInput3 >= 0.0 && yInput3 <= 20.0 ? yInput3 : 1 );
33       y4 = ( yInput4 >= 0.0 && yInput4 <= 20.0 ? yInput4 : 1 );
34
35       if ( !isRectangle() )
36          JOptionPane.showMessageDialog( null, "This is not a rectangle.",
37             "Information", JOptionPane.INFORMATION_MESSAGE );
```

```
38    }
39
40    // calculate distance between two points
41    public double distance( double x1, double y1, double x2, double y2 )
42    {
43       double distance;
44
45       // calculate vertical lines
46       if ( x1 == x2 )
47          distance = y1 - y2 ;
48
49       // calculate horizontal lines
50       else if ( y1 == y2 )
51          distance = x1 - x2 ;
52
53       // calculate lines that aren't horizontal or vertical
54       else
55          distance = Math.sqrt( ( Math.pow( x1 - x2, 2 ) +
56             Math.pow( y1 - y2, 2 ) ) );
57
58       if ( distance < 0 )
59          distance *= -1;
60
61       return distance;
62    }
63
64    // check if coordinates specify a rectangle by determining if the
65    // two diagonals are of the same length.
66    public boolean isRectangle()
67    {
68       double side1 = distance( x1, y1, x2, y2 );
69       double side2 = distance( x2, y2, x3, y3 );
70       double side3 = distance( x3, y3, x4, y4 );
71
72       if ( side1 * side1 + side2 * side2 ==
73          side2 * side2 + side3 * side3 )
74          return true;
75
76       else
77          return false;
78    }
79
80    // check if rectangle is a square
81    public boolean isSquare()
82    {
83       return ( getLength() == getWidth() );
84    }
85
86    // get length of rectangle
87    public double getLength()
88    {
89       double side1 = distance( x1, y1, x2, y2 );
90       double side2 = distance( x2, y2, x3, y3 );
91
```

```
92          return ( side1 > side2 ? side1 : side2 );
93       }
94
95       // get width of rectangle
96       public double getWidth()
97       {
98          double side1 = distance( x1, y1, x2, y2 );
99          double side2 = distance( x2, y2, x3, y3 );
100
101         return ( side1 < side2 ? side1 : side2 );
102      }
103
104      // calculate perimeter
105      public double perimeter()
106      {
107         return 2 * getLength() + 2 * getWidth();
108      }
109
110      // calculate area
111      public double area()
112      {
113         return getLength() * getWidth();
114      }
115
116      // convert to String
117      public String toRectangleString()
118      {
119         return ( "Length: " + getLength() + "\n" + " Width: " + getWidth() +
120             "\n" + " Perimeter: " + perimeter() + "\n" + " Area: " +
121             area() );
122      }
123
124   }  // end class Rectangle
```

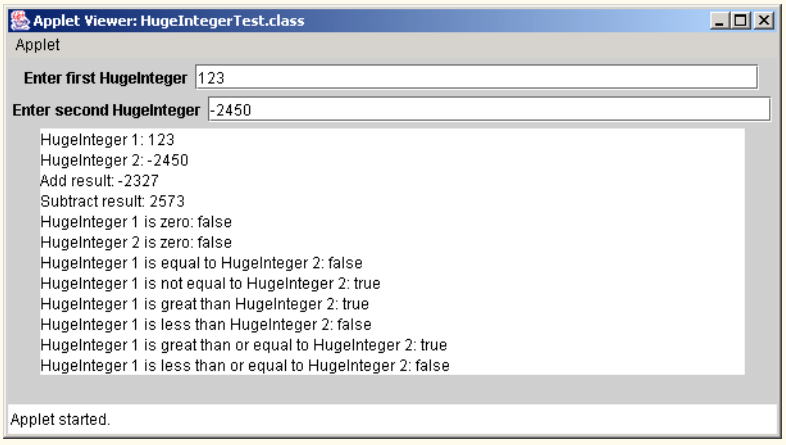
8.10 Create a class `HugeInteger` which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods `input`, `output`, `add` and `subtract`. For comparing `HugeInteger` objects, provide the following methods: `isEqualTo`, `isNotEqualTo`, `isGreaterThan`, `isLessThan`, `isGreaterThanOrEqualTo` and `isLessThanOrEqualTo`. Each of these is a predicate method that returns `true` if the relationship holds between the two `HugeInteger` objects and returns `false` if the relationship does not hold. Provide a predicate method `isZero`. If you feel ambitious, also provide methods `multiply`, `divide` and `remainder`.

ANS:

```
1  // Exercise 8.10 solution: HugeIntegerTest.java
2  // Test class HugeInteger
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class HugeIntegerTest extends JApplet implements ActionListener {
8     private JLabel firstLabel, secondLabel;
9     private JTextField firstField, secondField;
```

```
10      private JTextArea outputArea;
11
12      // set up GUI components
13      public void init()
14      {
15         firstLabel = new JLabel( "Enter first HugeInteger" );
16         secondLabel = new JLabel( "Enter second HugeInteger" );
17         firstField = new JTextField( 40 );
18         secondField = new JTextField( 40 );
19         secondField.addActionListener( this );
20         outputArea = new JTextArea( 12, 50 );
21
22         Container container = getContentPane();
23         container.setLayout( new FlowLayout() );
24         container.add( firstLabel );
25         container.add( firstField );
26         container.add( secondLabel );
27         container.add( secondField );
28         container.add( outputArea );
29
30      } // end constructor
31
32      public void actionPerformed( ActionEvent event )
33      {
34         HugeInteger integer1 = new HugeInteger();
35         HugeInteger integer2 = new HugeInteger();
36         integer1.input( firstField.getText() );
37         integer2.input( secondField.getText() );
38
39         // clear output area
40         outputArea.setText( "" );
41
42         String output = "HugeInteger 1: " + integer1.toHugeIntegerString() +
43            "\nHugeInteger 2: " + integer2.toHugeIntegerString();
44
45         HugeInteger result;
46
47         // add two HugeIntegers
48         result = integer1.add( integer2 );
49         output += "\nAdd result: " + result.toHugeIntegerString();
50
51         // subtract two HugeIntegers
52         result = integer1.subtract( integer2 );
53         output += "\nSubtract result: " + result.toHugeIntegerString();
54
55         // compare two HugeIntegers
56         output += "\nHugeInteger 1 is zero: " + integer1.isZero();
57         output += "\nHugeInteger 2 is zero: " + integer2.isZero();
58         output += "\nHugeInteger 1 is equal to HugeInteger 2: " +
59            integer1.isEqualTo( integer2 );
60         output += "\nHugeInteger 1 is not equal to HugeInteger 2: " +
61            integer1.isNotEqualTo( integer2 );
62         output += "\nHugeInteger 1 is great than HugeInteger 2: " +
63            integer1.isGreatThan( integer2 );
```

```
64          output += "\nHugeInteger 1 is less than HugeInteger 2: " +
65             integer1.isLessThan( integer2 );
66          output += "\nHugeInteger 1 is great than or equal to " +
67             "HugeInteger 2: " + integer1.isGreatThanOrEqualTo( integer2 );
68          output += "\nHugeInteger 1 is less than or equal to " +
69             "HugeInteger 2: " + integer1.isLessThanOrEqualTo( integer2 );
70
71          // append output string to outputArea
72          outputArea.append( output );
73
74       } // end method ActionPerformed
75
76    } // end class HugeIntegerTest
```




```
1  // Exercise 8.10 solution: HugeInteger.java
2  // HugeInteger class definition
3
4  public class HugeInteger {
5     private final int DIGITS = 40;
6     private int[] integer;
7     private boolean positive;
8
9     public HugeInteger()
10    {
11       integer = new int[ DIGITS ];
12       positive = true;
13    }
14
15    // convert a String to HugeInteger
16    public void input( String inputString )
17    {
18       char[] integerChar = inputString.toCharArray();
19
20       // check if input is a negative number
21       if ( integerChar[ 0 ] == '-' )
22          positive = false;
```

```
23
24          if ( positive )
25             integer[ DIGITS - integerChar.length ] =
26                ( int )integerChar[ 0 ] - ( int )'0';
27
28          // convert string to integer array
29          for ( int i = 1; i < integerChar.length; i++ )
30             integer[ DIGITS - integerChar.length + i ] =
31                ( int )integerChar[ i ] - ( int )'0';
32       }
33
34       // add two HugeIntegers
35       public HugeInteger add( HugeInteger addValue )
36       {
37          HugeInteger temp = new HugeInteger();   // temporary result
38
39          // both HugeInteger are positive or negative
40          if ( ( positive && addValue.positive ) ||
41             !( positive || addValue.positive ) )
42             temp = addPositives( addValue );
43
44          // addValue is negative
45          else if ( positive && ( !addValue.positive ) ) {
46             addValue.positive = true;
47
48             if ( isGreatThan( addValue ) )
49                temp = subtractPositives( addValue );
50             else {
51                temp = addValue.subtractPositives( this );
52                temp.positive = false;
53             }
54
55             addValue.positive = false;  // reset sign for addValue
56          }
57
58          // this is negative
59          else if ( !positive && addValue.positive ) {
60             addValue.positive = false;
61
62             if ( isGreatThan( addValue ) )
63                temp = addValue.subtractPositives( this );
64             else {
65                temp = subtractPositives( addValue );
66                temp.positive = false;
67             }
68
69             addValue.positive = true;  // reset sign for addValue
70          }
71
72          return temp;  // return the sum
73
74       } // end method add
75
```

```
76    // add two positive HugeIntegers
77    public HugeInteger addPositives( HugeInteger addValue )
78    {
79       HugeInteger temp = new HugeInteger();
80       int carry = 0;
81
82       // iterate through HugeInteger
83       for ( int i = 39; i >= 0; i-- ) {
84          temp.integer[ i ] =
85          integer[ i ] + addValue.integer[ i ] + carry;
86
87          // determine whether to carry a 1
88          if ( temp.integer[ i ] > 9 ) {
89             temp.integer[ i ] %= 10;  // reduce to 0-9
90             carry = 1;
91          }
92          else
93             carry = 0;
94       }
95
96       // if both are negative, set the result to negative
97       if ( !positive )
98          temp.positive = false;
99
100      return temp;
101
102   } // end method addPositives
103
104   // subtract two HugeIntegers
105   public HugeInteger subtract( HugeInteger subtractValue )
106   {
107      HugeInteger temp = new HugeInteger();   // temporary result
108
109      // subtractValue is negative
110      if ( positive && ( !subtractValue.positive ) )
111         temp = addPositives( subtractValue );
112
113      // this HugeInteger is negative
114      else if ( !positive && subtractValue.positive ) {
115         temp = addPositives( subtractValue );
116      }
117
118      // both HugeIntegers are positive
119      if ( positive && subtractValue.positive )
120         temp = subtractPositives( subtractValue );
121
122      // both HugeIntegers are negative
123      if ( !( positive || subtractValue.positive ) )
124         temp = subtractValue.subtractPositives( this );
125
126      return temp;
127
128   } // end method subtract
129
```

```
130    // subtract two positive HugeIntegers
131    public HugeInteger subtractPositives( HugeInteger subtractValue )
132    {
133       int borrow = 0;
134       HugeInteger temp = new HugeInteger();
135
136       // iterate through HugeInteger
137       for ( int i = 39; i >= 0; i-- ) {
138
139          // determine to add 10 to smaller integer
140          if ( integer[i] < subtractValue.integer[ i ] ){
141             temp.integer[ i ] = ( integer[ i ] + 10 ) -
142                subtractValue.integer[ i ] - borrow;
143             borrow = 1;      // set borrow to one
144          }
145
146          // if borrowing is not needed
147          else {
148             temp.integer[ i ] =
149             integer[ i ] - subtractValue.integer[ i ] - borrow;
150             borrow = 0;      // set borrow to zero
151          }
152
153       } // end for
154
155       return temp;  // return difference of two HugeIntegers
156
157    } // end method subtractPositives
158
159    // find first non-zero position of two HugeIntegers
160    public int findFirstNonZeroPosition()
161    {
162       int position = 39;
163
164       // find first non-zero position for first HugeInteger
165       for ( int i = 0; i < DIGITS; i++ ) {
166
167             if ( integer[ i ] > 0 ) {
168                position = i;
169                break;
170             }
171       }
172
173       return position;
174
175    } // end method findFirstNonZeroPosition
176
177    // get string representation of HugeInteger
178    public String toHugeIntegerString()
179    {
180       String sign = "", output = "";
181
182       if ( !positive )
183          sign = "-";
```

```
184
185       // get HugeInteger values without leading zeros
186       for ( int i = findFirstNonZeroPosition(); i < DIGITS; i++ )
187          output += integer[ i ];
188
189       return sign + output;
190
191    } // end method toHugeIntegerString
192
193    // test if two HugeIntegers are equal
194    public boolean isEqualTo( HugeInteger compareValue )
195    {
196       // compare each digit
197       for ( int i = 0; i < DIGITS; i++ ) {
198
199          if ( integer[ i ] != compareValue.integer[ i ] )
200             return false;
201       }
202
203       // if digits are same, compare the sign
204       if ( ( positive && compareValue.positive ) ||
205          !( positive || compareValue.positive ) )
206          return true;
207       else
208          return false;
209    }
210
211    // test if two HugeIntegers are not equal
212    public boolean isNotEqualTo( HugeInteger compareValue )
213    {
214       return !isEqualTo( compareValue );
215    }
216
217    // test if one HugeInteger is greater than another
218    public boolean isGreatThan( HugeInteger compareValue )
219    {
220       // different signs
221       if ( positive && ( !compareValue.positive ) )
222          return true;
223       else if ( !positive && compareValue.positive )
224          return false;
225
226       // same sign
227       else {
228
229          // first number's length is less than second number's length
230          if ( findFirstNonZeroPosition() >
231             compareValue.findFirstNonZeroPosition() ) {
232
233             if ( positive )
234                return false;
235             else
236                return true;
237          }
```

```
238
239          // first number's length is larger than that of second number
240          else if ( findFirstNonZeroPosition() <
241             compareValue.findFirstNonZeroPosition() ) {
242
243             if ( positive )
244                return true;
245             else
246                return false;
247          }
248
249          // two numbers have same length
250          else {
251             for ( int i = 0; i < DIGITS; i++ ) {
252
253                if ( integer[ i ] > compareValue.integer[ i ] )
254
255                   if ( positive )
256                      return true;
257                   else
258                      return false;
259             }
260
261             if ( positive )
262                return false;
263             else
264                return true;
265          }
266
267       } // end outer if-elseif-else
268
269    } // end method isGreatThan
270
271    // test if one HugeInteger is less than another
272    public boolean isLessThan( HugeInteger compareValue )
273    {
274       return !( isGreatThan( compareValue ) ||
275          isEqualTo( compareValue ) );
276    }
277
278    // test if one HugeInteger is great than or equal to another
279    public boolean isGreatThanOrEqualTo( HugeInteger compareValue )
280    {
281       return !isLessThan( compareValue );
282    }
283
284    // test if one HugeInteger is less than or equal to another
285    public boolean isLessThanOrEqualTo( HugeInteger compareValue )
286    {
287       return !isGreatThan( compareValue );
288    }
289
290    // test if one HugeInteger is zero
291    public boolean isZero()
292    {
```

```
293         // compare each digit
294         for ( int i = 0; i < DIGITS; i++ ) {
295
296            if ( integer[ i ] != 0 )
297               return false;
298         }
299
300         return true;
301      }
302
303   } // end class HugeInteger
```

8.11 Create a class `TicTacToe` that will enable you to write a complete program to play the game of Tic-Tac-Toe. The class contains a private 3-by-3 two-dimensional array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square; place a 2 wherever the second player moves. Each move must be to an empty square. After each move determine whether the game has been won and whether the game is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board [*Note:* This is a challenging project that could take many weeks of effort!].

ANS:

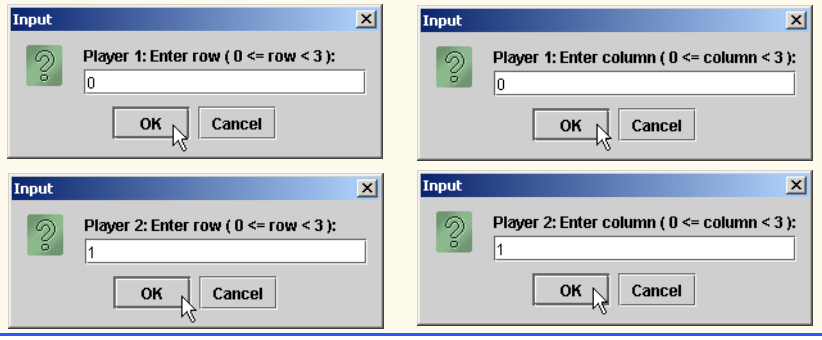
```
1  // Exercise 8.11 solution: TicTacToe.java
2  // Program that plays the game of tic-tac-toe.
3  import javax.swing.*;
4
5  public class TicTacToe {
6     private final int BOARDSIZE = 3;
7     private final int WIN = 1, DRAW = 2, CONTINUE = 3;
8     private int board[][];
9     private boolean firstPlayer, gameOver;
10
11    public TicTacToe()
12    {
13       board = new int[ BOARDSIZE ][ BOARDSIZE ];
14       firstPlayer = true;
15       gameOver = false;
16    }
17
18    // start game
19    public void makeMove()
20    {
21       int row = 0, column = 0;
22
23       while ( !gameOver ) {
24
25          // first player's turn
26          if ( firstPlayer ) {
27             row = Integer.parseInt( JOptionPane.showInputDialog(
28                "Player 1: Enter row ( 0 <= row < 3 ):" ) );
```

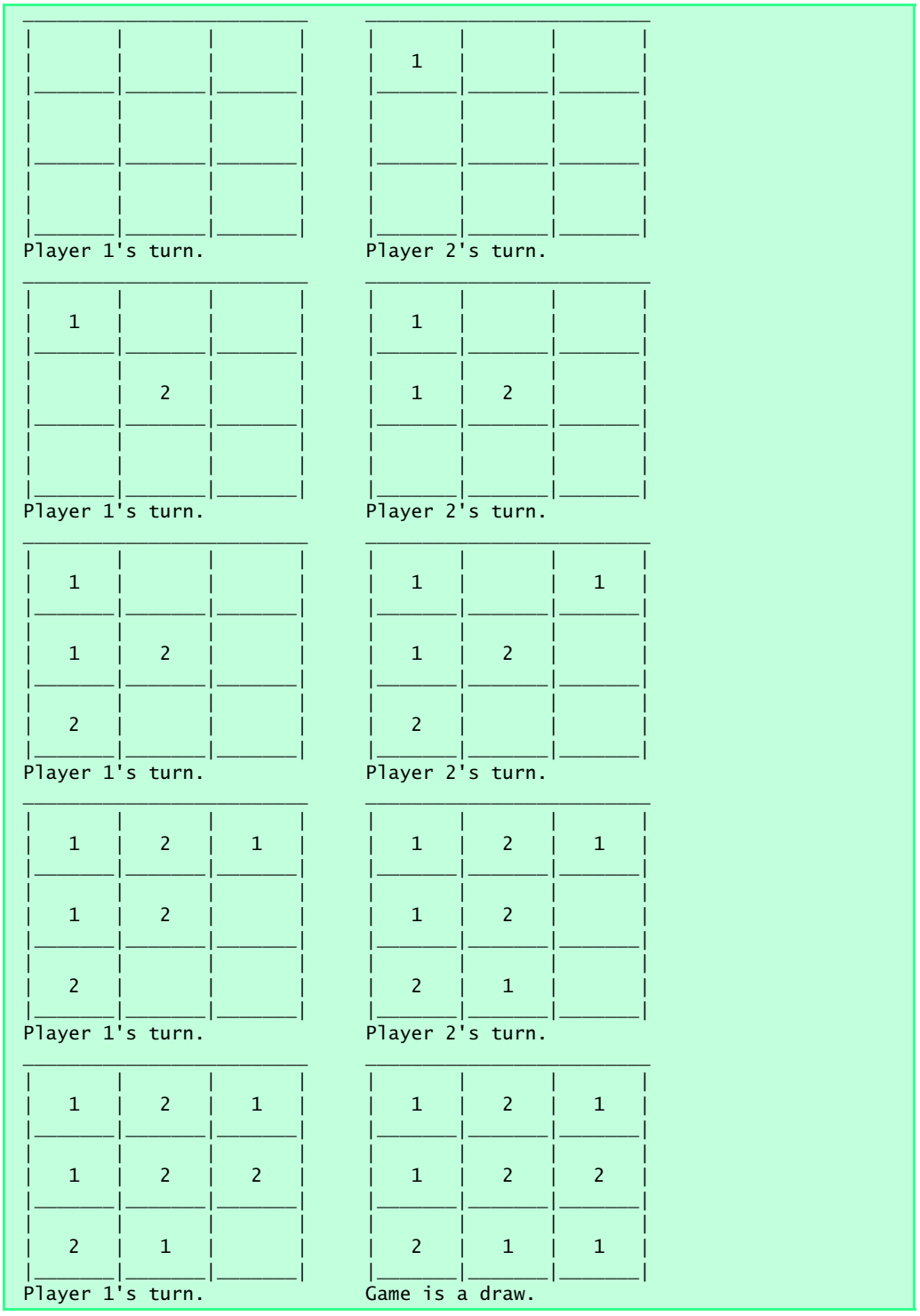
```
29             column = Integer.parseInt( JOptionPane.showInputDialog(
30                "Player 1: Enter column ( 0 <= column < 3 ):" ) );
31
32             // validate move
33             while ( !validMove( row, column ) ) {
34                row = Integer.parseInt( JOptionPane.showInputDialog(
35                   "Player 1: Enter row ( 0 <= row < 3 ):" ) );
36                column = Integer.parseInt( JOptionPane.showInputDialog(
37                   "Player 1: Enter column ( 0 <= column < 3 ):" ) );
38             }
39
40             firstPlayer = false;
41             board[ row ][ column ] = 1;
42             printBoard();
43
44             printStatus( 1 );
45
46          } // end first player's turn
47
48          // second player's turn
49          else {
50             row = Integer.parseInt( JOptionPane.showInputDialog(
51                "Player 2: Enter row ( 0 <= row < 3 ):" ) );
52             column = Integer.parseInt( JOptionPane.showInputDialog(
53                "Player 2: Enter column ( 0 <= column < 3 ):" ) );
54
55             // validate move
56             while ( !validMove( row, column ) ) {
57                row = Integer.parseInt( JOptionPane.showInputDialog(
58                   "Player 2: Enter row ( 0 <= row < 3 ):" ) );
59                column = Integer.parseInt( JOptionPane.showInputDialog(
60                   "Player 2: Enter column ( 0 <= column < 3 ):" ) );
61             }
62
63             firstPlayer = true;
64             board[ row ][ column ] = 2;
65             printBoard();
66
67             printStatus( 2 );
68
69          } // end seond player's turn
70
71       } // end while
72
73    } // end method makeMove
74
75    // show game status in status bar
76    public void printStatus( int player )
77    {
78       int status = gameStatus();
79
80       // check game status
81       switch ( status ) {
82
```

```
83             case WIN:
84                System.out.println( "Player " + player + " wins." );
85                gameOver = true;
86                break;
87
88             case DRAW:
89                System.out.println( "Game is a draw." );
90                gameOver = true;
91                break;
92
93             case CONTINUE:
94                if ( player == 1 )
95                   System.out.println( "Player 2's turn." );
96                else
97                   System.out.println( "Player 1's turn." );
98                break;
99
100            } // end switch
101
102         } // end method printStatus
103
104         // get game status
105         public int gameStatus()
106         {
107            int a;
108
109            // check for a win on diagonals
110            if ( board[ 0 ][ 0 ] != 0 && board[ 0 ][ 0 ] == board[ 1 ][ 1 ] &&
111               board[ 0 ][ 0 ] == board[ 2 ][ 2 ] )
112               return WIN;
113            else if ( board[ 2 ][ 0 ] != 0 && board[ 2 ][ 0 ] ==
114               board[ 1 ][ 1 ] && board[ 2 ][ 0 ] == board[ 0 ][ 2 ] )
115               return WIN;
116
117            // check for win in rows
118            for ( a = 0; a < 3; ++a )
119               if ( board[ a ][ 0 ] != 0 && board[ a ][ 0 ] ==
120               board[ a ][ 1 ] && board[ a ][ 0 ] == board[ a ][ 2 ] )
121                  return WIN;
122
123            // check for win in columns
124            for ( a = 0; a < 3; ++a )
125               if ( board[ 0 ][ a ] != 0 && board[ 0 ][ a ] ==
126                  board[ 1 ][ a ] && board[ 0 ][ a ] == board[ 2 ][ a ] )
127                  return WIN;
128
129            // check for a completed game
130            for ( int r = 0; r < 3; ++r )
131               for ( int c = 0; c < 3; ++c )
132                  if ( board[ r ][ c ] == 0 )
133                     return CONTINUE;  // game is not finished
134
135            return DRAW;  // game is a draw
136
137         } // end method gameStatus
```

```
138
139    // display board
140    public void printBoard()
141    {
142       System.out.println( "_____________________________" );
143
144       for ( int row = 0; row < BOARDSIZE; ++row ) {
145          System.out.println( "|       |       |       |" );
146
147          for ( int column = 0; column < BOARDSIZE; ++column )
148             printSymbol( column, board[ row ][ column ] );
149
150          System.out.println( "|_______|_______|_______|" );
151       }
152
153    } // end method printBoard
154
155    // print moves
156    public void printSymbol( int column, int player )
157    {
158       String output = "";
159
160       if ( column != 2 ) {  // first two columns
161
162          switch ( player ) {
163             case 0:
164                output = "|       ";
165                break;
166             case 1:
167                output = "|   1   ";
168                break;
169             case 2:
170                output = "|   2   ";
171                break;
172          }
173       }
174       else {  // last column
175
176          switch ( player ) {
177             case 0:
178                output = "|       |\n";
179                break;
180             case 1:
181                output = "|   1   |\n";
182                break;
183             case 2:
184                output = "|   2   |\n";
185                break;
186          }
187       }
188
189       System.out.print( output );
190
191    } // end method printSymbol
```

```
192
193    // validate move
194    public boolean validMove( int row, int column )
195    {
196       return row >= 0 && row < 3 && column >= 0 && column < 3 &&
197          board[ row ][ column ] == 0;
198    }
199
200    // main method
201    public static void main( String[] args )
202    {
203       TicTacToe game = new TicTacToe();
204       game.printBoard();
205       System.out.println( "Player 1's turn." );
206       game.makeMove();
207       System.exit( 0 );
208    }
209
210 } // end class TicTacToe
```





8.12 Explain the notion of package access in Java. Explain the negative aspects of package access.
ANS: Package access allows a class, method, or variable to be accessible within the same package. Package access does not promote good OOP when applied to an instance variable, because it destroys the notion of information hiding.

8.13 What happens when a return type, even `void`, is specified for a constructor?
ANS: It is treated as a method and is not considered to be a constructor.

8.14 Create class `Date` with the following capabilities:

a) Output the date in multiple formats such as

```
MM/DD/YYYY
June 14, 1992
DDD YYYY
```

b) Use overloaded constructors to create `Date` objects initialized with dates of the formats in part (a). In the first case, constructor should receive three integer values. In the second case the constructor should receive a `String` and two integer values. In the third case the constructor should receive two integer values, the first of which represents the day number in the year. [*Hint:* To convert the string representation of the month to a numeric value, compare strings using the `equals` method. For example, if `s1` and `s2` are strings, the method call `s1.equals( s2 )` returns `true` of the strings are identical; otherwise, the method call returns `false`.]

ANS:

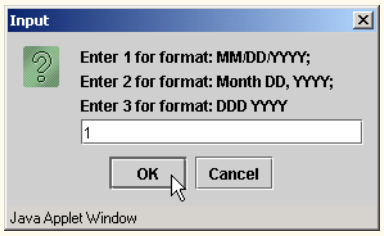
```
1  // Exercise 8.14 solution: DateTest.java
2  // Program that tests Date class
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DateTest extends JApplet implements ActionListener {
8     private JLabel mmLabel, ddLabel1, yyyyLabel1;
9     private JLabel monthLabel, ddLabel2, yyyyLabel2;
10    private JLabel dddLabel, yyyyLabel3;
11    private JTextField mmField, ddField1, yyyyField1;
12    private JTextField monthField, ddField2, yyyyField2;
13    private JTextField dddField, yyyyField3;
14    private JButton resetChoiceButton;
15    private int choice;
16    private Date date;
17
18    // set up GUI components
19    public void init()
20    {
21       // get user input format
22       getChoice();
23
24       // GUI for format: MM/DD/YYYY
25       mmLabel = new JLabel( "Enter MM:" );
26       ddLabel1 = new JLabel( "Enter DD:" );
27       yyyyLabel1 = new JLabel( "Enter YYYY:" );
28       mmField = new JTextField( 10 );
29       ddField1 = new JTextField( 10 );
```

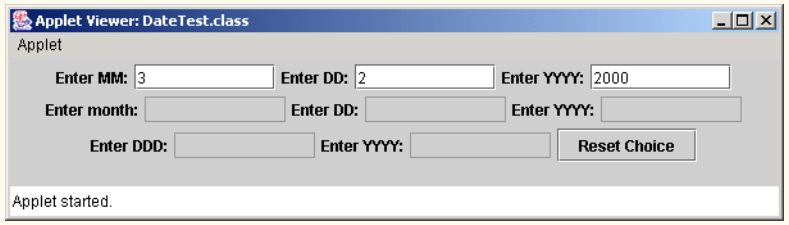
```
30          yyyyField1 = new JTextField( 10 );
31          yyyyField1.addActionListener( this );
32
33          // GUI for format: Month DD, YYYY
34          monthLabel = new JLabel( "Enter month:" );
35          ddLabel2 = new JLabel( "Enter DD:" );
36          yyyyLabel2 = new JLabel( "Enter YYYY:" );
37          monthField = new JTextField( 10 );
38          ddField2 = new JTextField( 10 );
39          yyyyField2 = new JTextField( 10 );
40          yyyyField2.addActionListener( this );
41
42          // GUI for format: DDD YYYY
43          dddLabel = new JLabel( "Enter DDD:" );
44          yyyyLabel3 = new JLabel( "Enter YYYY:" );
45          dddField = new JTextField( 10 );
46          yyyyField3 = new JTextField( 10 );
47          yyyyField3.addActionListener( this );
48
49          setEditable();
50
51          // button to reset choice
52          resetChoiceButton = new JButton( "Reset Choice" );
53          resetChoiceButton.addActionListener( this );
54
55          Container container = getContentPane();
56          container.setLayout( new FlowLayout() );
57
58          // add all components together
59          container.add( mmLabel );
60          container.add( mmField );
61          container.add( ddLabel1 );
62          container.add( ddField1 );
63          container.add( yyyyLabel1 );
64          container.add( yyyyField1 );
65          container.add( monthLabel );
66          container.add( monthField );
67          container.add( ddLabel2 );
68          container.add( ddField2 );
69          container.add( yyyyLabel2 );
70          container.add( yyyyField2 );
71          container.add( dddLabel );
72          container.add( dddField );
73          container.add( yyyyLabel3 );
74          container.add( yyyyField3 );
75          container.add( resetChoiceButton );
76
77       } // end method init
78
79       // get user choice
80       public void getChoice()
81       {
82          do {
83             choice = Integer.parseInt( JOptionPane.showInputDialog(
```

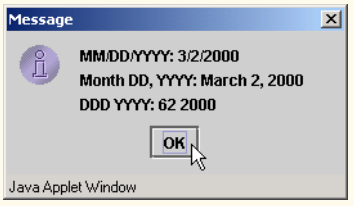
```
84                "Enter 1 for format: MM/DD/YYYY;\n" +
85                "Enter 2 for format: Month DD, YYYY;\n" +
86                "Enter 3 for format: DDD YYYY" ) );
87          } while ( !( choice == 1 || choice == 2 || choice == 3 ) );
88       }
89
90       // set JTextField editable based on user choice
91       public void setEditable()
92       {
93          switch ( choice ) {
94
95             case 1:  // format: MM/DD/YYYY
96                setFieldEditable(
97                   true, true, true, false, false, false, false, false );
98                break;
99
100            case 2:  // format: Month DD, YYYY
101               setFieldEditable(
102                  false, false, false, true, true, true, false, false );
103               break;
104
105            case 3:  // format: DDD YYYY
106               setFieldEditable(
107                  false, false, false, false, false, false, true, true );
108               break;
109
110         } // end switch
111
112      } // end method setFieldEditable
113
114      // set all JTextFields
115      public void setFieldEditable( boolean mm, boolean dd1,
116         boolean yyyy1, boolean month, boolean dd2, boolean yyyy2,
117         boolean ddd, boolean yyyy3 )
118      {
119         mmField.setEditable( mm );
120         ddField1.setEditable( dd1 );
121         yyyyField1.setEditable( yyyy1 );
122         monthField.setEditable( month );
123         ddField2.setEditable( dd2 );
124         yyyyField2.setEditable( yyyy2 );
125         dddField.setEditable( ddd );
126         yyyyField3.setEditable( yyyy3 );
127      }
128
129      // handle events
130      public void actionPerformed( ActionEvent event )
131      {
132         int mm, dd, yyyy, ddd;
133         String month;
134
135         // handle MM/DD/YYYY format
136         if ( event.getSource() == yyyyField1 ) {
137            mm = Integer.parseInt( mmField.getText() );
```

```
138          dd = Integer.parseInt( ddField1.getText() );
139          yyyy = Integer.parseInt( yyyyField1.getText() );
140
141          mmField.setText( "" );
142          ddField1.setText( "" );
143          yyyyField1.setText( "" );
144
145          date = new Date( mm, dd, yyyy );
146          showOutput( date );
147       }
148
149       // handle Month DD, YYYY format
150       else if ( event.getSource() == yyyyField2 ) {
151          month = monthField.getText();
152          dd = Integer.parseInt( ddField2.getText() );
153          yyyy = Integer.parseInt( yyyyField2.getText() );
154
155          monthField.setText( "" );
156          ddField2.setText( "" );
157          yyyyField2.setText( "" );
158
159          date = new Date( month, dd, yyyy );
160          showOutput( date );
161       }
162
163       // handle DDD YYYY format
164       else if ( event.getSource() == yyyyField3 ) {
165          ddd = Integer.parseInt( dddField.getText() );
166          yyyy = Integer.parseInt( yyyyField3.getText() );
167
168          dddField.setText( "" );
169          yyyyField3.setText( "" );
170
171          date = new Date( ddd, yyyy );
172          showOutput( date );
173       }
174
175       // handle resetChoicebutton
176       else {
177          getChoice();
178          setEditable();
179       }
180
181    } // end method actionPerformed
182
183    // display output in a message dialog
184    public void showOutput( Date displayDate )
185    {
186       JOptionPane.showMessageDialog( null, "MM/DD/YYYY: " +
187          displayDate.toSlashDateString() + "\nMonth DD, YYYY: " +
188          displayDate.toMonthNameDateString() + "\nDDD YYYY: " +
189          displayDate.toDayDateString() );
190    }
191
```

```
192 } // end class DateTest
```








```
1  // Exercise 8.14 solution: Date.java
2  // Date class definition
3
4  public class Date {
5     private int day, month, year;
6     private final String[] monthNames = { "January", "February",
7        "March", "April", "May", "June", "July", "August",
8        "September", "October", "November", "December" };
9     private final int monthDays[] = { 31, 28, 31, 30, 31, 30,
10       31, 31, 30, 31, 30, 31 };
11
12    // no-argument constructor
13    public Date()
14    {
15       day = 1;
16       month = 1;
17       year = 2000;
18    }
19
20    // constructor for format MM/DD/YYYY
21    public Date( int mm, int dd, int yyyy )
22    {
23       setMonth( mm );
24       setDay( dd );
25       setYear( yyyy );
26    }
```

```
27
28    // constructor for format MonthName dd, yyyy
29    public Date( String mm, int dd, int yyyy )
30    {
31       convertFromMonthName( mm );
32       setDay( dd );
33       setYear( yyyy );
34    }
35
36    // constructor for format DDD YYYY
37    public Date( int ddd, int yyyy )
38    {
39       convertFromDayOfYear( ddd );
40       setYear( yyyy );
41    }
42
43    // Set the day
44    public void setDay( int dd )
45    {
46       day = ( dd >= 1 && dd <= daysOfMonth() ) ? dd : 1;
47    }
48
49    // Set the month
50    public void setMonth( int mm )
51    {
52       month = ( mm >= 1 && mm <= 12 ) ? mm : 1;
53    }
54
55    // Set the year
56    public void setYear( int yyyy )
57    {
58       year = ( yyyy >= 1900 && yyyy <= 2100 ) ? yyyy : 1900;
59    }
60
61    // return Date in format: mm/dd/yyyy
62    public String toSlashDateString()
63    {
64       return month + "/" + day + "/" + year;
65    }
66
67    // return Date in format: MonthName dd, yyyy
68    public String toMonthNameDateString()
69    {
70       return monthNames[ month - 1 ] + " " + day + ", " + year;
71    }
72
73    // return Date in format DDD YYYY
74    public String toDayDateString()
75    {
76       return convertToDayOfYear() + " " + year;
77    }
78
79    // Return the number of days in the month
80    public int daysOfMonth()
81    {
```

```
82          return leapYear() && month == 2 ? 29 : monthDays[ month - 1 ];
83       }
84
85       // test for a leap year
86       public boolean leapYear()
87       {
88          if ( year % 400 == 0 || ( year % 4 == 0 && year % 100 != 0 ) )
89             return true;
90          else
91             return false;
92       }
93
94       // convert ddd to mm and dd
95       public void convertFromDayOfYear( int ddd )
96       {
97          int dayTotal = 0;
98
99          if ( ddd < 1 || ddd > 366 )  // check for invalid day
100            ddd = 1;
101
102         setMonth( 1 );
103
104         for ( int m = 1;
105            m < 13 && ( dayTotal + daysOfMonth() ) < ddd; ++m ) {
106            dayTotal += daysOfMonth();
107            setMonth( m + 1 );
108         }
109
110         setDay( ddd - dayTotal );
111      }
112
113      // convert mm and dd to ddd
114      public int convertToDayOfYear()
115      {
116         int ddd = 0;
117
118         for ( int m = 1; m < month; ++m ) {
119
120            if ( leapYear() && m == 2 )
121               ddd += 29;
122            else
123               ddd += monthDays[ m -1 ];
124         }
125
126         ddd += day;
127         return ddd;
128      }
129
130      // convert from month name to month number
131      public void convertFromMonthName( String monthName )
132      {
133         boolean flag = false;
134
135         for ( int subscript = 0; subscript < 12; ++subscript )
```

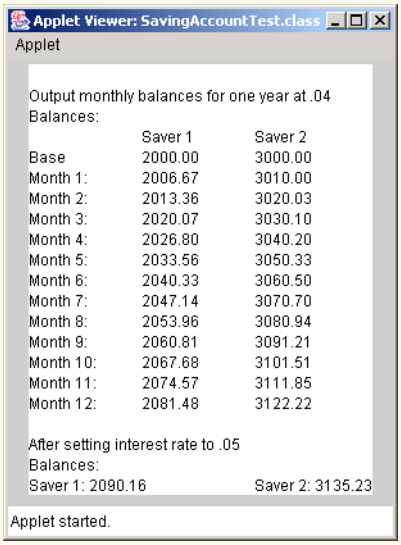
```
136
137          if ( monthName.equals( monthNames[ subscript ] ) ) {
138             setMonth( subscript + 1 );
139             flag = true;  // set flag
140             break;    // stop checking for month
141          }
142
143       if ( !flag )
144          setMonth( 1 ); // invalid month default is january
145    }
146
147 } // end class Date
```

8.15 Create class `SavingsAccount`. Use a `static` variable `annualInterestRate` to store the annual interest rate for all account holders. Each object of the class contains a `private` instance variable `savingsBalance` indicating the amount the saver currently has on deposit. Provide method `calculateMonthlyInterest` to calculate the monthly interest by multiplying the `savingsBalance` by `annualInterestRate` divided by 12; this interest should be added to `savingsBalance`. Provide a `static` method `modifyInterestRate` that sets the `annualInterestRate` to a new value. Write a program to test class `SavingsAccount`. Instantiate two `savingsAccount` objects, `saver1` and `saver2`, with balances of $2000.00 and $3000.00, respectively. Set `annualInterestRate` to 4%, then calculate the monthly interest and print the new balances for both savers. Then set the `annualInterestRate` to 5%, calculate the next month's interest and print the new balances for both savers.

ANS:

```
1  // Exercise 8.15 solution: SavingAccountTest.java
2  // Program that tests SavingAccount class
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class SavingAccountTest extends JApplet {
7     private SavingAccount saver1, saver2;
8     private JTextArea outputArea;
9     private String output;
10
11    // set up GUI
12    public void init()
13    {
14       output = "";
15       outputArea = new JTextArea( 20, 15 );
16
17       Container container = getContentPane();
18       container.setLayout( new FlowLayout() );
19       container.add( outputArea );
20    }
21
22    public void start()
23    {
24       saver1 = new SavingAccount( 2000 );
25       saver2 = new SavingAccount( 3000 );
26       SavingAccount.modifyInterestRate( 0.04 );
```

```
27
28          output += "\nOutput monthly balances for one year at .04" +
29             "\nBalances: \n\tSaver 1 " + "\tSaver 2\nBase\t" +
30             saver1.toSavingAccountString() + "\t" +
31             saver2.toSavingAccountString();
32
33          for ( int month = 1; month <= 12; ++month ) {
34             saver1.calculateMonthlyInterest();
35             saver2.calculateMonthlyInterest();
36
37             output += "\nMonth " +  month + ":\t" +
38                saver1.toSavingAccountString() + "\t" +
39                saver2.toSavingAccountString();
40          }
41
42          SavingAccount.modifyInterestRate( .05 );
43          saver1.calculateMonthlyInterest();
44          saver2.calculateMonthlyInterest();
45          output += "\n\nAfter setting interest rate to .05" +
46             "\nBalances: \nSaver 1: " + saver1.toSavingAccountString() +
47             "\tSaver 2: " + saver2.toSavingAccountString();
48
49          outputArea.append( output );
50
51       } // end method start
52
53    } // end class SavingAccountTest
```




```
1  // Exercise 8.15 solution: SavingAccount
2  // SavingAccount class definition
3  import java.text.DecimalFormat;
4
```

```
 5  public class SavingAccount {
 6     private double savingsBalance;
 7     private static double annualInterestRate = 0;
 8
 9     // constructor
10     public SavingAccount( double balance )
11     {
12        savingsBalance = balance;
13     }
14
15     // get monthly interest
16     public void calculateMonthlyInterest()
17     {
18        savingsBalance += savingsBalance * ( annualInterestRate / 12.0 );
19     }
20
21     // modify interest rate
22     public static void modifyInterestRate( double newRate )
23     {
24        annualInterestRate =
25           ( newRate >= 0 && newRate <= 1.0 ) ? newRate : 0.04;
26     }
27
28     // get string representation of SavingAccount
29     public String toSavingAccountString()
30     {
31        DecimalFormat twoDigits = new DecimalFormat( "0.00" );
32
33        return twoDigits.format( savingsBalance );
34     }
35
36  } // end class SavingAccount
```

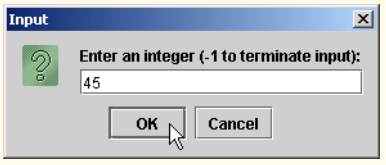
8.16 Create class `IntegerSet`. Each `IntegerSet` object can hold integers in the range 0–100. The set is represented by an array of `boolean`s. Array element `a[i]` is `true` if integer *i* is in the set. Array element `a[j]` is `false` if integer *j* is not in the set. The no-argument constructor initializes the Java array to the "empty set" (i.e., a set whose array representation contains all `false` values).

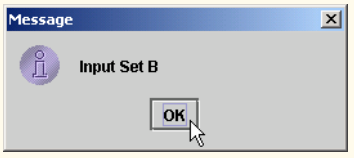
Provide the following methods: Method `union` creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to `true` if that element is `true` in either or both of the existing sets; otherwise, the element of the third set is set to `false`). Method `intersection` creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to `false` if that element is `false` in either or both of the existing sets; otherwise, the element of the third set is set to `true`). Method `insertElement` inserts a new integer *k* into a set (by setting `a[k]` to `true`). Method `deleteElement` deletes integer *m* (by setting `a[m]` to `false`). Method `toSetString` returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use `---` to represent an empty set. Method `isEqualTo` determines whether two sets are equal. Write a program to test class `IntegerSet`. Instantiate several `IntegerSet` objects. Test that all your methods work properly.

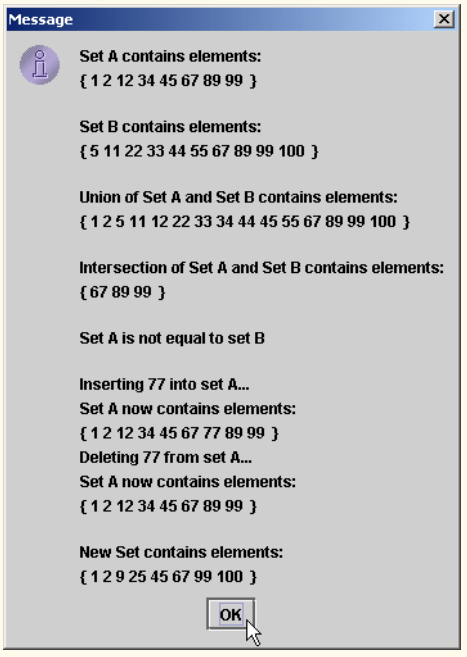
ANS:

```
// Exercise 8.16 solution: IntegerSetTest.java
// Program that tests IntegerSet
import javax.swing.*;

public class IntegerSetTest {
   private IntegerSet set1, set2, set3, set4;
   private String output;

   // no-argument constructor
   public IntegerSetTest()
   {
      set1 = new IntegerSet();
      set2 = new IntegerSet();
      output = "";
   }

   public void start()
   {
      // initialize two sets
      JOptionPane.showMessageDialog( null, "Input Set A" );
      set1.inputSet();
      JOptionPane.showMessageDialog( null, "Input Set B" );
      set2.inputSet();

      // get union and intersection of two sets
      set3 = set1.union( set2 );
      set4 = set1.intersection( set2 );

      // prepare output
      output += "Set A contains elements:\n" + set1.toSetString() +
         "\n\nSet B contains elements:\n" + set2.toSetString() +
         "\n\nUnion of Set A and Set B contains elements:\n" +
         set3.toSetString() + "\n\nIntersection of Set A and Set B" +
         " contains elements:\n" + set4.toSetString();

      // test whether two sets are equal
      if ( set1.isEqualTo( set2 ) )
         output += "\n\nSet A is equal to set B\n";
      else
         output += "\n\nSet A is not equal to set B\n";

      // test insert and delete
      output += "\nInserting 77 into set A...\n";
      set1.insertElement( 77 );
      output += "Set A now contains elements:\n" + set1.toSetString();

      output += "\nDeleting 77 from set A...\n";
      set1.deleteElement( 77 );
      output += "Set A now contains elements:\n" + set1.toSetString();

      // test constructor
      int intArray[] = { 25, 67, 2, 9, 99, 105, 45, -5, 100, 1 };
```

```
53          IntegerSet set5 = new IntegerSet( intArray );
54
55          output += "\n\nNew Set contains elements:\n" + set5.toSetString();
56
57          JOptionPane.showMessageDialog( null, output );
58
59       } // end method start
60
61       public static void main( String[] args )
62       {
63          IntegerSetTest test = new IntegerSetTest();
64          test.start();
65          System.exit( 0 );
66       }
67
68    } // end class IntegerSetTest
```








```
1  // Exercise 8.16 solution: IntegerSet.java
2  // IntegerSet class definition
3  import javax.swing.*;
4
5  public class IntegerSet {
6     private final int SETSIZE = 101;
7     private boolean[] set;
8
9     // no-argument constructor
10    public IntegerSet()
11    {
```

```
12          set = new boolean[ SETSIZE ];
13       }
14
15       // constructor creates a set from array of integers
16       public IntegerSet( int array[] )
17       {
18          set = new boolean[ SETSIZE ];
19
20          for( int i = 0; i < array.length; i++ )
21             insertElement( array[ i ] );
22       }
23
24       // input a set from the user
25       public void inputSet()
26       {
27          int number;
28
29          // get a set of integers from user
30          do {
31             number = Integer.parseInt( JOptionPane.showInputDialog(
32                "Enter an integer (-1 to terminate input):" ) );
33
34             if ( validEntry( number ) )
35                set[ number ] = true;
36
37          } while ( number != -1 );
38       }
39
40       // return string representation of set
41       public String toSetString()
42       {
43          int x = 1;
44          boolean empty = true;  // assume set is empty
45          String setString = "{ ";
46
47          // get set elements
48          for ( int count = 0; count < 101; ++count ) {
49             if ( set[ count ] ) {
50                setString += count + " ";
51                empty = false;  // set is not empty
52                ++x;
53             }
54          }
55
56          // empty set
57          if ( empty )
58             setString += "---";  // display an empty set
59
60          setString += " }";
61
62          return setString;
63
64       } // end method toSetString
65
```

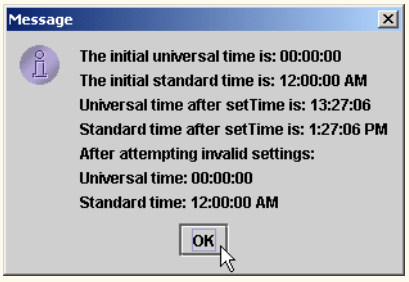
```
66    // returns the union of two sets
67    public IntegerSet union( IntegerSet integerSet )
68    {
69       IntegerSet temp = new IntegerSet();
70
71       for ( int count = 0; count < 101; ++count )
72          if ( set[ count ] || integerSet.set[ count ] )
73             temp.set[ count ] = true;
74
75       return temp;
76    }
77
78    // returns the intersection of two sets
79    public IntegerSet intersection( IntegerSet integerSet )
80    {
81       IntegerSet temp = new IntegerSet();
82
83       for ( int count = 0; count < 101; ++count )
84          if ( set[ count ] && integerSet.set[ count ] )
85             temp.set[ count ] = true;
86
87       return temp;
88    }
89
90    // insert a new integer into this set
91    public void insertElement( int insertInteger )
92    {
93       if ( validEntry( insertInteger ) )
94          set[ insertInteger ] = true;
95    }
96
97    // remove an integer from this set
98    public void deleteElement( int deleteInteger )
99    {
100      if ( validEntry( deleteInteger ) )
101         set[ deleteInteger ] = false;
102   }
103
104   // determine if two sets are equal
105   public boolean isEqualTo( IntegerSet integerSet )
106   {
107      for ( int count = 0; count < 101; ++count )
108         if ( set[ count ] != integerSet.set[ count ] )
109            return false;   // sets are not-equal
110
111      return true;   // sets are equal
112   }
113
114   // determin if input is valid
115   public boolean validEntry( int input )
116   {
117      return input >= 0 && input <= 100;
118   }
119
```

```
120 } // end class IntegerSet
```

8.17 It would be perfectly reasonable for the `Time3` class of Fig. 8.7 to represent the time internally as the number of seconds since midnight rather than the three integer values `hour`, `minute` and `second`. Clients could use the same `public` methods and get the same results. Modify the `Time3` class of Fig. 8.7 to implement the `Time3` as the number of seconds since midnight and show that there is no change visible to the clients of the class.

ANS:

```
1  // Exercise 8.17 Solution: TimeTest3.java
2  // Class TimeTest3 to test class Time3
3  import javax.swing.*;
4
5  public class TimeTest3 {
6     public static void main( String args[] )
7     {
8        Time3 t = new Time3();
9        String result = "";
10
11       result += "The initial universal time is: " + t.toUniversalString();
12       result += "\nThe initial standard time is: " +
13          t.toStandardString();
14
15       t.setTime( 13, 27, 6 );
16       result += "\nUniversal time after setTime is: " +
17          t.toUniversalString();
18       result += "\nStandard time after setTime is: " +
19          t.toStandardString();
20
21       t.setTime( 99, 99, 99 );
22       result += "\nAfter attempting invalid settings:";
23       result += "\nUniversal time: " + t.toUniversalString();
24       result += "\nStandard time: " + t.toStandardString();
25
26       JOptionPane.showMessageDialog( null, result );
27       System.exit( 0 );
28
29    } // end method main
30
31 } // end class TimeTest3
```



```
1  // Exercise 8.17 solution: Time3.java
2  // Time3 class definition maintains the time in 24-hour format.
3  import java.text.DecimalFormat;
4
5  public class Time3 {
6     private int totalSeconds;
7
8     // no-argument constructor initializes totalSeconds to zero;
9     public Time3()
10    {
11       totalSeconds = 0;
12    }
13
14    // Time3 constructor: hour supplied, minute and second defaulted to 0
15    public Time3( int h )
16    {
17       setTime( h, 0, 0 );
18    }
19
20    // Time3 constructor: hour and minute supplied, second defaulted to 0
21    public Time3( int h, int m )
22    {
23       setTime( h, m, 0 );
24    }
25
26    // Time3 constructor: hour, minute and second supplied
27    public Time3( int h, int m, int s )
28    {
29       setTime( h, m, s );
30    }
31
32    // Time3 constructor: another Time3 object supplied
33    public Time3( Time3 time )
34    {
35       setTime( time.getHour(), time.getMinute(), time.getSecond() );
36    }
37
38    // set a new time value using total seconds; perform
39    // validity checks on data; set invalid values to zero
40    public void setTime( int h, int m, int s )
41    {
42       setHour( h );
43       setMinute( m );
44       setSecond( s );
45    }
46
47    // set hour value
48    public void setHour( int h )
49    {
50       int hours = ( h >= 0 && h < 24 ) ? h : 0;
51
52       totalSeconds = ( hours * 3600 ) + ( getMinute() * 60 ) +
53          getSecond();
54    } // end function setHour
```

```
55
56    // set minute value
57    public void setMinute( int m )
58    {
59       int minutes = ( m >= 0 && m < 60 ) ? m : 0;
60
61       totalSeconds = ( getHour() * 3600 ) + ( minutes * 60 ) +
62          getSecond();
63    } // end function setMinute
64
65    // set second value
66    public void setSecond( int s )
67    {
68       int seconds = ( s >= 0 && s < 60 ) ? s : 0;
69
70       totalSeconds = ( getHour() * 3600 ) + ( getMinute() *60 ) + seconds;
71
72    } // end function setSecond
73
74    // get hour value
75    public int getHour()
76    {
77       return ( totalSeconds / 3600 );
78
79    } // end function getHour
80
81    // get minute value
82    public int getMinute()
83    {
84       return ( ( totalSeconds % 3600 ) / 60 );
85
86    } // end function getMinute
87
88    // get second value
89    public int getSecond()
90    {
91       return ( ( totalSeconds % 3600 ) % 60 );
92
93    } // end function getSecond
94
95    // convert to String in universal-time format
96    public String toUniversalString()
97    {
98       DecimalFormat twoDigits = new DecimalFormat( "00" );
99       int hour, minute, second, temp;
100
101      hour = totalSeconds / 3600;
102      temp = totalSeconds % 3600;
103      minute = temp / 60;
104      second = temp % 60;
105
106      return twoDigits.format( hour ) + ":" +  twoDigits.format(
107         minute ) + ":" + twoDigits.format( second );
108   }
```

```
109
110    // convert to String in standard-time format
111    public String toStandardString()
112    {
113       DecimalFormat twoDigits = new DecimalFormat( "00" );
114       int hour, minute, second, temp;
115
116       hour = totalSeconds / 3600;
117       temp = totalSeconds % 3600;
118       minute = temp / 60;
119       second = temp % 60;
120
121       return ( ( hour == 12 || hour == 0 ) ? 12 : hour % 12 ) +
122          ":" + twoDigits.format( minute ) + ":" +
123          twoDigits.format( second ) + ( hour < 12 ? " AM" : " PM" );
124    }
125
126 } // end class Time3
```

8.18 *(Drawing Program)* Create a drawing applet that randomly draws lines, rectangles and ovals. For this purpose, create a set of "smart" shape classes where objects of these classes know how to draw themselves if provided with a `Graphics` object that tells them where to draw (i.e., the applet's `Graphics` object allows a shape to draw on the applet's background). The class names should be `MyLine`, `MyRect` and `MyOval`.

The data for class `MyLine` should include *x1*, *y1*, *x2* and *y2* coordinates. Method `drawLine` of class `Graphics` will connect the two points supplied with a line. The data for classes `MyRect` and `MyOval` should include an upper-left *x*-coordinate value, an upper-left *y*-coordinate value, a *width* (must be nonnegative) and a *height* (must be nonnegative). All data in each class must be `private`.

In addition to the data, each class should declare at least the following `public` methods:

a) A constructor with no arguments that sets the coordinates to 0.
b) A constructor with arguments that sets the coordinates to the supplied values.
c) *Set* methods for each individual piece of data that allow the programmer to set any piece of data in a shape independently (e.g., if you have an instance variable `x1`, you should have a method `setX1`).
d) *Get* methods for each individual piece of data that allow the programmer to retrieve any piece of data in a shape independently (e.g., if you have an instance variable `x1`, you should have a method `getX1`).
e) A `draw` method with the first line

```
public void draw( Graphics g )
```

that will be called from the applet's `paint` method to draw a shape onto the screen.

If you would like to provide more methods for flexibility, please do so.

Begin by declaring class `MyLine` and an applet to test your classes. The applet should have a `MyLine` instance variable `line` that can refer to one `MyLine` object (created in the applet's `init` method with random coordinates). The applet's `paint` method should draw the shape with a statement like

```
line.draw( g );
```

where `line` is the `MyLine` reference and `g` is the `Graphics` object that the shape will use to draw itself on the applet.

Next, change the single `MyLine` reference into an array of `MyLine` references and hard code several `MyLine` objects into the program for drawing. The applet's `paint` method should walk through the array of `MyLine` objects and draw every one.

After the preceding part is working, you should declare the `MyOval` and `MyRect` classes and add objects of these classes into the `MyRect` and `MyOval` arrays. The applet's `paint` method should walk through each array and draw every shape. Create five shapes of each type.

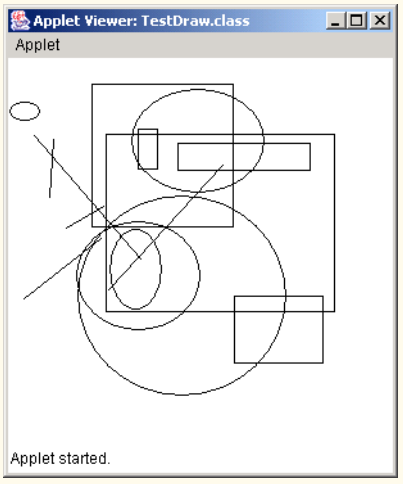
Once the applet is running, select **Reload** from the `appletviewer`'s **Applet** menu to reload the applet. This will cause the applet to choose new random numbers for the shapes and draw the shapes again.[1]

ANS:

```
1  // Exercise 8.18 Solution: TestDraw.java
2  // Program that tests classes MyLine, MyOval and MyRect
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class TestDraw extends JApplet {
7     private MyLine line[];
8     private MyOval oval[];
9     private MyRect rectangle[];
10
11    // initialize arrays which hold five of each shape
12    public void init()
13    {
14       line = new MyLine[ 5 ];
15       oval = new MyOval[ 5 ];
16       rectangle = new MyRect[ 5 ];
17
18       for ( int count = 0; count < line.length; count++ ) {
19          int x1 = (int) ( Math.random() * 400 );
20          int y1 = (int) ( Math.random() * 400 );
21          int x2 = (int) ( Math.random() * 400 );
22          int y2 = (int) ( Math.random() * 400 );
23          line[ count] = new MyLine( x1, y1, x2, y2 );
24       }
25
26       for ( int count = 0; count < oval.length; count++ ) {
27          int x = (int) ( Math.random() * 400 );
28          int y = (int) ( Math.random() * 400 );
29          int theLength = (int) ( Math.random() * 400 );
30          int theWidth = (int) ( Math.random() * 400 );
31          oval[ count ] = new MyOval( x, y, theLength, theWidth );
32       }
33
34       for ( int count = 0; count < rectangle.length; count++ ) {
35          int x = (int) ( Math.random() * 400 );
36          int y = (int) ( Math.random() * 400 );
37          int theLength = (int) ( Math.random() * 400 );
38          int theWidth = (int) ( Math.random() * 400 );
```

1. In Chapter 10, we will modify this exercise to take advantage of the similarities between the classes and to avoid reinventing the wheel.

```
39             rectangle[ count ] = new MyRect( x, y, theLength, theWidth );
40          }
41
42       } // end method init
43
44       // for each shape array, create shapes with random values,
45       // then draw them.
46       public void paint( Graphics g )
47       {
48          for ( int count = 0; count < line.length; count++ ) {
49             line[ count ].draw( g );
50          }
51
52          for ( int count = 0; count < oval.length; count++ ) {
53             oval[ count ].draw( g );
54          }
55
56          for ( int count = 0; count < rectangle.length; count++ ) {
57             rectangle[ count ].draw( g );
58          }
59
60       }  // end method paint
61
62    }  // end method paint
```




```
1  // Exercise 8.18 Solution: MyLine.java
2  // Definition of class MyLine
3  import java.awt.Graphics;
4
5  public class MyLine {
6     private int x1, x2, y1, y2;
7
8     // constructor initializes private vars with
9     // default values
```

```
10    public MyLine()
11    {
12       x1 = 0;
13       y1 = 0;
14       x2 = 0;
15       y2 = 0;
16    }
17
18    // constructor with input values
19    public MyLine( int x1, int y1, int x2, int y2 )
20    {
21       setX1( x1 );
22       setX2( x2 );
23       setY1( y1 );
24       setY2( y2 );
25    }
26
27    // accessor and mutator methods for each of the
28    // four private variables:
29    public void setX1( int x1 )
30    {
31      this.x1 = ( x1 >= 0 ? x1 : 0 );
32    }
33
34    public int getX1()
35    {
36      return x1;
37    }
38
39    public void setX2( int x2 )
40    {
41      this.x2 = ( x2 >= 0 ? x2 : 0 );
42    }
43
44    public int getX2()
45    {
46      return x2;
47    }
48
49    public void setY1( int y1 )
50    {
51      this.y1 = ( y1 >= 0 ? y1 : 0 );
52    }
53
54    public int getY1()
55    {
56      return y1;
57    }
58
59    public void setY2( int y2 )
60    {
61      this.y2 = ( y2 >= 0 ? y2 : 0 );
62    }
63
```

```
64     public int getY2()
65     {
66       return y2;
67     }
68
69     // Actually draws the line
70     public void draw( Graphics g )
71     {
72        g.drawLine( x1, y1, x2, y2 );
73     }
74
75  }  // end class MyLine
```

```
 1  // Exercise 8.18 Solution: MyRect.java
 2  // Definition of class MyRect
 3  import java.awt.Graphics;
 4
 5  public class MyRect {
 6     private int length, width, upperLeftX, upperLeftY;
 7
 8     // constructor initializes private vars with
 9     // default values
10     public MyRect()
11     {
12        length =  0;
13        width = 0;
14        upperLeftX = 0;
15        upperLeftY = 0;
16     }
17
18     // constructor
19     public MyRect( int x, int y, int theLength, int theWidth )
20     {
21        setUpperLeftX( x );
22        setUpperLeftY( y );
23        setLength( theLength );
24        setWidth( theWidth );
25     }
26
27     // accessor and mutator methods for each of the
28     // four private variables:
29     public void setUpperLeftX( int x )
30     {
31       upperLeftX = x;
32     }
33
34     public int getUpperLeftX()
35     {
36       return upperLeftX;
37     }
38
39     public void setUpperLeftY( int y )
40     {
```

```
41        upperLeftY = y;
42     }
43
44     public int getUpperLeftY()
45     {
46        return upperLeftY;
47     }
48
49     public void setWidth( int theWidth )
50     {
51        width = ( theWidth >= 0 ? theWidth : 1 );
52     }
53
54     public int getWidth()
55     {
56        return width;
57     }
58
59     public void setLength( int theLength )
60     {
61        length = ( theLength >= 0.0 ? theLength : 1 );
62     }
63
64     public int getLength()
65     {
66        return length;
67     }
68
69     // Actually draws the rectangle
70     public void draw( Graphics g )
71     {
72        g.drawRect( upperLeftX, upperLeftY, length, width );
73     }
74
75  }  // end class MyRect
```

```
1  // Exercise 8.18 Solution: MyOval.java
2  // Definition of class MyRect
3  import java.awt.Graphics;
4
5  public class MyOval {
6     private int length, width, upperLeftX, upperLeftY;
7
8     // constructor initializes private vars with
9     // default values
10    public MyOval()
11    {
12       length =  0;
13       width = 0;
14       upperLeftX = 0;
15       upperLeftY = 0;
16    }
17
```

```
18    // constructor with input values
19    public MyOval( int x, int y, int theLength, int theWidth )
20    {
21       setUpperLeftX( x );
22       setUpperLeftY( y );
23       setLength( theLength );
24       setWidth( theWidth );
25    }
26
27    // accessor and mutator methods for each of the
28    // four private variables:
29    public void setUpperLeftX( int x )
30    {
31      upperLeftX = x;
32    }
33
34    public int getUpperLeftX()
35    {
36      return upperLeftX;
37    }
38
39    public void setUpperLeftY( int y )
40    {
41      upperLeftY = y;
42    }
43
44    public int getUpperLeftY()
45    {
46      return upperLeftY;
47    }
48
49    public void setWidth( int theWidth )
50    {
51      width = ( theWidth >= 0 ? theWidth : 0 );
52    }
53
54    public int getWidth()
55    {
56      return width;
57    }
58
59    public void setLength( int theLength )
60    {
61      length = ( theLength >= 0 ? theLength : 0 );
62    }
63
64    public int getLength()
65    {
66      return length;
67    }
68
69    // Actually draws the oval
70    public void draw( Graphics g )
71    {
```

```
72          g.drawOval( upperLeftX, upperLeftY, length, width );
73       }
74
75    }  // end class MyOval
```

Object-Oriented Programming

Objectives

- To understand how inheritance promotes software reusability.
- To understand the notions of superclasses and subclasses.
- To understand access modifier `protected`.
- To be able to access superclass members with `super`.
- To understand the use of constructors and finalizers in inheritance hierarchies.
- To present a case study that demonstrates the mechanics of inheritance.

Say not you know another entirely, till you have divided an inheritance with him.
Johann Kasper Lavater

This method is to define as the number of a class the class of all classes similar to the given class.
Bertrand Russell

Good as it is to inherit a library, it is better to collect one.
Augustine Birrell

SELF-REVIEW EXERCISES

9.1 Fill in the blanks in each of the following statements:

a) _____ is a form of software reusability in which new classes acquire the data and behaviors of existing classes and embellish those classes with new capabilities.

ANS: Inheritance

b) A superclass's _____ members can be accessed only in the superclass declaration or in inherited by subclasses.

ANS: `protected`

c) In a(n) _____ relationship, an object of a subclass also can be treated as an object of its superclass.

ANS: "is-a" or inheritance

d) In a(n) _____ relationship, a class object has references to objects of other classes as members.

ANS: "has-a" or composition or aggregation

e) In single inheritance, a class exists in a(n) _____ relationship with its subclasses.

ANS: hierarchical

f) A superclass's _____ members are accessible anywhere that the program has a reference to an object of that superclass or to an object of one of its subclasses.

ANS: `public`

g) A superclass's `protected` access members have a level of protection between those of _____ and _____ access.

ANS: `public`, `private`

h) When an object of a subclass is instantiated, a superclass _____ is called implicitly or explicitly.

ANS: constructor

i) Subclass constructors can call superclass constructors via the _____ keyword.

ANS: `super`

9.2 State whether each of the following statements is *true* or *false*. If *false*, explain why.

a) It is possible to treat superclass objects and subclass objects similarly.

ANS: True.

b) Superclass constructors are not inherited by subclasses.

ANS: True.

c) A "has-a" relationship is implemented via inheritance.

ANS: False. A "has-a" relationship is implemented via composition. An "is-a" relationship is implemented via inheritance.

d) A `Car` class has an "is a" relationship with its `SteeringWheel` and `Brakes`.

ANS: False. This is an example of a "has–a" relationship. Class `Car` has an "is–a" relationship with class `Vehicle`.

e) Inheritance encourages the reuse of proven high-quality software.

ANS: True.

f) When a subclass redefines a superclass method by using the same signature, the subclass is said to overload that superclass method.

ANS: False. This is known as overriding, not overloading.

EXERCISES

9.3 Many programs written with inheritance could be written with composition instead, and vice versa. Rewrite classes `Circle4` (Fig. 9.13) and `Cylinder` (Fig. 9.15) of the `Point3/Circle4/Cylinder` hierarchy to use composition rather than inheritance. After you do this, assess the relative merits of the two approaches for the `Point3`, `Circle4`, and `Cylinder` problems, as well as for object-oriented programs in general. Which approach is more natural? Why?

ANS: For a relatively short program like this one, either approach is acceptable. But as programs become larger with more and more objects being instantiated, inheritance becomes preferable because it makes the program easier to modify and promotes the reuse of code.

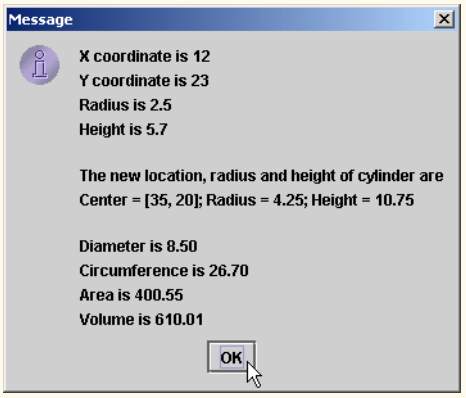
```
1  // Exercise 9.3 solution: Circle4.java
2  // Definition of class Circle4.
3
4  public class Circle4 {
5     private double radius;  // Circle4's radius
6     private Point3 point;   // composition
7
8     // no-argument constructor
9     public Circle4()
10    {
11       point = new Point3( 0, 0 );
12       setRadius( 0 );
13    }
14
15    // constructor
16    public Circle4( int xValue, int yValue, double radiusValue )
17    {
18       // instantiate point object
19       point = new Point3( xValue, yValue );
20       setRadius( radiusValue );
21    }
22
23    // set radius
24    public void setRadius( double radiusValue )
25    {
26       radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
27    }
28
29    // return radius
30    public double getRadius()
31    {
32       return radius;
33    }
34
35    // set x
36    public void setX( int x )
37    {
38       point.setX( x );
39    }
40
41    // return x
42    public int getX()
43    {
```

```
44          return point.getX();
45       }
46
47       // set y
48       public void setY( int y )
49       {
50          point.setY( y );
51       }
52
53       // return y
54       public int getY()
55       {
56          return point.getY();
57       }
58
59       // calculate and return diameter
60       public double getDiameter()
61       {
62          return 2 * getRadius();
63       }
64
65       // calculate and return circumference
66       public double getCircumference()
67       {
68          return Math.PI * getDiameter();
69       }
70
71       // calculate and return area
72       public double getArea()
73       {
74          return Math.PI * getRadius() * getRadius();
75       }
76
77       // return String representation of Circle4 object
78       public String toString()
79       {
80          return "Center = " + point.toString() + "; Radius = " + getRadius();
81       }
82
83    } // end class Circle4
```

```
1  // Exercise 9.3 solution: Cylinder.java
2  // Cylinder class definition.
3
4  public class Cylinder {
5     private double height;  // Cylinder's height
6     private Circle4 circle;
7
8     // no-argument constructor
9     public Cylinder()
10    {
11       circle = new Circle4( 0, 0, 0 );
12       setHeight( 0 );
```

```
13    }
14
15    // constructor
16    public Cylinder( int xValue, int yValue, double radiusValue,
17       double heightValue )
18    {
19       circle = new Circle4( xValue, yValue, radiusValue );
20       setHeight( heightValue );
21    }
22
23    // set Cylinder's height
24    public void setHeight( double heightValue )
25    {
26       height = ( heightValue < 0.0 ? 0.0 : heightValue );
27    }
28
29    // get Cylinder's height
30    public double getHeight()
31    {
32       return height;
33    }
34
35    // set x
36    public void setX( int x )
37    {
38       circle.setX( x );
39    }
40
41    // return x
42    public int getX()
43    {
44       return circle.getX();
45    }
46
47    // set y
48    public void setY( int y )
49    {
50       circle.setY( y );
51    }
52
53    // return y
54    public int getY()
55    {
56       return circle.getY();
57    }
58
59    // set radius
60    public void setRadius( double radiusValue )
61    {
62       circle.setRadius( radiusValue );
63    }
64
65    // return radius
66    public double getRadius()
67    {
```

```
68          return circle.getRadius();
69       }
70
71       // return diameter
72       public double getDiameter()
73       {
74          return circle.getDiameter();
75       }
76
77       // return circumference
78       public double getCircumference()
79       {
80          return circle.getCircumference();
81       }
82
83       // calculate Cylinder area
84       public double getArea()
85       {
86          return 2 * circle.getArea() +
87             circle.getCircumference() * getHeight();
88       }
89
90       // calculate Cylinder volume
91       public double getVolume()
92       {
93          return  circle.getArea() * getHeight();
94       }
95
96       // return String representation of Cylinder object
97       public String toString()
98       {
99          return  circle.toString() + "; Height = " + getHeight();
100      }
101
102   } // end class Cylinder
```



9.4 Some programmers prefer not to use `protected` access, because they believe it breaks the encapsulation of the superclass. Discuss the relative merits of using `protected` access vs. using `private` access in superclasses.

ANS: Inherited private data is hidden in the subclass and is accessible only through the public or protected methods of the superclass. Using protected access enables the subclass to manipulate the protected members without using the access methods of the superclass. If the superclass members are private, the methods of the superclass must be used to access the data. This may result in a decrease in performance due to the extra method calls.

9.5 Rewrite the case study of Section 9.5 as a `Point`–`Square`–`Cube` hierarchy. Do this two ways—once via inheritance and once via composition.

ANS: Composition Solution:

```
1  // Exercise 9.5 solution: Point.java
2  // Point class definition represents an x-y coordinate pair.
3
4  public class Point {
5     private int x;  // x part of coordinate pair
6     private int y;  // y part of coordinate pair
7
8     // no-argument constructor
9     public Point()
10    {
11       // implicit call to Object constructor occurs here
12    }
13
14    // constructor
15    public Point( int xValue, int yValue )
16    {
17       // implicit call to Object constructor occurs here
18       x = xValue;  // no need for validation
19       y = yValue;  // no need for validation
20    }
21
22    // set x in coordinate pair
23    public void setX( int xValue )
24    {
25       x = xValue;  // no need for validation
26    }
27
28    // return x from coordinate pair
29    public int getX()
30    {
31       return x;
32    }
33
34    // set y in coordinate pair
35    public void setY( int yValue )
36    {
37       y = yValue;  // no need for validation
38    }
39
```

```
40     // return y from coordinate pair
41     public int getY()
42     {
43        return y;
44     }
45
46     // return String representation of Point3 object
47     public String toString()
48     {
49        return "[" + getX() + ", " + getY() + "]";
50     }
51
52  } // end class Point
```

```
1   // Exercise 9.5 solution: Square.java
2   // Definition of class Square.
3
4   public class Square {
5      private double sideLength;   // Square's side length
6      private Point point;   // composition
7
8      // no-argument constructor
9      public Square()
10     {
11        point = new Point( 0, 0 );
12        setSideLength( 0 );
13     }
14
15     // constructor
16     public Square( int xValue, int yValue, double sidelength )
17     {
18        // instantiate point object
19        point = new Point( xValue, yValue );
20        setSideLength( sidelength );
21     }
22
23     // set sideLength
24     public void setSideLength( double sidelength )
25     {
26        sideLength = ( sidelength < 0.0 ? 0.0 : sidelength );
27     }
28
29     // return sideLength
30     public double getSideLength()
31     {
32        return sideLength;
33     }
34
35     // set x
36     public void setX( int x )
37     {
38        point.setX( x );
39     }
```

```
40
41    // return x
42    public int getX()
43    {
44       return point.getX();
45    }
46
47    // set y
48    public void setY( int y )
49    {
50       point.setY( y );
51    }
52
53    // return y
54    public int getY()
55    {
56       return point.getY();
57    }
58
59    // calculate and return circumference
60    public double getCircumference()
61    {
62       return 4 * getSideLength();
63    }
64
65    // calculate and return area
66    public double getArea()
67    {
68       return getSideLength() * getSideLength();
69    }
70
71    // return String representation of Square object
72    public String toString()
73    {
74       return "Up-left point = " + point.toString() + "; Side = " +
75          getSideLength();
76    }
77
78 } // end class Square
```

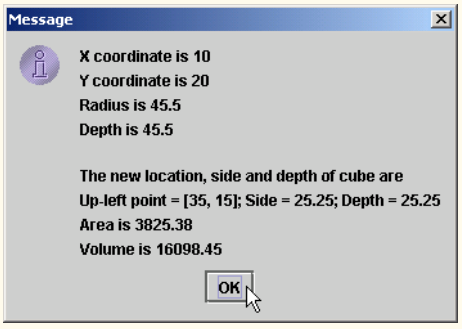
```
1  // Exercise 9.5 solution: Cube.java
2  // Cube class definition.
3
4  public class Cube {
5     private double depth;  // Cube's depth
6     private Square square;
7
8     // no-argument constructor
9     public Cube()
10    {
11       square = new Square( 0, 0, 0 );
12       depth = 0;
13    }
```

```
14
15    // constructor
16    public Cube( int xValue, int yValue, double sideValue )
17    {
18       square = new Square( xValue, yValue, sideValue );
19       setDepth( sideValue );
20    }
21
22    // set Cube's depth
23    public void setDepth( double depthValue )
24    {
25       depth = ( depthValue < 0.0 ? 0.0 : depthValue );
26    }
27
28    // get Cube's depth
29    public double getDepth()
30    {
31       return depth;
32    }
33
34    // set x
35    public void setX( int x )
36    {
37       square.setX( x );
38    }
39
40    // return x
41    public int getX()
42    {
43       return square.getX();
44    }
45
46    // set y
47    public void setY( int y )
48    {
49       square.setY( y );
50    }
51
52    // return y
53    public int getY()
54    {
55       return square.getY();
56    }
57
58    // set side length
59    public void setSideLength( double lengthValue )
60    {
61       square.setSideLength( lengthValue );
62    }
63
64    // return side length
65    public double getSideLength()
66    {
67       return square.getSideLength();
68    }
```

```
69
70     // calculate Cube area
71     public double getArea()
72     {
73        return 6 * square.getArea();
74     }
75
76     // calculate Cube volume
77     public double getVolume()
78     {
79        return square.getArea() * getDepth();
80     }
81
82     // return String representation of Cube object
83     public String toString()
84     {
85        return square.toString() + "; Depth = " + getDepth();
86     }
87
88  } // end class Cube
```

```
1  // Exercise 9.5 solution: CubeTest.java
2  // Testing class Cube.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class CubeTest {
7
8     public static void main( String[] args )
9     {
10       // create Cube object
11       Cube cube = new Cube( 10, 20, 45.5 );
12
13       // get Cube's initial x-y coordinates, side and depth
14       String output = "X coordinate is " + cube.getX() +
15          "\nY coordinate is " + cube.getY() + "\nRadius is " +
16          cube.getSideLength() + "\nDepth is " + cube.getDepth();
17
18       cube.setX( 35 );          // set new x-coordinate
19       cube.setY( 15 );          // set new y-coordinate
20       cube.setSideLength( 25.25 );   // set new side length
21       cube.setDepth( 25.25 );   // set new depth
22
23       // get String representation of new cube value
24       output += "\n\nThe new location, side and depth of cube are\n" +
25          cube.toString();
26
27       // format floating-point values with 2 digits of precision
28       DecimalFormat twoDigits = new DecimalFormat( "0.00" );
29
30       // get Cube's area
31       output += "\nArea is " + twoDigits.format( cube.getArea() );
32
```

```
33         // get Cube's volume
34         output += "\nVolume is " + twoDigits.format( cube.getVolume() );
35
36         JOptionPane.showMessageDialog( null, output ); // display output
37
38         System.exit( 0 );
39
40      } // end main
41
42   } // end class CubeTest
```



ANS: Inheritance Solution: Classes **`Point`** and **`CubeTest`** are the same as in composition solution.

```
1  // Exercise 9.5 solution: Square.java
2  // Definition of class Square.
3
4  public class Square extends Point {
5     private double sideLength;   // Square's side length
6
7     // no-argument constructor
8     public Square()
9     {
10       // implicit call to Point constructor occurs here
11    }
12
13    // constructor
14    public Square( int xValue, int yValue, double sidelength )
15    {
16       // instantiate point object
17       super( 0, 0 );
18       setSideLength( sidelength );
19    }
20
21    // set sideLength
22    public void setSideLength( double sidelength )
23    {
24       sideLength = ( sidelength < 0.0 ? 0.0 : sidelength );
25    }
```

```
26
27     // return sideLength
28     public double getSideLength()
29     {
30        return sideLength;
31     }
32
33     // calculate and return circumference
34     public double getCircumference()
35     {
36        return 4 * getSideLength();
37     }
38
39     // calculate and return area
40     public double getArea()
41     {
42        return getSideLength() * getSideLength();
43     }
44
45     // return String representation of Square object
46     public String toString()
47     {
48        return "Up-left point = " + super.toString() + "; Side = " +
49           getSideLength();
50     }
51
52  } // end class square
```

```
1   // Exercise 9.5 solution: Cube.java
2   // Cube class definition.
3
4   public class Cube extends Square{
5      private double depth;  // Cube's depth
6
7      // no-argument constructor
8      public Cube()
9      {
10        // implicit call to Square constructor occurs here
11     }
12
13     // constructor
14     public Cube( int xValue, int yValue, double sideValue )
15     {
16        super( xValue, yValue, sideValue );
17        setDepth( sideValue );
18     }
19
20     // set Cube's depth
21     public void setDepth( double depthValue )
22     {
23        depth = ( depthValue < 0.0 ? 0.0 : depthValue );
24     }
25
```

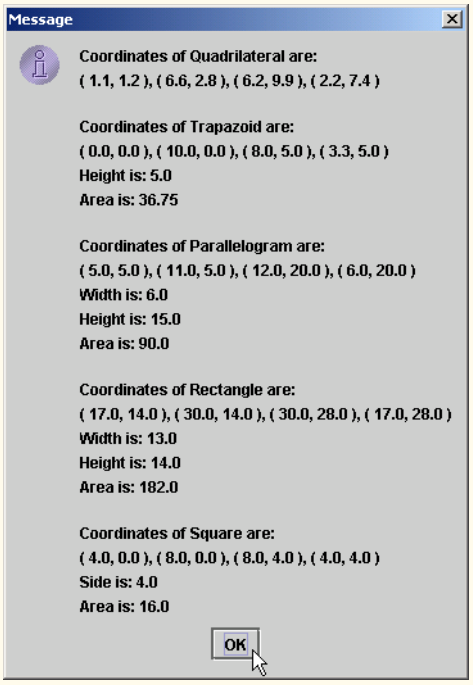
```
26     // get Cube's depth
27     public double getDepth()
28     {
29        return depth;
30     }
31
32     // calculate Cube area
33     public double getArea()
34     {
35        return 6 * super.getArea();
36     }
37
38     // calculate Cube volume
39     public double getVolume()
40     {
41        return  super.getArea() * getDepth();
42     }
43
44     // return String representation of Cube object
45     public String toString()
46     {
47        return super.toString() + "; Depth = " + getDepth();
48     }
49
50  } // end class Cube
```

9.6 Write an inheritance hierarchy for class `Quadrilateral`, `Trapezoid`, `Parallelogram`, `Rectangle` and `Square`. Use `Quadrilateral` as the superclass of the hierarchy. Make the hierarchy as deep (i.e., as many levels) as possible. Specify the instance variables and methods for each class. The `private` data of `Quadrilateral` should be the *x*-*y* coordinate pairs for the four endpoints of the `Quadrilateral`. Write a program that instantiates objects of your classes and outputs each object's area (except `Quadrilateral`).

ANS:

```
1  // Exercise 9.6 Solution: QuadrilateralTest.java
2  // driver for exercise 9.6
3  import javax.swing.*;
4
5  public class QuadrilateralTest {
6
7     public static void main( String[] args )
8     {
9        // NOTE: All coordinates are assumed to form the proper shapes
10       // A quadrilateral is a four-sided polygon
11       Quadrilateral quadrilateral = new Quadrilateral(
12          1.1, 1.2, 6.6, 2.8, 6.2, 9.9, 2.2, 7.4 );
13
14       // A trapazoid is a quadrilateral having two and only two
15       // parallel sides
16       Trapazoid trapazoid = new Trapazoid(
17          0.0, 0.0, 10.0, 0.0, 8.0, 5.0, 3.3, 5.0 );
18
```

```
19         // A parallelogram is a quadrilateral whose opposite sides are
20         // parallel
21         Parallelogram parallelogram = new Parallelogram(
22            5.0, 5.0, 11.0, 5.0, 12.0, 20.0, 6.0, 20.0 );
23
24         // A rectangle is an equiangular parallelogram
25         Rectangle rectangle = new Rectangle(
26            17.0, 14.0, 30.0, 14.0, 30.0, 28.0, 17.0, 28.0 );
27
28         // A square is an equiangular and equilateral parallelogram
29         Square square = new Square(
30            4.0, 0.0, 8.0, 0.0, 8.0, 4.0, 4.0, 4.0 );
31
32         String result = quadrilateral.toString() + "\n" +
33            trapazoid.toString() + "\n" + parallelogram.toString() +
34            "\n" + rectangle.toString() + "\n" + square.toString();
35
36         JOptionPane.showMessageDialog( null, result );
37         System.exit( 0 );
38
39      } // end method main
40
41   } // end class QuadrilateralTest
```




```
1  // Exercise 9.6 solution: Point.java
2  // Class Point definition
3
```

```
4  public class Point {
5     private double x, y;
6
7     public Point( double xCoordinate, double yCoordinate )
8     {
9        x = xCoordinate;
10       y = yCoordinate;
11    }
12
13    public double getX()
14    {
15       return x;
16    }
17
18    public double getY()
19    {
20       return y;
21    }
22
23    public String toString()
24    {
25       return "( " + getX() + ", " + getY() + " )";
26    }
27
28 } // end class Point
```

```
1  // Exercise 9.6 solution: Quadrilateral.java
2  // Class Quadrilateral definition
3
4  public class Quadrilateral {
5     Point point1, point2, point3, point4;
6
7     public Quadrilateral( double x1, double y1, double x2, double y2,
8        double x3, double y3, double x4, double y4 )
9     {
10       point1 = new Point( x1, y1 );
11       point2 = new Point( x2, y2 );
12       point3 = new Point( x3, y3 );
13       point4 = new Point( x4, y4 );
14    }
15
16    public Point getPoint1()
17    {
18       return point1;
19    }
20
21    public Point getPoint2()
22    {
23       return point2;
24    }
25
26    public Point getPoint3()
27    {
```

```
28          return point3;
29       }
30
31       public Point getPoint4()
32       {
33          return point4;
34       }
35
36       public String toString()
37       {
38          return "Coordinates of Quadrilateral are: \n" + printCoordinates();
39       }
40
41       public String printCoordinates()
42       {
43          return point1.toString() + ", " + point2.toString() + ", " +
44             point3.toString() + ", " + point4.toString();
45       }
46
47    } // end class Quadrilateral
```

```
1   // Exercise 9.6 solution: Trapazoid.java
2   // Class Trapazoid definition
3
4   public class Trapazoid extends Quadrilateral {
5      double height;
6
7      public Trapazoid( double x1, double y1, double x2, double y2,
8         double x3, double y3, double x4, double y4 )
9      {
10        super( x1, y1, x2, y2, x3, y3, x4, y4 );
11     }
12
13     public double getHeight()
14     {
15        if ( getPoint1().getY() == getPoint2().getY() )
16           return Math.abs( getPoint2().getY() - getPoint3().getY() );
17        else
18           return Math.abs( getPoint1().getY() - getPoint2().getY() );
19     }
20
21     public double getArea()
22     {
23        return getSumOfTwoSides() * getHeight() / 2.0;
24     }
25
26     public double getSumOfTwoSides()
27     {
28        if ( getPoint1().getY() == getPoint2().getY() )
29           return Math.abs( getPoint1().getX() - getPoint2().getX() ) +
30              Math.abs( getPoint3().getX() - getPoint4().getX() );
31        else
32           return Math.abs( getPoint2().getX() - getPoint3().getX() ) +
```

```
33              Math.abs( getPoint4().getX() - getPoint1().getX() );
34     }
35
36     public String toString()
37     {
38        return "\nCoordinates of Trapazoid are: \n" + printCoordinates() +
39           "\nHeight is: " + getHeight() + "\nArea is: " + getArea();
40     }
41
42  } // end class Trapazoid
```

```
1   // Exercise 9.6 solution: Parallelogram.java
2   // Class Parallelogram definition
3
4   public class Parallelogram extends Trapazoid {
5
6      public Parallelogram( double x1, double y1, double x2, double y2,
7         double x3, double y3, double x4, double y4 )
8      {
9         super( x1, y1, x2, y2, x3, y3, x4, y4 );
10     }
11
12     public double getWidth()
13     {
14        if ( getPoint1().getY() == getPoint2().getY() )
15           return Math.abs( getPoint1().getX() - getPoint2().getX() );
16        else
17           return Math.abs( getPoint2().getX() - getPoint3().getX() );
18     }
19
20     public String toString()
21     {
22        return "\nCoordinates of Parallelogram are: \n" +
23           printCoordinates() + "\nWidth is: " + getWidth() +
24           "\nHeight is: " + getHeight() + "\nArea is: " + getArea();
25     }
26
27  } // end class Parallelogram
```

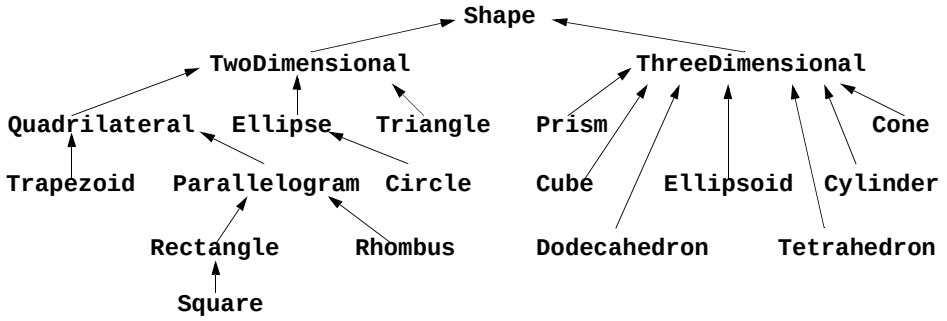
```
1   // Exercise 9.6 solution: Rectangle.java
2   // Class Square definition
3   x
4   public class Rectangle extends Parallelogram {
5
6      public Rectangle( double x1, double y1, double x2, double y2,
7         double x3, double y3, double x4, double y4 )
8      {
9         super( x1, y1, x2, y2, x3, y3, x4, y4 );
10     }
11
12     public String toString()
13     {
```

```
14         return "\nCoordinates of Rectangle are: \n" + printCoordinates() +
15            "\nWidth is: " + getWidth() + "\nHeight is: " + getHeight() +
16            "\nArea is: " + getArea();
17      }
18
19   } // end class Square
```

```
1   // Exercise 9.6 solution: Square.java
2   // Class Square definition
3
4   public class Square extends Parallelogram {
5
6      public Square( double x1, double y1, double x2, double y2,
7         double x3, double y3, double x4, double y4 )
8      {
9         super( x1, y1, x2, y2, x3, y3, x4, y4 );
10     }
11
12     public String toString()
13     {
14        return "\nCoordinates of Square are: \n" + printCoordinates() +
15           "\nSide is: " + getHeight() + "\nArea is: " + getArea();
16     }
17
18  } // end class Square
```

9.7 Write down all the shapes you can think of—both two-dimensional and three-dimensional—and form those shapes into a shape hierarchy. Your hierarchy should have class `Shape` at the top. Class `TwoDimensionalShape` and class `ThreeDimensionalShape` should extend `Shape`. Once you have developed the hierarchy, declare each of the classes in it. We will use this hierarchy in the exercises to process all shapes as objects of superclass `Shape`.

ANS:



9.8 Create the classes in the inheritance hierarchy of Fig. 9.20. An `Employee` should have a first name, last name and social-security number. In addition, a `SalariedEmployee` should have a weekly salary; an `HourlyEmployee` should have a wage and a number of hours worked; a `CommissionEmployee` should have a commission rate and gross sales; and a `BasePlusCommissionEmployee` should have a base salary. Each class should have appropriate constructors, *set* methods and *get* methods. Write a program that instantiates objects of each of these classes and outputs all the information associated with each object (including the inherited information).

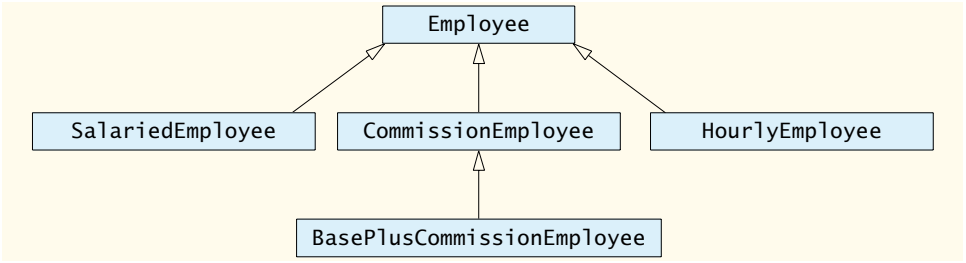


Fig. 9.20 `Employee` inheritance hierarchy.

ANS:

```
1  // Exercise 9.8 solution: Employee.java
2  // Employee superclass.
3
4  public class Employee {
5     private String firstName;
6     private String lastName;
7     private String socialSecurityNumber;
8
9     // constructor
10    public Employee( String first, String last, String ssn )
11    {
12       firstName = first;
13       lastName = last;
14       socialSecurityNumber = ssn;
15    }
16
17    // set first name
18    public void setFirstName( String first )
19    {
20       firstName = first;
21    }
22
23    // return first name
24    public String getFirstName()
25    {
26       return firstName;
27    }
28
29    // set last name
30    public void setLastName( String last )
31    {
32       lastName = last;
33    }
34
35    // return last name
36    public String getLastName()
37    {
38       return lastName;
39    }
```

```
40
41    // set social security number
42    public void setSocialSecurityNumber( String number )
43    {
44       socialSecurityNumber = number;  // should validate
45    }
46
47    // return social security number
48    public String getSocialSecurityNumber()
49    {
50       return socialSecurityNumber;
51    }
52
53    // return String representation of Employee object
54    public String toString()
55    {
56       return getFirstName() + " " + getLastName() +
57          "\nsocial security number: " + getSocialSecurityNumber();
58    }
59
60 } // end class Employee
```

```
1  // Exercise 9.8 solution: SalariedEmployee.java
2  // SalariedEmployee class derived from Employee.
3
4  public class SalariedEmployee extends Employee {
5     private double weeklySalary;
6
7     // constructor
8     public SalariedEmployee( String first, String last,
9        String socialSecurityNumber, double salary )
10    {
11       super( first, last, socialSecurityNumber );
12       setWeeklySalary( salary );
13    }
14
15    // set salaried employee's salary
16    public void setWeeklySalary( double salary )
17    {
18       weeklySalary = salary < 0.0 ? 0.0 : salary;
19    }
20
21    // return salaried employee's salary
22    public double getWeeklySalary()
23    {
24       return weeklySalary;
25    }
26
27    // return String representation of SalariedEmployee object
28    public String toString()
29    {
30       return "\nSalaried employee: " + super.toString() +
31          "\nWeekly salary: " + getWeeklySalary();
```

```
32     }
33
34  } // end class SalariedEmployee
```

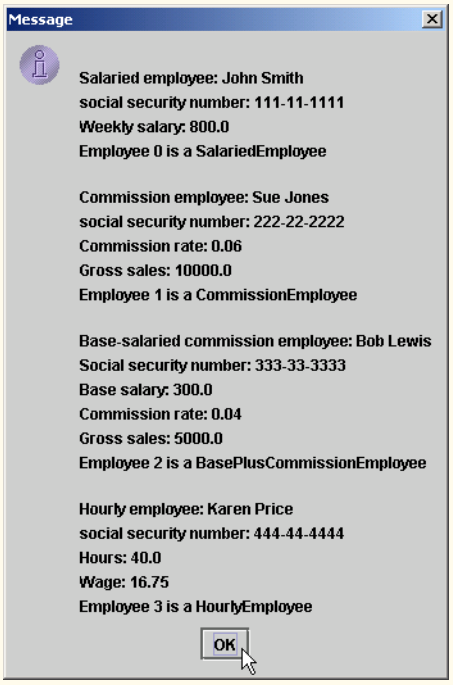
```
 1  // Exercise 9.8 solution: HourlyEmployee.java
 2  // HourlyEmployee class derived from Employee.
 3
 4  public class HourlyEmployee extends Employee {
 5     private double wage;   // wage per hour
 6     private double hours;  // hours worked for week
 7
 8     // constructor
 9     public HourlyEmployee( String first, String last,
10        String socialSecurityNumber, double hourlyWage, double hoursWorked )
11     {
12        super( first, last, socialSecurityNumber );
13        setWage( hourlyWage );
14        setHours( hoursWorked );
15     }
16
17     // set hourly employee's wage
18     public void setWage( double wageAmount )
19     {
20        wage = wageAmount < 0.0 ? 0.0 : wageAmount;
21     }
22
23     // return wage
24     public double getWage()
25     {
26        return wage;
27     }
28
29     // set hourly employee's hours worked
30     public void setHours( double hoursWorked )
31     {
32        hours = ( hoursWorked >= 0.0 && hoursWorked <= 168.0 ) ?
33           hoursWorked : 0.0;
34     }
35
36     // return hours worked
37     public double getHours()
38     {
39        return hours;
40     }
41
42     // return String representation of HourlyEmployee object
43     public String toString()
44     {
45        return "\nHourly employee: " + super.toString() + "\nHours: " +
46           getHours() + "\nWage: " + getWage();
47     }
48
49  } // end class HourlyEmployee
```

```
1  // Exercise 9.8 solution: CommissionEmployee.java
2  // CommissionEmployee class derived from Employee.
3
4  public class CommissionEmployee extends Employee {
5     private double grossSales;      // gross weekly sales
6     private double commissionRate;  // commission percentage
7
8     // constructor
9     public CommissionEmployee( String first, String last,
10       String socialSecurityNumber,
11       double grossWeeklySales, double percent )
12    {
13       super( first, last, socialSecurityNumber );
14       setGrossSales( grossWeeklySales );
15       setCommissionRate( percent );
16    }
17
18    // set commission employee's rate
19    public void setCommissionRate( double rate )
20    {
21       commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
22    }
23
24    // return commission employee's rate
25    public double getCommissionRate()
26    {
27       return commissionRate;
28    }
29
30    // set commission employee's weekly base salary
31    public void setGrossSales( double sales )
32    {
33       grossSales = sales < 0.0 ? 0.0 : sales;
34    }
35
36    // return commission employee's gross sales amount
37    public double getGrossSales()
38    {
39       return grossSales;
40    }
41
42    // return String representation of CommissionEmployee object
43    public String toString()
44    {
45       return "\nCommission employee: " + super.toString() +
46          "\nCommission rate: " + getCommissionRate() +
47          "\nGross sales: " + getGrossSales();
48    }
49
50 } // end class CommissionEmployee
```

```
1  // Exercise 9.8 solution: BasePlusCommissionEmployee.java
2  // BasePlusCommissionEmployee class derived from CommissionEmployee.
3
4  public class BasePlusCommissionEmployee extends CommissionEmployee {
5     private double baseSalary;  // base salary per week
6
7     // constructor
8     public BasePlusCommissionEmployee( String first, String last,
9        String socialSecurityNumber, double grossSalesAmount,
10       double rate, double baseSalaryAmount )
11    {
12       super( first, last, socialSecurityNumber, grossSalesAmount, rate );
13       setBaseSalary( baseSalaryAmount );
14    }
15
16    // set base-salaried commission employee's base salary
17    public void setBaseSalary( double salary )
18    {
19       baseSalary = salary < 0.0 ? 0.0 : salary;
20    }
21
22    // return base-salaried commission employee's base salary
23    public double getBaseSalary()
24    {
25       return baseSalary;
26    }
27
28    // return String representation of BasePlusCommissionEmployee
29    public String toString()
30    {
31       return "\nBase-salaried commission employee: " +
32          super.getFirstName() + " " + super.getLastName() +
33          "\nSocial security number: " + super.getSocialSecurityNumber() +
34          "\nBase salary: " + getBaseSalary() + "\nCommission rate: " +
35          getCommissionRate() + "\nGross sales: " + getGrossSales();
36    }
37
38 } // end class BasePlusCommissionEmployee
```

```
1  // Exercise 9.8 solution. EmployeeTest.java
2  // Employee hierarchy test program.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class EmployeeTest {
7
8     public static void main( String[] args )
9     {
10       DecimalFormat twoDigits = new DecimalFormat( "0.00" );
11
12       // create Employee array
13       Employee employees[] = new Employee[ 4 ];
14
```

```
15         // initialize array with Employees
16         employees[ 0 ] = new SalariedEmployee( "John", "Smith",
17            "111-11-1111", 800.00 );
18         employees[ 1 ] = new CommissionEmployee( "Sue", "Jones",
19            "222-22-2222", 10000, .06 );
20         employees[ 2 ] = new BasePlusCommissionEmployee( "Bob", "Lewis",
21            "333-33-3333", 5000, .04, 300 );
22         employees[ 3 ] = new HourlyEmployee( "Karen", "Price",
23            "444-44-4444", 16.75, 40 );
24
25         String output = "";
26
27         // generically process each element in array employees
28         for ( int i = 0; i < employees.length; i++ )
29            output += employees[ i ].toString() + "\nEmployee " + i +
30               " is a " + employees[ i ].getClass().getName() + "\n";
31
32         JOptionPane.showMessageDialog( null, output );  // display output
33         System.exit( 0 );
34
35      } // end main
36
37   } // end class EmployeeTest
```



10

Object-Oriented Programming: Polymorphism

Objectives

- To understand the concept of polymorphism.
- To understand how to use overridden methods to effect polymorphism.
- To distinguish between abstract and concrete classes.
- To learn how to declare `abstract` methods to create abstract classes.
- To appreciate how polymorphism makes systems extensible and maintainable.
- To be able to determine an object's type at execution time.

One Ring to rule them all, One Ring to find them,
One Ring to bring them all and in the darkness bind them.
John Ronald Reuel Tolkien

General propositions do not decide concrete cases.
Oliver Wendell Holmes

A philosopher of imposing stature doesn't think in a vacuum. Even his most abstract ideas are, to some extent, conditioned by what is or is not known in the time when he lives.
Alfred North Whitehead

SELF-REVIEW EXERCISES

10.1 Fill in the blanks in each of the following statements:

a) Treating a superclass object as a(n) _____ can cause errors.

ANS: subclass object

b) Polymorphism helps eliminate logic.

ANS: `switch`

c) If a class contains at least one abstract method, it is a(n) _____ class.

ANS: abstract

d) Classes from which objects can be instantiated are called _____ classes.

ANS: concrete

e) _____ involves using a superclass reference to invoke methods on superclass and subclass objects.

ANS: Polymorphism

f) Abstract methods are declared using keyword _____.

ANS: `abstract`

g) Casting a superclass object to a subclass object is called _____.

ANS: downcasting

10.2 State whether each of the following statements is *true* or *false*. If *false*, explain why.

a) It is possible to treat superclass objects and subclass objects similarly.

ANS: True.

b) All methods in an `abstract` superclass must be declared as `abstract` methods.

ANS: False. An abstract class can include methods with implementations.

c) Referring to a subclass object with a superclass reference is dangerous.

ANS: False. Referring to a superclass object with a subclass variable is dangerous.

d) A class is made abstract by declaring that class `abstract`.

ANS: True.

e) If a superclass declares an `abstract` method, a subclass must implement that method to become a concrete class.

ANS: True.

f) Inner classes are not allowed to access the members of the enclosing class.

ANS: False. Inner classes have access to all members of the enclosing class declaration.

EXERCISES

10.3 How is it that polymorphism enables you to program "in the general" rather than "in the specific"? Discuss the key advantages of programming "in the general."

ANS: Polymorphism enables the programmer to concentrate on the processing of common operations that are applied to all data types in the system without going into the individual details of each data type. The general processing capabilities are separated from the internal details of each type.

10.4 Distinguish between inheriting interface and inheriting implementation. How do inheritance hierarchies designed for inheriting interface differ from those designed for inheriting implementation?

ANS: When a class inherits implementation, it inherits previously defined functionality from another class. When a class inherits interface, it inherits the definition of what the interface to the new class type should be. The implementation is then provided by the programmer defining the new class type. Inheritance hierarchies designed for inheriting implementation are used to reduce the amont of new code that is being written. Such hierarchies are used to facilitate software reusability. Inheritance hierarchies designed for inheriting interface are used to write programs that perform generic processing of many class types. Such hierarchies

are commonly used to facilitate software extensibility (i.e., new types can be added to the hierarchy without changing the generic processing capabilitiesof the program.)

10.5 What are `abstract` methods? Describe a circumstance in which `abstract` methods would be appropriate.

ANS: Abstract methods are methods with the same method header that are defined throughout a class hierarchy. Abstract methods do not provide implementations. At least the super class occurrence of the method is preceded by the keyword **`abstract`**. Abstract methods are used to enable generic processing of an entire class hierarchy of objects through a super class reference. For example, in a shape hierarchy, all shapes can be drawn. If all shapes are derived from a super class **Shape** which contains an **`abstrace`** method, then generic processing of the hierarchy can be performed by calling every shape's **`draw`** generically through a super class **Shape** reference.

10.6 How does polymorphism promote extensibility?

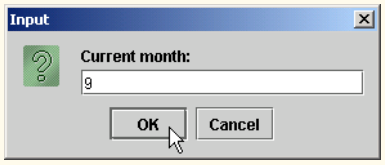
ANS: Polymorphism makes programs more extensible by making all method calls generic. When a new class type with the appropriate **`abstract`** method is added to the hierarchy, no changes need to be made to the generic method calls.

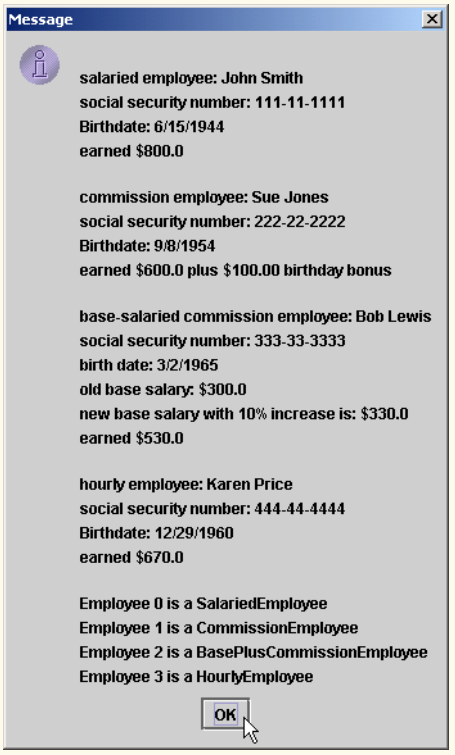
10.7 Modify the payroll system of Fig. 10.12–Fig. 10.17 to include `private` instance variable `birthDate` (use class `Date`) in class `Employee`. Assume that payroll is processed once per month. Create an array of `Employee` variables to store references to the various employee objects. In a loop, calculate the payroll for each `Employee` (polymorphically), and add a $100.00 bonus to the person's payroll amount if the current month is the month in which the `Employee`'s birthday occurs.

ANS:

```
1  // Exercise 10.7 solution: PayrollSystemTest.java
2  // Employee hierarchy test program.
3  import java.text.DecimalFormat;
4  import javax.swing.JOptionPane;
5
6  public class PayrollSystemTest {
7
8     public static void main( String[] args )
9     {
10       DecimalFormat twoDigits = new DecimalFormat( "0.00" );
11
12       // create Employee array
13       Employee employees[] = new Employee[ 4 ];
14
15       // initialize array with Employees
16       employees[ 0 ] = new SalariedEmployee( "John", "Smith",
17          "111-11-1111", 6, 15, 1944, 800.00 );
18       employees[ 1 ] = new CommissionEmployee( "Sue", "Jones",
19          "222-22-2222", 9, 8, 1954, 10000, .06 );
20       employees[ 2 ] = new BasePlusCommissionEmployee( "Bob", "Lewis",
21          "333-33-3333", 3, 2, 1965, 5000, .04, 300 );
22       employees[ 3 ] = new HourlyEmployee( "Karen", "Price",
23          "444-44-4444", 12, 29, 1960, 16.75, 40 );
24
25       String output = "";
26       int currentMonth = Integer.parseInt(
27          JOptionPane.showInputDialog( "Current month: " ) );
28
```

```
29          // validate current month
30          while ( !( currentMonth >= 0 && currentMonth <= 12 ) ) {
31             currentMonth = Integer.parseInt(
32                JOptionPane.showInputDialog( "Current month: " ) );
33          }
34
35          // generically process each element in array employees
36          for ( int i = 0; i < employees.length; i++ ) {
37             output += employees[ i ].toString();
38
39             // determine whether element is a BasePlusCommissionEmployee
40             if ( employees[ i ] instanceof BasePlusCommissionEmployee ) {
41
42                // downcast Employee reference to
43                // BasePlusCommissionEmployee reference
44                BasePlusCommissionEmployee currentEmployee =
45                   ( BasePlusCommissionEmployee ) employees[ i ];
46
47                double oldBaseSalary = currentEmployee.getBaseSalary();
48                output += "\nold base salary: $" + oldBaseSalary;
49
50                currentEmployee.setBaseSalary( 1.10 * oldBaseSalary );
51                output += "\nnew base salary with 10% increase is: $" +
52                   currentEmployee.getBaseSalary();
53
54             } // end if
55
56             // if month of employee's birthday, add $100 to salary
57             if ( currentMonth == employees[ i ].getBirthDate().getMonth() )
58                output += "\nearned $" + employees[ i ].earnings() +
59                   " plus $100.00 birthday bonus\n";
60             else
61                output += "\nearned $" + employees[ i ].earnings() + "\n";
62
63          } // end for
64
65          // get type name of each object in employees array
66          for ( int j = 0; j < employees.length; j++ )
67             output += "\nEmployee " + j + " is a " +
68                employees[ j ].getClass().getName();
69
70          JOptionPane.showMessageDialog( null, output );  // display output
71          System.exit( 0 );
72
73       } // end main
74
75    } // end class PayrollSystemTest
```






```
1  // Exercise 10.7 solution: Date.java
2  // Date class definition.
3
4  public class Date {
5     private int month;  // 1-12
6     private int day;    // 1-31 based on month
7     private int year;   // any year
8
9     // constructor: call checkMonth to confirm proper value for month;
10    // call checkDay to confirm proper value for day
11    public Date( int theMonth, int theDay, int theYear )
12    {
13       month = checkMonth( theMonth ); // validate month
14       year = theYear;                 // could validate year
15       day = checkDay( theDay );       // validate day
16    }
17
18    // utility method to confirm proper month value
19    private int checkMonth( int testMonth )
20    {
21       if ( testMonth > 0 && testMonth <= 12 )  // validate month
22          return testMonth;
23
```

```
24          else  // month is invalid
25             return 1;  // maintain object in consistent state
26       }
27
28       // utility method to confirm proper day value based on month and year
29       private int checkDay( int testDay )
30       {
31          int daysPerMonth[] =
32             { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
33
34          // check if day in range for month
35          if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
36             return testDay;
37
38          // check for leap year
39          if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
40               ( year % 4 == 0 && year % 100 != 0 ) ) )
41             return testDay;
42
43          return 1;  // maintain object in consistent state
44
45       } // end method checkDay
46
47       // return month
48       public int getMonth()
49       {
50          return month;
51       }
52
53       // return a String of the form month/day/year
54       public String toDateString()
55       {
56          return month + "/" + day + "/" + year;
57       }
58
59    } // end class Date
```

```
1  // Exercise 10.7 solution: Employee.java
2  // Employee abstract superclass.
3
4  public abstract class Employee {
5     private String firstName;
6     private String lastName;
7     private String socialSecurityNumber;
8     private Date birthDate;
9
10    // constructor
11    public Employee( String first, String last, String ssn,
12       int month, int day, int year )
13    {
14       firstName = first;
15       lastName = last;
16       socialSecurityNumber = ssn;
```

```
17          birthDate = new Date( month, day, year );
18       }
19
20       // set first name
21       public void setFirstName( String first )
22       {
23          firstName = first;
24       }
25
26       // return first name
27       public String getFirstName()
28       {
29          return firstName;
30       }
31
32       // set last name
33       public void setLastName( String last )
34       {
35          lastName = last;
36       }
37
38       // return last name
39       public String getLastName()
40       {
41          return lastName;
42       }
43
44       // set social security number
45       public void setSocialSecurityNumber( String number )
46       {
47          socialSecurityNumber = number;  // should validate
48       }
49
50       // return social security number
51       public String getSocialSecurityNumber()
52       {
53          return socialSecurityNumber;
54       }
55
56       // set birth date
57       public void setBirthDate( int month, int day, int year )
58       {
59          birthDate = new Date( month, day, year );
60       }
61
62       // return birth date
63       public Date getBirthDate()
64       {
65          return birthDate;
66       }
67
68       // return String representation of Employee object
69       public String toString()
70       {
```

```
71          return getFirstName() + " " + getLastName() +
72             "\nsocial security number: " + getSocialSecurityNumber() +
73             "\nBirthdate: " + birthDate.toDateString();
74       }
75
76       // abstract method overridden by subclasses
77       public abstract double earnings();
78
79    } // end abstract class Employee
```

```
1  // Exercise 10.7 solution: SalariedEmployee.java
2  // SalariedEmployee class derived from Employee.
3
4  public class SalariedEmployee extends Employee {
5     private double weeklySalary;
6
7     // constructor
8     public SalariedEmployee( String first, String last, String
9        socialSecurityNumber, int month, int day, int year, double salary )
10    {
11       super( first, last, socialSecurityNumber, month, day, year );
12       setWeeklySalary( salary );
13    }
14
15    // set salaried employee's salary
16    public void setWeeklySalary( double salary )
17    {
18       weeklySalary = salary < 0.0 ? 0.0 : salary;
19    }
20
21    // return salaried employee's salary
22    public double getWeeklySalary()
23    {
24       return weeklySalary;
25    }
26
27    // calculate salaried employee's pay;
28    // override abstract method earnings in Employee
29    public double earnings()
30    {
31       return getWeeklySalary();
32    }
33
34    // return String representation of SalariedEmployee object
35    public String toString()
36    {
37       return "\nsalaried employee: " + super.toString();
38    }
39
40 } // end class SalariedEmployee
```

```
1  // Exercise 10.7 solution: HourlyEmployee.java
2  // HourlyEmployee class derived from Employee.
3
4  public class HourlyEmployee extends Employee {
5     private double wage;   // wage per hour
6     private double hours;  // hours worked for week
7
8     // constructor
9     public HourlyEmployee( String first, String last,
10       String socialSecurityNumber, int month, int day, int year,
11       double hourlyWage, double hoursWorked )
12    {
13       super( first, last, socialSecurityNumber, month, day, year );
14       setWage( hourlyWage );
15       setHours( hoursWorked );
16    }
17
18    // set hourly employee's wage
19    public void setWage( double wageAmount )
20    {
21       wage = wageAmount < 0.0 ? 0.0 : wageAmount;
22    }
23
24    // return wage
25    public double getWage()
26    {
27       return wage;
28    }
29
30    // set hourly employee's hours worked
31    public void setHours( double hoursWorked )
32    {
33       hours = ( hoursWorked >= 0.0 && hoursWorked <= 168.0 ) ?
34          hoursWorked : 0.0;
35    }
36
37    // return hours worked
38    public double getHours()
39    {
40       return hours;
41    }
42
43    // calculate hourly employee's pay;
44    // override abstract method earnings in Employee
45    public double earnings()
46    {
47       if ( hours <= 40 )  // no overtime
48          return wage * hours;
49       else
50          return 40 * wage + ( hours - 40 ) * wage * 1.5;
51    }
52
53    // return String representation of HourlyEmployee object
54    public String toString()
55    {
```

```
56          return "\nhourly employee: " + super.toString();
57       }
58
59    } // end class HourlyEmployee
```

```
 1  // Exercise 10.7 solution: CommissionEmployee.java
 2  // CommissionEmployee class derived from Employee.
 3
 4  public class CommissionEmployee extends Employee {
 5     private double grossSales;      // gross weekly sales
 6     private double commissionRate;  // commission percentage
 7
 8     // constructor
 9     public CommissionEmployee( String first, String last,
10        String socialSecurityNumber, int month, int day, int year,
11        double grossWeeklySales, double percent )
12     {
13        super( first, last, socialSecurityNumber, month, day, year );
14        setGrossSales( grossWeeklySales );
15        setCommissionRate( percent );
16     }
17
18     // set commission employee's rate
19     public void setCommissionRate( double rate )
20     {
21        commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
22     }
23
24     // return commission employee's rate
25     public double getCommissionRate()
26     {
27        return commissionRate;
28     }
29
30     // set commission employee's weekly base salary
31     public void setGrossSales( double sales )
32     {
33        grossSales = sales < 0.0 ? 0.0 : sales;
34     }
35
36     // return commission employee's gross sales amount
37     public double getGrossSales()
38     {
39        return grossSales;
40     }
41
42     // calculate commission employee's pay;
43     // override abstract method earnings in Employee
44     public double earnings()
45     {
46        return getCommissionRate() * getGrossSales();
47     }
48
```

```
49    // return String representation of CommissionEmployee object
50    public String toString()
51    {
52       return "\ncommission employee: " + super.toString();
53    }
54
55 } // end class CommissionEmployee
```

```
1  // Exercise 10.7 solution: BasePlusCommissionEmployee.java
2  // BasePlusCommissionEmployee class derived from CommissionEmployee.
3
4  public class BasePlusCommissionEmployee extends CommissionEmployee {
5     private double baseSalary;  // base salary per week
6
7     // constructor
8     public BasePlusCommissionEmployee( String first, String last,
9        String socialSecurityNumber, int month, int day, int year,
10       double grossSalesAmount, double rate, double baseSalaryAmount )
11    {
12       super( first, last, socialSecurityNumber, month, day, year,
13          grossSalesAmount, rate );
14       setBaseSalary( baseSalaryAmount );
15    }
16
17    // set base-salaried commission employee's base salary
18    public void setBaseSalary( double salary )
19    {
20       baseSalary = salary < 0.0 ? 0.0 : salary;
21    }
22
23    // return base-salaried commission employee's base salary
24    public double getBaseSalary()
25    {
26       return baseSalary;
27    }
28
29    // calculate base-salaried commission employee's earnings;
30    // override method earnings in CommissionEmployee
31    public double earnings()
32    {
33       return getBaseSalary() + super.earnings();
34    }
35
36    // return String representation of BasePlusCommissionEmployee
37    public String toString()
38    {
39       return "\nbase-salaried commission employee: " +
40          super.getFirstName() + " " + super.getLastName() +
41          "\nsocial security number: " + super.getSocialSecurityNumber() +
42          "\nbirth date: " + super.getBirthDate().toDateString();
43    }
44
45 } // end class BasePlusCommissionEmployee
```

10.8 Implement the `Shape` hierarchy shown in Fig. 9.3. Each `TwoDimensionalShape` should contain method `getArea` to calculate the area of the two-dimensional shape. Each `ThreeDimensionalShape` should have methods `getArea` and `getVolume` to calculate the surface area and volume, respectively, of the three-dimensional shape, respectively. Create a program that uses an array of `Shape` references to objects of each concrete class in the hierarchy. The program should print the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a `TwoDimensionalShape` or a `ThreeDimensionalShape`. If a shape is a `TwoDimensionalShape`, display its area. If a shape is a `ThreeDimensionalShape`, display its area and volume.

ANS:

```
1  // Exercise 10.8 Solution: ShapeTest.java
2  // Program tests the Shape hierarchy.
3
4  public class ShapeTest {
5     private Shape shapeArray[];
6     private TwoDimensionalShape twoDArray[];
7     private ThreeDimensionalShape threeDArray[];
8
9     // create shapes
10    public ShapeTest() {
11       shapeArray = new Shape[ 4 ];
12       twoDArray = new TwoDimensionalShape[ 2 ];
13       threeDArray = new ThreeDimensionalShape[ 2 ];
14
15       Circle circle = new Circle( 22, 88, 21 );
16       shapeArray[ 0 ] = circle;
17       twoDArray[ 0 ] = circle;
18
19       Square square = new Square( 71, 96, 95 );
20       shapeArray[ 1 ] = square;
21       twoDArray[ 1 ] = square;
22
23       Sphere sphere = new Sphere( 8, 89, 78 );
24       shapeArray[ 2 ] = sphere;
25       threeDArray[ 0 ] = sphere;
26
27       Cube cube = new Cube( 79, 61, 73 );
28       shapeArray[ 3 ] = cube;
29       threeDArray[ 1 ] = cube;
30    }
31
32    // display shape info
33    public void displayShapeInfo()
34    {
35       // call method print on all shapes
36       for ( int i = 0; i < shapeArray.length; i++ ) {
37          System.out.print( shapeArray[ i ].getName() + ": " );
38          shapeArray[ i ].print();
39       }
40
41       // print area of 2D shapes
42       for ( int j = 0; j < twoDArray.length; j++ )
```

```
43             System.out.println( twoDArray[ j ].getName() +
44                "'s area is " + twoDArray[ j ].area() );
45
46          // print area and volume of 3D shapes
47          for ( int k = 0; k < threeDArray.length; k++ ) {
48             System.out.println( threeDArray[ k ].getName() +
49                "'s area is " + threeDArray[ k ].area() );
50             System.out.println( threeDArray[ k ].getName() +
51                "'s volume is " + threeDArray[ k ].volume() );
52          }
53       }
54
55       // create ShapeTest object and display info
56       public static void main( String args[] )
57       {
58          ShapeTest driver = new ShapeTest();
59          driver.displayShapeInfo();
60       }
61
62    }  // end class ShapeTest
```

```
Circle: (22, 88) radius: 21
Square: (71, 96) side: 95
Sphere: (8, 89) radius: 78
Cube: (79, 61) side: 73
Circle's area is 1385
Square's area is 9025
Sphere's area is 76453
Sphere's volume is 1490849
Cube's area is 31974
Cube's volume is 389017
```

```
1  // Exercise 10.8 Solution: Shape.java
2  // Definition of class Shape.
3
4  public abstract class Shape {
5     private int x, y;    // coordinates of shape
6
7     // constructor
8     public Shape( int x, int y )
9     {
10       this.x = x;
11       this.y = y;
12    }
13
14    // set x coordinate
15    public void setX( int x )
16    {
17       this.x = x;
18    }
19
```

```
20      // set y coordinate
21      public void setY( int y )
22      {
23         this.y = y;
24      }
25
26      // get x coordinate
27      public int getX()
28      {
29         return x;
30      }
31
32      // get y coordinate
33      public int getY()
34      {
35         return y;
36      }
37
38      // abstract methods
39      public abstract String getName();
40      public abstract void print();
41
42   }  // end class Shape
```

```
1   // Exercise 10.8 Solution: TwoDimensionalShape.java
2   // Definition of class TwoDimensionalShape.
3
4   public abstract class TwoDimensionalShape extends Shape {
5      private int dimension1, dimension2;
6
7      // constructor
8      public TwoDimensionalShape( int x, int y, int d1, int d2 )
9      {
10        super( x, y );
11        dimension1 = d1;
12        dimension2 = d2;
13     }
14
15     // set methods
16     public void setDimension1( int d )
17     {
18        dimension1 = d;
19     }
20
21     public void setDimension2( int d )
22     {
23        dimension2 = d;
24     }
25
26     // get methods
27     public int getDimension1()
28     {
29        return dimension1;
```

```
30     }
31
32     public int getDimension2()
33     {
34        return dimension2;
35     }
36
37     // abstract method
38     public abstract int area();
39
40  }  // end class TwoDimensionalShape
```

```
 1  // Exercise 10.8 Solution: ThreeDimensionalShape.java
 2  // Definition of class ThreeDimensionalShape.
 3
 4  public abstract class ThreeDimensionalShape extends Shape {
 5     private int dimension1, dimension2, dimension3;
 6
 7     // constructor
 8     public ThreeDimensionalShape(
 9        int x, int y, int d1, int d2, int d3 )
10     {
11        super( x, y );
12        dimension1 = d1;
13        dimension2 = d2;
14        dimension3 = d3;
15     }
16
17     // set methods
18     public void setDimension1( int d )
19     {
20        dimension1 = d;
21     }
22
23     public void setDimension2( int d )
24     {
25        dimension2 = d;
26     }
27
28     public void setDimension3( int d )
29     {
30        dimension3 = d;
31     }
32
33     // get methods
34     public int getDimension1() {
35        return dimension1;
36     }
37
38     public int getDimension2()
39     {
40        return dimension2;
41     }
```

```
42
43     public int getDimension3()
44     {
45        return dimension3;
46     }
47
48     // abstract methods
49     public abstract int area();
50     public abstract int volume();
51
52  }  // end class ThreeDimensionalShape
```

```
1  // Exercise 10.8 Solution: Circle.java
2  // Definition of class Circle.
3
4  public class Circle extends TwoDimensionalShape {
5
6     // constructor
7     public Circle( int x, int y, int radius )
8     {
9        super( x, y, radius, radius );
10    }
11
12    // overridden methods
13    public String getName()
14    {
15       return "Circle";
16    }
17
18    public void print()
19    {
20       System.out.println( "(" + super.getX() + ", " + super.getY() +
21          ") " + "radius: " + super.getDimension1() );
22    }
23
24    public int area()
25    {
26       return ( int )
27          ( Math.PI * super.getDimension1() * super.getDimension1() );
28    }
29
30    // set method
31    public void setRadius( int radius )
32    {
33       super.setDimension1( radius );
34    }
35
36    // get method
37    public int getRadius()
38    {
39       return super.getDimension1();
40    }
41
```

```
42 } // end class Circle
```

```
1  // Exercise 10.8 Solution: Square.java
2  // Definition of class Square.
3
4  public class Square extends TwoDimensionalShape {
5
6     // constructor
7     public Square( int x, int y, int side )
8     {
9        super( x, y, side, side );
10    }
11
12    // overridden methods
13    public String getName()
14    {
15       return "Square";
16    }
17
18    public void print()
19    {
20       System.out.println( "(" + super.getX() + ", " + super.getY() +
21          ") " + "side: " + super.getDimension1() );
22    }
23
24    public int area()
25    {
26       return super.getDimension1() * super.getDimension1();
27    }
28
29    // set method
30    public void setSide( int side )
31    {
32       super.setDimension1( side );
33    }
34
35    // get method
36    public int getSide()
37    {
38       return super.getDimension1();
39    }
40
41 } // end class Square
```

```
1  // Exercise 10.8 Solution: Sphere.java
2  // Definition of class Sphere.
3
4  public class Sphere extends ThreeDimensionalShape {
5
6     // constructor
7     public Sphere( int x, int y, int radius )
8     {
```

```
9           super( x, y, radius, radius, radius );
10       }
11
12       // overridden methods
13       public String getName()
14       {
15          return "Sphere";
16       }
17
18       public int area()
19       {
20          return ( int ) ( 4 * Math.PI *
21             super.getDimension1() * super.getDimension1() );
22       }
23
24       public int volume()
25       {
26          return ( int ) ( 4 / 3 * Math.PI * super.getDimension1() *
27             super.getDimension1() * super.getDimension1() );
28       }
29
30       public void print()
31       {
32          System.out.println( "(" + super.getX() + ", " + super.getY() +
33             ") " + "radius: " + super.getDimension1() );
34       }
35
36       // set method
37       public void setRadius( int radius )
38       {
39          super.setDimension1( radius );
40       }
41
42       // get method
43       public int getRadius()
44       {
45          return super.getDimension1();
46       }
47
48    }  // end class Sphere
```

```
1  // Exercise 10.8 Solution: Cube.java
2  // Definition of class Cube.
3
4  public class Cube extends ThreeDimensionalShape {
5
6     // constructor
7     public Cube( int x, int y, int side )
8     {
9        super( x, y, side, side, side );
10    }
11
```

```
12      // overridden methods
13      public String getName()
14      {
15         return "Cube";
16      }
17
18      public int area()
19      {
20         return ( int )
21            ( 6 * super.getDimension1() * super.getDimension1() );
22      }
23
24      public int volume()
25      {
26         return ( int ) ( super.getDimension1() *
27            super.getDimension1() * super.getDimension1() );
28      }
29
30      public void print()
31      {
32         System.out.println( "(" + super.getX() + ", " + super.getY() +
33            ") " + "side: " + super.getDimension1() );
34      }
35
36      // set method
37      public void setSide( int side )
38      {
39         super.setDimension1( side );
40      }
41
42      // get method
43      public int getSide()
44      {
45         return super.getDimension1();
46      }
47
48   }  // end class Cube
```

10.9 *(Drawing Application)* Modify the drawing program of Exercise 8.18 to create an application that draws random lines, rectangles and ovals. [*Note:* Like an applet, a `JFrame` has a `paint` method that you can override to draw on the background of the `JFrame`.]

For this exercise, modify the `MyLine`, `MyOval` and `MyRect` classes of Exercise 8.18 to create the class hierarchy in Fig. 10.35. The classes of the `MyShape` hierarchy should be "smart" shape classes such that objects of these classes know how to draw themselves (if provided with a `Graphics` object that tells them where to draw). The only `switch` or `if...else` logic in your program should be to determine the type of shape object to create. (Use random numbers to pick the shape type and the coordinates of each shape.) Once an object from this hierarchy is created, it will be manipulated for the rest of its lifetime as a superclass `MyShape` reference.

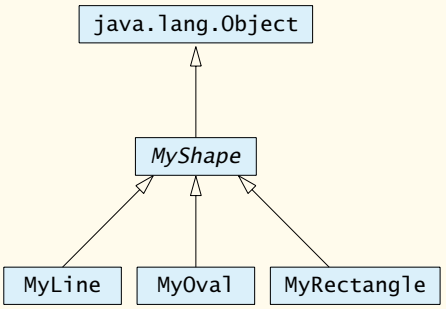


Fig. 10.35 MyShape hierarchy.

Class MyShape in Fig. 10.35 *must* be abstract. The only data representing the coordinates of the shapes in the hierarchy should be declared in class MyShape. Lines, rectangles and ovals can all be drawn if you know two points in space. Lines require *x1*, *y1*, *x2* and *y2* coordinates. The drawLine method of the Graphics class will connect the two points supplied with a line. If you have the same four coordinate values (*x1*, *y1*, *x2* and *y2*) for ovals and rectangles, you can calculate the four arguments needed to draw them. Each requires an upper-left *x*-coordinate value (the minimum of the two *x*-coordinate values), an upper-left *y*-coordinate value (minimum of the two *y* coordinate values), a *width* (the absolute value of the difference between the two *x*-coordinate values and a *height* (the absolute value of difference between the two *y*-coordinate values). [*Note*: In Chapter 13, each *x*,*y* pair will be captured by using mouse events from mouse interactions between the user and the program's background. These coordinates will be stored in an appropriate shape object selected by the user. As you begin the exercise, you will use random coordinate values as arguments to the constructor.]

In addition to the common data, class MyShape should declare at least the following methods:

a) A constructor with no arguments that sets the coordinates to 0.
b) A constructor with arguments that sets the coordinates to the values supplied.
c) Set methods, for each individual piece of data, that allow the programmer to set any piece of data independently for a shape in the hierarchy (e.g., if you have an instance variable x1, you should have a method setX1).
d) Get methods, for each individual piece of data, that allow the programmer to retrieve any piece of data independently for a shape in the hierarchy (e.g., if you have an instance variable x1, you should have a method getX1).
e) The abstract method

```
public abstract void draw( Graphics g );
```

which will be called from the program's paint method to draw a shape on the screen.

The preceding methods are required. If you would like to provide more methods for flexibility, please do so. However, be sure that any method you declare in this class is a method that would be used by *all* shapes in the hierarchy.

All data *must* be private to class MyShape. This forces you to use proper encapsulation of the data and provide proper *set/get* methods to manipulate the data. You are not allowed to declare new data that can be derived from existing information. As explained previously, the upper-left *x*, upper-left *y*, *width* and *height* needed to draw an oval or a rectangle can be calculated if you already know two points in space. All subclasses of MyShape should provide two constructors that mimic those provided by class MyShape.

Objects of the MyOval and MyRect classes should not calculate their upper-left *x*-coordinate, upper-left *y*-coordinate, *width* or *height* until they are about to draw. Never modify the *x1*, *y1*, *x2* and

y2 coordinates of a `MyOval` or `MyRect` object to prepare to draw them. Instead, use the temporary results of the calculations described above. This will help us enhance the program in Chapter 13 by allowing the user to select each shape's coordinates with the mouse.

There should be no `MyLine`, `MyOval` or `MyRect` variables in the program—only `MyShape` variables that contain references to `MyLine`, `MyOval` and `MyRect` objects. The program should keep an array of `MyShape` variables containing all shapes. The program's `paint` method should walk through the array of `MyShape` variables and draw every shape (i.e., call every shape's `draw` method).

Begin by declaring class `MyShape`, class `MyLine` and an application to test your classes. The application should have a `MyShape` instance variable that can refer to one `MyLine` object (created in the application's constructor). The `paint` method (for your subclass of `JFrame`) should draw the shape with a statement like

```
currentShape.draw( g );
```

where `currentShape` is the `MyShape` reference and `g` is the `Graphics` object that the shape will use to draw itself on the background of the window.

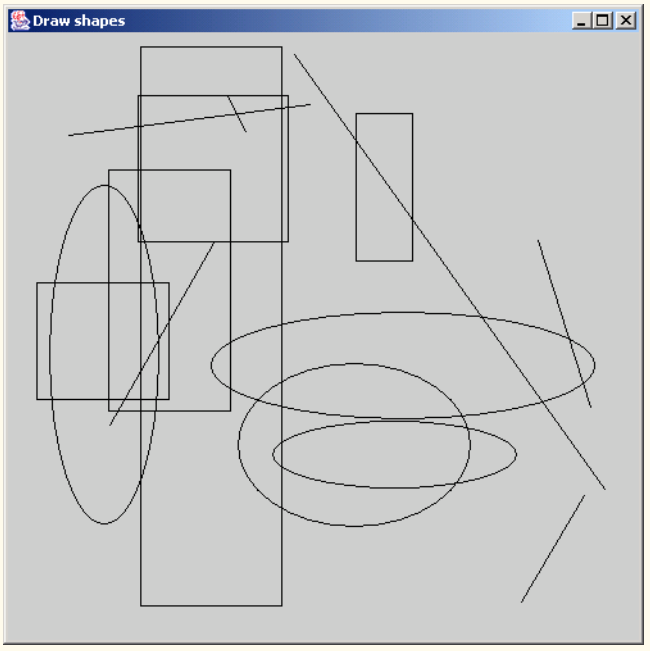
Next, change the single `MyShape` reference into an array of `MyShape` references, and hard code several `MyLine` objects into the program for drawing. The application's `paint` method should walk through the array of shapes and draw every shape.

After the preceding part is working, you should declare the `MyOval` and `MyRect` classes and add objects of these classes into the existing array. For now, all the shape objects should be created in the constructor for your subclass of `JFrame`. In Chapter 13, we will create the objects when the user chooses a shape and begins drawing it with the mouse.

ANS:

```
1  // Exercise 10.9 Solution: TestDrawWindow.java
2  // Program randomly draws shapes.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class TestDrawWindow extends JFrame {
7     private MyShape shape[];
8
9     // contructor
10    public TestDrawWindow()
11    {
12       super( "Draw shapes" );
13
14       int shapeType, x1, x2, y1, y2;
15       shape = new MyShape[ 15 ];
16
17       for ( int i = 0; i < shape.length; i++ ) {
18          x1 = ( int ) ( Math.random() * 450 + 25 );
19          x2 = ( int ) ( Math.random() * 450 + 25 );
20          y1 = ( int ) ( Math.random() * 450 + 25 );
21          y2 = ( int ) ( Math.random() * 450 + 25 );
22          shapeType = ( int ) ( Math.random() * 3 ) + 1;
23
24          switch ( shapeType ) {
25             case 1:  // line
26                shape[ i ] = new MyLine( x1, y1, x2, y2 );
27                break;
```

```
28                case 2:  // oval
29                   shape[ i ] = new MyOval( x1, y1, x2, y2 );
30                   break;
31                case 3:  // rectangle
32                   shape[ i ] = new MyRect( x1, y1, x2, y2 );
33                   break;
34             }
35
36          } // end for
37
38       }  // end constructor
39
40       // draw shapes
41       public void paint( Graphics g )
42       {
43          for ( int i = 0; i < shape.length; i++ )
44             shape[ i ].draw( g );
45       }
46
47       public static void main( String args[] )
48       {
49          TestDrawWindow window = new TestDrawWindow();
50          window.setSize( 500, 500 );
51          window.setVisible( true );
52          window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
53       }
54
55    }  // end class TestDrawWindow
```



```
1  // Exercise 10.9 Solution: MyShape.java
2  // Definition of class MyShape.
3
4  public abstract class MyShape extends Object {
5     private int x1, x2, y1, y2;
6
7     // default constructor initializes values with 0
8     public MyShape()
9     {
10       setX1( 0 );
11       setX2( 0 );
12       setY1( 0 );
13       setY2( 0 );
14    }
15
16    // constructor
17    public MyShape( int x1, int y1, int x2, int y2 )
18    {
19       setX1( x1 );
20       setX2( x2 );
21       setY1( y1 );
22       setY2( y2 );
23    }
24
25    // accessor and mutator methods for each of the four private variables
26    public void setX1( int x )
27    {
28
29       x1 = ( x >= 0 ? x : 0 );
30    }
31
32    public void setX2( int x )
33    {
34
35       x2 = ( x >= 0 ? x : 0 );
36    }
37
38    public void setY1( int y )
39    {
40       y1 = ( y >= 0 ? y : 0 );
41    }
42
43    public void setY2( int y )
44    {
45       y2 = ( y >= 0 ? y : 0 );
46    }
47
48    public int getX1()
49    {
50       return x1;
51    }
52
53    public int getX2()
54    {
```

```
55          return x2;
56       }
57
58       public int getY1()
59       {
60          return y1;
61       }
62
63       public int getY2()
64       {
65          return y2;
66       }
67
68       // abstract draw method
69       public abstract void draw( java.awt.Graphics g );
70
71    }  // end class MyShape
```

```
1   // Exercise 10.9 Solution: MyLine.java
2   // Definition of class MyLine.
3
4   public class MyLine extends MyShape {
5
6      // call default superclass constructor
7      public MyLine()
8      {
9         super();
10     }
11
12     // call superclass constructor passing parameters
13     public MyLine( int x1, int y1, int x2, int y2 )
14     {
15        super( x1, y1, x2, y2 );
16     }
17
18     // draw line
19     public void draw( java.awt.Graphics g )
20     {
21        g.drawLine( getX1(), getY1(), getX2(), getY2() );
22     }
23
24  }  // end class MyLine
```

```
1   // Exercise 10.9 Solution: MyOval.java
2   // Definition of class MyOval.
3
4   public class MyOval extends MyShape {
5
6      // call default superclass constructor
7      public MyOval()
8      {
```

```
 9          super();
10       }
11
12       // call superclass constructor passing parameters
13       public MyOval( int x1, int y1, int x2, int y2 )
14       {
15          super( x1, y1, x2, y2 );
16       }
17
18       // draw oval
19       public void draw( java.awt.Graphics g )
20       {
21          g.drawOval( Math.min( getX1(), getX2() ),
22             Math.min( getY1(), getY2() ), Math.abs( getX2() - getX1() ),
23             Math.abs( getY2() - getY1() ) );
24       }
25
26    }  // end class MyOval
```

```
 1  // Exercise 10.9 Solution: MyRect.java
 2  // Definition of class MyRect.
 3
 4  public class MyRect extends MyShape {
 5
 6     // call default superclass constructor
 7     public MyRect()
 8     {
 9        super();
10     }
11
12     // call superclass constructor passing parameters
13     public MyRect( int x1, int y1, int x2, int y2 )
14     {
15        super( x1, y1, x2, y2 );
16     }
17
18     // draw rectangle
19     public void draw( java.awt.Graphics g )
20     {
21        g.drawRect( Math.min( getX1(), getX2() ),
22           Math.min( getY1(), getY2() ), Math.abs( getX2() - getX1() ),
23           Math.abs( getY2() - getY1() ) );
24     }
25
26  }  // end class MyRect
```

10.10 In Exercise 10.9, you created a `MyShape` hierarchy in which classes `MyLine`, `MyOval` and `MyRect` extend `MyShape` directly. If the hierarchy was properly designed, you should be able to see the tremendous similarities between the `MyOval` and `MyRect` classes. Redesign and reimplement the code for the `MyOval` and `MyRect` classes to "factor out" the common features into the abstract class `MyBoundedShape` to produce the hierarchy in Fig. 10.36.

Class `MyBoundedShape` should declare two constructors that mimic the constructors of class `MyShape` and should also declare methods that calculate the upper-left *x*-coordinate, upper-left *y*-coordinate, *width* and *height*. No new data pertaining to the dimensions of the shapes should be declared in this class. Remember, the values needed to draw an oval or a rectangle can be calculated from two *(x,y)* coordinates. If designed properly, the new `MyOval` and `MyRect` classes should each have two constructors and a `draw` method.

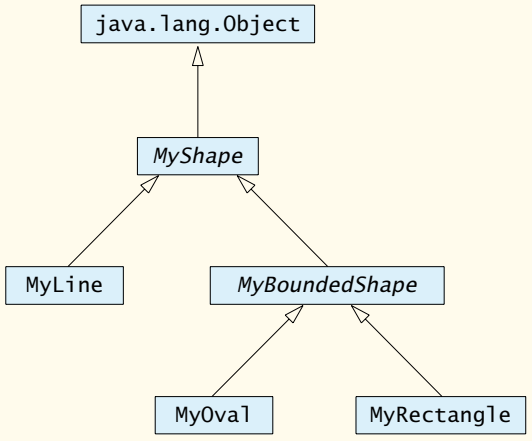
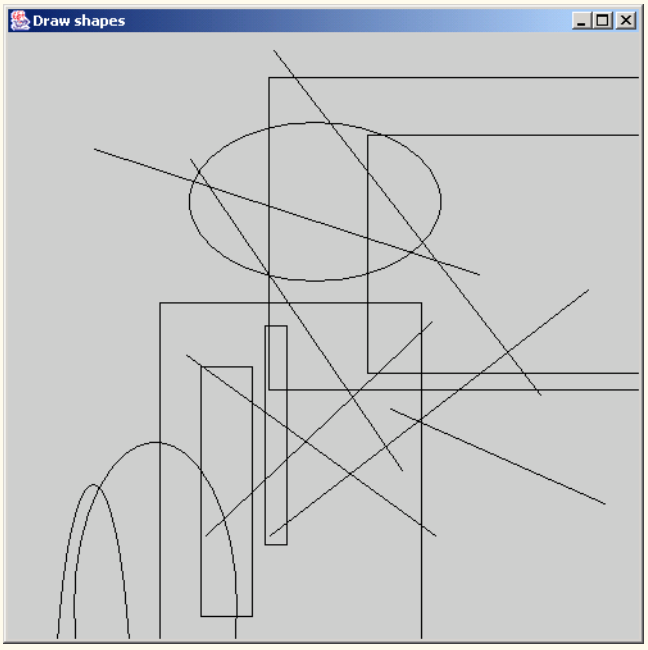


Fig. 10.36 MyShape hierarchy with MyBoundedShape.

ANS:

```
1  // Exercise 10.10 Solution: MyBoundedShape.java
2  // Definition of class MyBoundedShape.
3
4  public abstract class MyBoundedShape extends MyShape {
5
6     // call default superclass constructor
7     public MyBoundedShape()
8     {
9        super();
10    }
11
12    // call superclass constructor passing parameters
13    public MyBoundedShape( int x1, int y1, int x2, int y2 )
14    {
15       super( x1, y1, x2, y2 );
16    }
17
18    // get upper left x coordinate
19    protected int upperLeftX()
20    {
21       return Math.min( getX1(), getX2() );
22    }
23
```

```
24      // get upper left y coordinate
25      protected int upperLeftY()
26      {
27         return Math.min( getY1(), getY2() );
28      }
29
30      // get shape length
31      protected int length()
32      {
33         return Math.abs( getY2() - getY1() );
34      }
35
36      // get shape width
37      protected int width()
38      {
39         return Math.abs( getX2() - getX1() );
40      }
41
42   }  // end class MyBoundedShape
```




```
1   // Exercise 10.10 Solution: MyOval.java
2   // Definition of class MyOval.
3
4   public class MyOval extends MyBoundedShape {
5
6      // call default superclass constructor
7      public MyOval()
8      {
```

```
 9          super();
10       }
11
12       // call superclass constructor passing parameters
13       public MyOval( int x1, int y1, int x2, int y2 )
14       {
15          super( x1, y1, x2, y2 );
16       }
17
18       // draw oval
19       public void draw( java.awt.Graphics g )
20       {
21          g.drawOval( upperLeftX(), upperLeftY(), length(), width() );
22       }
23
24    }  // end class MyOval
```

```
 1  // Exercise 10.10 Solution: MyRect.java
 2  // Definition of class MyRect.
 3
 4  public class MyRect extends MyBoundedShape {
 5
 6     // call default superclass constructor
 7     public MyRect()
 8     {
 9        super();
10     }
11
12     // call superclass constructor passing parameters
13     public MyRect( int x1, int y1, int x2, int y2 )
14     {
15        super( x1, y1, x2, y2 );
16     }
17
18     // draw rectangle
19     public void draw ( java.awt.Graphics g )
20     {
21        g.drawRect( upperLeftX(), upperLeftY(), length(), width() );
22     }
23
24  }  // end class MyRect
```

11

Strings and Characters

Objectives

- To be able to create and manipulate nonmodifiable character string objects of class `String`.
- To be able to create and manipulate modifiable character string objects of class `StringBuffer`.
- To be able to create and manipulate objects of class `Character`.
- To be able to use a `StringTokenizer` object to break a `String` object into tokens.

The chief defect of Henry King
Was chewing little bits of string.
Hilaire Belloc

Vigorous writing is concise. A sentence should contain no unnecessary words, a paragraph no unnecessary sentences.
William Strunk, Jr.

I have made this letter longer than usual, because I lack the time to make it short.
Blaise Pascal

The difference between the almost-right word & the right word is really a large matter—it's the difference between the lightning bug and the lightning.
Mark Twain

Mum's the word.
Miguel de Cervantes

SELF-REVIEW EXERCISES

11.1 State whether each of the following is *true* or *false*. If *false*, explain why.

a) When `String` objects are compared using ==, the result is `true` if the `String`s contain the same values.

ANS: False. `String` objects that are compared using operator == are compared to determine whether they are the same object in memory.

b) A `String` can be modified after it is created.

ANS: False. `String` objects are immutable and cannot be modified after they are created. `StringBuffer` objects can be modified after they are created.

11.2 For each of the following, write a single statement that performs the indicated task.

a) Compare the string in `s1` to the string in `s2` for equality of contents.

ANS: `s1.equals( s2 )`

b) Append the string `s2` to the string `s1`, using +=.

ANS: `s1 += s2;`

c) Determine the length of the string in `s1`.

ANS: `s1.length()`

EXERCISES

Exercise 11.3–Exercise 11.6 are reasonably challenging. Once you have done these problems, you ought to be able to implement most popular card games easily.

11.3 Modify the program in Fig. 11.19 so that the card-dealing method deals a five-card poker hand. Then write methods that determine whether the hand contains

a) a pair.
b) two pairs.
c) three of a kind (e.g., three jacks).
d) four of a kind (e.g., four aces).
e) a flush (i.e., all five cards of the same suit).
f) a straight (i.e., five cards of consecutive face values).
g) a full house (i.e., two cards of one face value and three cards of another face value).

ANS:

```
1  // Exercise 11.3 Solution: Poker.java
2  // Program deals a five-card poker hand.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Poker extends JFrame {
8     private Card deck[];
9     private Card hand[];
10    private int currentCard;
11    private JButton dealButton, shuffleButton;
12    private JTextArea displayCard, status;
13    private int numbers[], triples, couples;
14    private String faces[], suits[], output;
15
16    // constructor
17    public Poker()
18    {
```

```
19         super( "Card Dealing Program" );
20
21         String faceArray[] = { "Ace", "Deuce", "Three", "Four", "Five",
22           "Six", "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
23         String suitArray[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
24
25         faces = faceArray;
26         suits = suitArray;
27
28         numbers = new int[ 13 ];
29         triples = 0;
30         couples = 0;
31
32         deck = new Card[ 52 ];
33         hand = new Card[ 5 ];
34
35         currentCard = -1;
36
37         // initialize the deck array
38         for ( int count = 0; count < deck.length; count++ )
39            deck[ count ] =
40               new Card( faces[ count % 13 ], suits[ count / 13 ] );
41
42         Container container = getContentPane();
43         container.setLayout( new FlowLayout() );
44
45         dealButton = new JButton( "Deal hand" );
46         dealButton.addActionListener(
47
48            new ActionListener() {  // anonymous inner class
49
50               // deal one hand of cards
51               public void actionPerformed( ActionEvent e )
52               {
53                  displayCard.setText( "" );    // clear text area
54                  output = "";
55                  triples = 0;
56                  couples = 0;
57
58                  // deal a round of cards
59                  for ( int count = 0; count < hand.length; count++ ) {
60                     Card dealt = dealCard();
61
62                     if ( dealt != null ) {
63                        hand[ count ] = dealt;
64                        displayCard.setText( displayCard.getText() +
65                           hand[ count ].toString() + "\n" );
66                     }
67
68                     else {
69                        displayCard.setText( "NOT ENOUGH CARDS TO DEAL" );
70                        status.setText( "Shuffle cards to continue" );
71                        return;
72                     }
73                  }
```

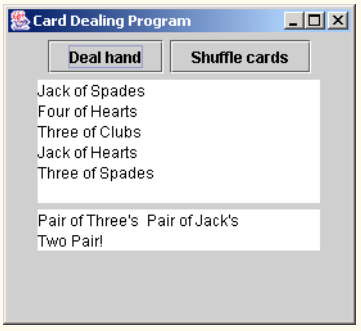
```
74
75                // calculate contents of the hand
76                totalHand();
77                pairs();
78                twoPair();
79                threeOfAKind();
80                fullHouse();
81                fourOfAKind();
82                straight();
83                flush();
84             }
85
86          }  // end anonymous inner class
87
88       ); // end call to addActionListener
89
90       container.add( dealButton );
91
92       shuffleButton = new JButton( "Shuffle cards" );
93       shuffleButton.addActionListener(
94
95          new ActionListener() {  // anonymous inner class
96
97             // shuffle deck
98             public void actionPerformed( ActionEvent e )
99             {
100               displayCard.setText( "SHUFFLING ..." );
101               shuffle();
102               displayCard.setText( "DECK IS SHUFFLED" );
103            }
104
105         }  // end anonymous inner class
106
107      ); // end call to addActionListener
108
109      container.add( shuffleButton );
110
111      displayCard = new JTextArea( 6, 20 );
112      displayCard.setEditable( false );
113      container.add( displayCard );
114
115      status = new JTextArea( 2, 20 );
116      status.setEditable( false );
117      container.add( status );
118
119      setSize( 275, 250 );  // set the window size
120      show();               // show the window
121
122   }  // end Poker
123
124   // shuffle deck of cards with one-pass algorithm
125   public void shuffle()
126   {
127      currentCard = -1;
```

```
128
129       // swap two cards as many times as there are cards in the deck
130       for ( int first= 0; first< deck.length; first++ ) {
131          int second =  ( int ) ( Math.random() * deck.length );
132          Card temp = deck[ first];
133          deck[ first] = deck[ second ];
134          deck[ second ] = temp;
135       }
136
137       dealButton.setEnabled( true );
138    }
139
140    // deal one card
141    public Card dealCard()
142    {
143       if ( ++currentCard < deck.length )
144          return deck[ currentCard ];
145
146       else {
147          dealButton.setEnabled( false );
148
149          return null;
150       }
151    }
152
153    // tally the number of each face card in hand
154    private void totalHand()
155    {
156       // initialize all elements of numbers[] to zero
157       for ( int x = 0; x < faces.length; x++ )
158          numbers[ x ] = 0;
159
160       // compare each card in the hand to each element in the faces array
161       for ( int h = 0; h < hand.length; h++ )
162
163          for ( int f = 0; f < faces.length; f++ )
164
165             if ( hand[ h ].getFace().equals( faces[ f ] ) )
166                ++numbers[ f ];
167    }
168
169    // determine if hand contains pairs
170    public void pairs()
171    {
172       for ( int k = 0; k < faces.length; k++ )
173
174          if ( numbers[ k ] == 2 ) {
175             output += ( "Pair of " + faces[ k ] + "'s  " );
176             couples++;
177          }
178
179       status.setText( output );
180    }
181
```

```
182    // determine if hand contains a three of a kind
183    public void threeOfAKind()
184    {
185       for ( int k = 0; k < faces.length; k++ )
186
187          if ( numbers[ k ] == 3 ) {
188             output += ( "Three " + faces[ k ] + "'s" );
189             triples++;
190             break;
191          }
192
193       status.setText( output );
194    }
195
196    // determine if hand contains a four of a kind
197    public void fourOfAKind()
198    {
199       for ( int k = 0; k < faces.length; k++ )
200
201          if ( numbers[ k ] == 4 )
202             output += ( "Four " + faces[ k ] + "'s" );
203
204       status.setText( output );
205    }
206
207    // determine if hand contains a flush
208    public void flush()
209    {
210       String theSuit = hand[ 0 ].getSuit();
211
212       for ( int s = 1; s < hand.length; s++ )
213
214          if ( hand[ s ].getSuit().compareTo( theSuit ) != 0 )
215             return;   // not a flush
216
217       output += ( "Flush in " + theSuit );
218       status.setText( output );
219    }
220
221    // determine if hand contains a straight
222    public void straight()
223    {
224       int locations[] = new int[ 5 ], z = 0;
225
226       for ( int y = 0; y < numbers.length; y++ )
227
228          if ( numbers[ y ] == 1 )
229             locations[ z++ ] = y;
230
231       bubbleSort( locations );
232
233       int faceValue = locations[ 0 ];
234
235       for ( int m = 1; m < locations.length; m++ ) {
```

```
236
237          if ( faceValue != locations[ m ] - 1 )
238             return;   // not a straight
239
240          else
241             faceValue = locations[ m ];
242       }
243
244       output += "Straight ";
245       status.setText( output );
246    }
247
248    // sort hand in ascending order
249    private void bubbleSort( int values[] )
250    {
251       for ( int pass = 1; pass < values.length; pass++ )
252
253          for ( int comp = 0; comp < values.length - 1; comp++ )
254
255             if ( values[ comp ] > values[ comp + 1 ] ) {
256                int temp = values[ comp ];
257                values[ comp ] = values[ comp + 1 ];
258                values[ comp + 1 ] = values[ comp ];
259             }
260    }
261
262    // determine if hand contains a full house
263    public void fullHouse()
264    {
265       if ( couples == 1 && triples == 1 ) {
266          output += "\nFull House!";
267          status.setText( output );
268       }
269    }
270
271    // determine if hand contains two pairs
272    public void twoPair()
273    {
274       if ( couples == 2 ) {
275          output += "\nTwo Pair!";
276          status.setText( output );
277       }
278    }
279
280    // execute application
281    public static void main( String args[] )
282    {
283       Poker application = new Poker();
284
285       // set application to terminate on close
286       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
287
288    }  // end method main
289
```

```
290
291    // internal class to represent card
292    class Card {
293       private String face;
294       private String suit;
295
296       // constructor to initialize Card
297       public Card( String f, String s )
298       {
299          face = f;
300          suit = s;
301       }
302
303       // get suit
304       protected String getSuit()
305       {
306          return suit;
307       }
308
309       // get face
310       protected String getFace()
311       {
312          return face;
313       }
314
315       // return String representation of Card
316       public String toString()
317       {
318          return face + " of " + suit;
319       }
320
321    }  // end class Card
322
323 }  // end class Poker
```



11.4 Use the methods developed in Exercise 11.3 to write a program that deals two five-card poker hands, evaluates each hand and determines which is the better hand.

ANS:

```
1  // Exercise 11.4 Solution: Poker2.java
2  // Program deals two Poker "hands"
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class Poker2 extends JFrame {
8     private Card deck[], hand1[], hand2[];
9     private int currentCard;
10    private JButton dealButton, shuffleButton;
11    private JTextField handField1, handField2;
12    private JTextArea displayCard1, displayCard2;
13    private JLabel handLabel1, handLabel2, status;
14    private String faces[], suits[], output;
15    private int numbers1[], numbers2[];
16    private int highValue1, highValue2, group1, group2;
17    private boolean straightHand1, straightHand2, pair1, pair2;
18    private final int ONEPAIR = 2;
19    private final int TWOPAIR = 4;
20    private final int THREEKIND = 6;
21    private final int STRAIGHT = 8;
22    private final int FULLHOUSE = 10;
23    private final int FLUSH = 12;
24    private final int FOURKIND = 14;
25    private final int STRAIGHTFLUSH = 16;
26
27    public Poker2()
28    {
29       super( "Card Dealing Program" );
30
31       String f[] = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
32          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
33       String s[] = { "Hearts", "Diamonds", "Clubs", "Spades" };
34
35       faces = f;
36       suits = s;
37       numbers1 = new int[ 13 ];
38       numbers2 = new int[ 13 ];
39       hand1 = new Card[ 5 ];
40       hand2 = new Card[ 5 ];
41       deck = new Card[ 52 ];
42       currentCard = -1;
43
44       for ( int i = 0; i < deck.length; i++ )
45          deck[ i ] = new Card( faces[ i % 13 ], suits[ i / 13 ] );
46
47       Container container = getContentPane();
48       container.setLayout( new FlowLayout() );
49
50       dealButton = new JButton( "Deal hand" );
51       dealButton.addActionListener(
52
```

```
53          new ActionListener() {  // anonymous inner class
54
55             // deal hands
56             public void actionPerformed( ActionEvent event )
57             {
58                handField1.setText( "" );
59                handField2.setText( "" );
60                displayCard1.setText( "" );
61                displayCard2.setText( "" );
62                output = "";
63
64                group1 = 0;
65                group2 = 0;
66                highValue1 = 0;
67                highValue2 = 0;
68
69                // deal hands
70                for ( int n = 0; n < hand1.length; n++ ) {
71                   Card dealt1 = dealCard();
72                   Card dealt2 = dealCard();
73
74                   if ( dealt1 != null && dealt2 != null ) {
75                      hand1[ n ] = dealt1;
76                      hand2[ n ] = dealt2;
77                      displayCard1.setText(
78                         displayCard1.getText() +
79                         hand1[ n ].toString() + "\n" );
80                      displayCard2.setText(
81                         displayCard2.getText() +
82                         hand2[ n ].toString() + "\n" );
83                   }
84
85                   // not enough cards to deal
86                   else if ( dealt1 == null || dealt2 == null ) {
87                      status.setText( "NOT ENOUGH CARDS. SHUFFLE DECK." );
88                      return;
89                   }
90                }
91
92                totalHand();    // calculate contents of hands
93                pair();
94                threeOfAKind();
95                fourOfAKind();
96                straight();
97                flush();
98
99                if ( group1 > group2 )
100                  status.setText( "top hand is better" );
101
102               else if ( group1 < group2 )
103                  status.setText( "bottom hand is better" );
104
105               else {
106
```

```
107                   // test for the highest card
108                   if ( highValue1 > highValue2 )
109                      status.setText( "top hand is better" );
110                   else if ( highValue1 < highValue2 )
111                      status.setText( "bottom hand is better" );
112                   else
113                      status.setText( "they are equal" );
114                }
115
116                // display results in text fields
117                results();
118
119             }  // end method actionPerformed
120
121          }  // end anonymous inner class
122
123       ); // end call to addActionListener
124
125       container.add( dealButton );
126
127       shuffleButton = new JButton( "Shuffle cards" );
128
129       shuffleButton.addActionListener(
130
131          new ActionListener() {  // anonymous inner class
132
133             // shuffle deck
134             public void actionPerformed( ActionEvent event )
135             {
136                displayCard1.setText( "" );
137                displayCard2.setText( "" );
138                handField1.setText( "" );
139                handField2.setText( "" );
140                status.setText( "SHUFFLING..." );
141                shuffle();
142                status.setText( "DECK IS SHUFFLED" );
143             }
144          }
145       );
146
147       container.add( shuffleButton );
148
149       handLabel1 = new JLabel( "Hand 1:" );
150       handField1 = new JTextField( 15 );
151       handField1.setEditable( false );
152       displayCard1 = new JTextArea( 6, 20 );
153       displayCard1.setEditable( false );
154       handLabel2 = new JLabel( "Hand 2:" );
155       handField2 = new JTextField( 15 );
156       handField2.setEditable( false );
157       displayCard2 = new JTextArea( 6, 20 );
158       displayCard2.setEditable( false );
159       status = new JLabel();
160
```

```
161         container.add( handLabel1 );
162         container.add( handField1 );
163         container.add( displayCard1 );
164         container.add( handLabel2 );
165         container.add( handField2 );
166         container.add( displayCard2 );
167         container.add( status );
168
169         setSize( 250, 350 );   // set the window size
170         setVisible( true );    // show the window
171
172      } // end constructor
173
174      // shuffle deck of cards with one-pass algorithm
175      public void shuffle()
176      {
177         currentCard = -1;
178
179         for ( int first= 0; first< deck.length; first++ ) {
180            int second =  ( int ) ( Math.random() * 52 );
181            Card temp = deck[ first];
182            deck[ first] = deck[ second ];
183            deck[ second ] = temp;
184         }
185
186         dealButton.setEnabled( true );
187      }
188
189      // deal one card
190      public Card dealCard()
191      {
192         if ( ++currentCard < deck.length )
193            return deck[ currentCard ];
194         else {
195            dealButton.setEnabled( false );
196            return null;
197         }
198      }
199
200      // tally the number of each kind of card in hand
201      private void totalHand()
202      {
203         for ( int x = 0; x < faces.length; x++ )
204            numbers1[ x ] = numbers2[ x ] = 0;
205
206         // check each card in both hands
207         for ( int h = 0; h < hand1.length; h++ )
208
209            for ( int f = 0; f < faces.length; f++ ) {
210
211               // hand 1
212               if ( hand1[ h ].getFace().equals( faces[ f ] ) ) {
213                  ++numbers1[ f ];
214
```

```
215                // store highest value in hand
216                if ( f == 0 )
217                   highValue1 = 13;  // special value for Ace
218
219                else if ( f > highValue1 )
220                   highValue1 = f;
221             }
222
223             // hand 2
224             if ( hand2[ h ].getFace().equals( faces[ f ] ) ) {
225                ++numbers2[ f ];
226
227                // store highest value in hand
228                if ( f == 0 )
229                   highValue2 = 13;  // special value for Ace
230
231                else if ( f > highValue2 )
232                   highValue2 = f;
233             }
234          }
235
236    }  // end method totalHand
237
238    // check hands for pairs
239    public void pair()
240    {
241       int numberPairs1 = 0, numberPairs2 = 0, highest1 = 0, highest2 = 0;
242
243       pair1 = false;
244       pair2 = false;
245
246       // count number of pairs in hands
247       for ( int k = 0; k < faces.length; k++ ) {
248
249          // pair found in hand 1
250          if ( numbers1[ k ] == 2 ) {
251
252             pair1 = true;
253
254             // store highest pair
255             if ( k == 0 )
256                highest1 = 13;  // special value for ace
257
258             if ( k > highest1 )
259                highest1 = k;
260
261             numberPairs1++;
262          }
263
264          // pair found in hand 2
265          if ( numbers2[ k ] == 2 ) {
266
267             pair2 = true;
268
```

```
269             // store highest pair
270             if ( k == 0 )
271                highest2 = 13;  // special value for ace
272
273             if ( k > highest2 )
274                highest2 = k;
275
276             numberPairs2++;
277          }
278       }
279
280       // evaluate number of pairs in each hand
281       if ( numberPairs1 == 1 )
282          group1 = ONEPAIR;
283
284       else if ( numberPairs1 == 2 )
285          group1 = TWOPAIR;
286
287       if ( numberPairs2 == 1 )
288          group2 = ONEPAIR;
289
290       else if ( numberPairs2 == 2 )
291          group2 = TWOPAIR;
292
293       if ( highest1 > highest2 )
294          group1++;
295
296       else if ( highest2 > highest1 )
297          group2++;
298
299    }  // end pair
300
301    // check hands for three of a kind
302    public void threeOfAKind()
303    {
304       int tripletValue1 = 0, tripletValue2 = 0;
305       boolean flag = false, flag2 = false;
306
307       // check for three of a kind
308       for ( int k = 0; k < faces.length; k++ ) {
309
310          // three of a kind found in hand 1
311          if ( numbers1[ k ] == 3 ) {
312             group1 = THREEKIND;
313             flag = true;
314
315             // store value of triplet
316             if ( k == 0 )
317                tripletValue1 = 13;   // special value for ace
318
319             if ( k > tripletValue1 )
320                tripletValue1 = k;
321
```

```
322             if ( pair1 == true )
323                group1 = FULLHOUSE;
324          }
325
326          // three of a kind found in hand 2
327          if ( numbers2[ k ] == 3 ) {
328             group2 = THREEKIND;
329             flag2 = true;
330
331             // store value of triplet
332             if ( k == 0 )
333                tripletValue2 = 13;   // special value for ace
334
335             if ( k > tripletValue2 )
336                tripletValue2 = k;
337
338             if ( pair2 == true )
339                group2 = FULLHOUSE;
340          }
341       }
342
343       // both hands have three of a kind,
344       // determine which triplet is higher in value
345       if ( flag == true && flag2 == true )
346
347          if ( tripletValue1 > tripletValue2 )
348             group1++;
349
350          else if ( tripletValue1 < tripletValue2 )
351             group2++;
352
353    }  // end threeOfAKind
354
355    // check hands for four of a kind
356    public void fourOfAKind()
357    {
358       int highest1 = 0, highest2 = 0;
359       boolean flag = false, flag2 = false;
360
361       // check for four of a kind
362       for ( int k = 0; k < faces.length; k++ ) {
363
364          // hand 1
365          if ( numbers1[ k ] == 4 ) {
366             group1 = FOURKIND;
367             flag = true;
368
369             if ( k == 0 )
370                highest1 = 13;   // special value for ace
371
372             if ( k > highest1 )
373                highest1 = k;
374          }
375
```

```
376          // hand 2
377          if ( numbers2[ k ] == 4 ) {
378             group2 = FOURKIND;
379             flag2 = true;
380
381             if ( k == 0 )
382                highest2 = 13;   // special value for ace
383
384             if ( k > highest2 )
385                highest2 = k;
386          }
387       }
388
389       // if both hands contain four of a kind, determine which is higher
390       if ( flag == true && flag2 == true )
391
392          if ( highest1 > highest2 )
393             group1++;
394
395          else if ( highest1 < highest2 )
396             group2++;
397
398    }  // end method fourOfAKind
399
400    // check hands for flush
401    public void flush()
402    {
403       boolean flag1 = true, flag2 = true;
404
405       // check hand 1
406       String theSuit = hand1[ 0 ].getSuit();
407
408       for ( int s = 1; s < hand1.length && flag1 == true; s++ )
409          if ( hand1[ s ].getSuit().compareTo( theSuit ) != 0 )
410             flag1 = false;   // not a flush
411
412       // check hand 2
413       theSuit = hand2[ 0 ].getSuit();
414
415       for ( int s = 1; s < hand2.length && flag2 == true; s++ )
416          if ( hand2[ s ].getSuit().compareTo( theSuit ) != 0 )
417             flag2 = false;   // not a flush
418
419       // hand 1 is a flush
420       if ( flag1 == true ) {
421          group1 = FLUSH;
422
423          // straight flush
424          if ( straightHand1 == true )
425             group1 = STRAIGHTFLUSH;
426       }
427
428       // hand 2 is a flush
429       if ( flag2 == true ) {
```

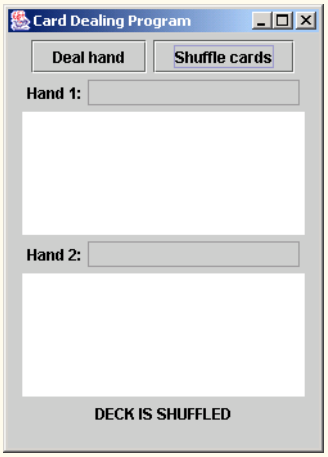
```
430          group2 = FLUSH;
431
432          // straight flush
433          if ( straightHand2 == true )
434             group2 = STRAIGHTFLUSH;
435       }
436
437    }  // end method flush
438
439    // check hands for straights
440    public void straight()
441    {
442       int locations1[] = new int[ 5 ],
443           locations2[] = new int[ 5 ],
444           value;
445
446
447       // check each card in both hands
448       for ( int h = 0; h < hand1.length; h++ )
449
450          for ( int f = 0; f < faces.length; f++ ) {
451
452             // hand 1
453             if ( hand1[ h ].getFace().equals( faces[ f ] ) ) {
454
455                if ( f == 0 )
456                   value = 13;  // special value for Ace
457
458                else
459                   value = f;
460
461                locations1[ h ] = value;
462             }
463
464             // hand 2
465             if ( hand2[ h ].getFace().equals( faces[ f ] ) ) {
466
467                if ( f == 0 )
468                   value = 13;  // special value for Ace
469
470                else
471                   value = f;
472
473                locations2[ h ] = value;
474             }
475          }
476
477
478       // sort hands
479       bubbleSort( locations1 );
480       bubbleSort( locations2 );
481
482       int faceValue = locations1[ 0 ];
483       boolean flag1 = true, flag2 = true;
```

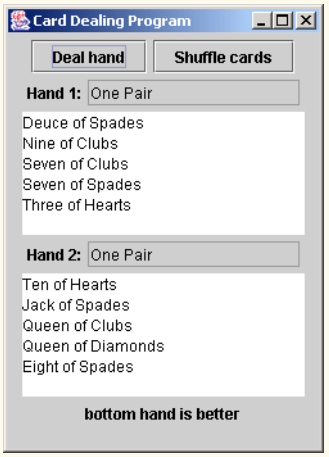
```
484
485       // test hand 1 for straight
486       for ( int m = 1; m < locations1.length && flag1 == true; m++ ) {
487
488          // not a straight
489          if ( faceValue != locations1[ m ] - 1 )
490             flag1 = false;   // not a straight
491
492          else
493             faceValue = locations1[ m ];
494       }
495
496       faceValue = locations2[ 0 ];
497
498       // test hand 2 for straight
499       for ( int m = 1; m < locations1.length &&
500          flag2 == true; m++ ) {
501
502          // not a straight
503          if ( faceValue != locations2[ m ] - 1 )
504             flag2 = false;
505
506          else
507             faceValue = locations2[ m ];
508       }
509
510       int highest1 = 0,
511           highest2 = 0;
512
513       // hand 1 is a straight
514       if ( flag1 == true ) {
515          straightHand1 = true;
516          group1 = STRAIGHT;
517          highest1 = locations1[ 4 ];
518       }
519
520       // hand 2 is a straight
521       if ( flag2 == true ) {
522          straightHand2 = true;
523          group2 = STRAIGHT;
524          highest2 = locations2[ 4 ];
525       }
526
527       // if both hands contain straights,
528       // determine which is higher
529       if ( straightHand1 == true && straightHand2 == true )
530
531          if ( highest1 > highest2 )
532             group1++;
533
534          else if ( highest2 > highest1 )
535             group2++;
536
537    }  // end method straight
```

```
538
539    // sort hand in ascending order
540    private void bubbleSort( int values[] )
541    {
542       for ( int pass = 1; pass < values.length; pass++ )
543
544          for ( int comp = 0; comp < values.length - 1; comp++ )
545
546             if ( values[ comp ] > values[ comp + 1 ] ) {
547                int temp = values[ comp ];
548                values[ comp ] = values[ comp + 1 ];
549                values[ comp + 1 ] = temp;
550             }
551    }
552
553    // display hand values as Strings in textFields
554    public void results()
555    {
556       handField1.setText( "" );
557       handField2.setText( "" );
558
559       String handValue[] = new String[ 2 ];
560       int value = group1;
561
562       for ( int i = 0; i < 2; i++ ) {
563
564          if ( i == 1 )
565             value = group2;
566
567          switch ( value ) {
568
569             case 2: case 3:
570                handValue[ i ] = "One Pair";
571                break;
572
573             case 4: case 5:
574                handValue[ i ] = "Two Pair";
575                break;
576
577             case 6: case 7:
578                handValue[ i ] = "Three of a Kind";
579                break;
580
581             case 8: case 9:
582                handValue[ i ] = "Straight";
583                break;
584
585             case 10: case 11:
586                handValue[ i ] = "Full House";
587                break;
588
589             case 12: case 13:
590                handValue[ i ] = "Flush";
591                break;
```

```
592
593               case 14: case 15:
594                  handValue[ i ] = "Four of a Kind";
595                  break;
596
597               case 16:
598                  handValue[ i ] = "Straight Flush";
599                  break;
600
601            }  // end switch
602
603         }  // end for
604
605         handField1.setText( handValue[ 0 ] );
606         handField2.setText( handValue[ 1 ] );
607
608      }  // end method results
609
610      // execute application
611      public static void main( String args[] )
612      {
613         Poker2 application = new Poker2();
614
615         // set application to terminate on close
616         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
617
618      }  // end method main
619
620   }  // end class Poker
621
622   // class to represent card
623   class Card {
624      private String face;
625      private String suit;
626
627      // constructor to initialize Card
628      public Card( String f, String s )
629      {
630         face = f;
631         suit = s;
632      }
633
634      // get suit
635      protected String getSuit()
636      {
637         return suit;
638      }
639
640      // get face
641      protected String getFace()
642      {
643         return face;
644      }
645
```

```
646    // return String representation of Card
647    public String toString()
648    {
649       return face + " of " + suit;
650    }
651
652 }  // end class Card
```





11.5 Modify the program developed in Exercise 11.4 so that it can simulate the dealer. The dealer's five-card hand is dealt "face down" so the player cannot see it. The program should then evaluate the dealer's hand and, based on the quality of the hand, the dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The program should then reevaluate the dealer's hand. (*Caution*: This is a difficult problem!)

11.6 Modify the program developed in Exercise 11.5 so that it can handle the dealer's hand automatically, but the player is allowed to decide which cards of the player's hand to replace. The program should then evaluate both hands and determine who wins. Now, use this new program to play 20 games against the computer. Who wins more games, you or the computer? Have a friend play 20 games against the computer. Who wins more games? Based on the results of these games, refine your poker-playing program. (This, too, is a difficult problem.) Play 20 more games. Does your modified program play a better game?

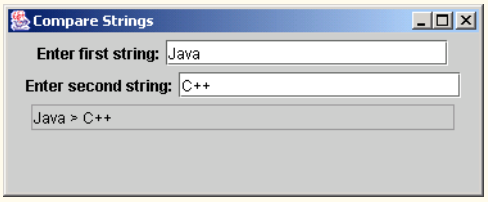
11.7 Write an application that uses `String` method `compareTo` to compare two strings input by the user. Output whether the first string is less than, equal to or greater than the second.

ANS:

```
1  // Exercise 11.7 Solution: CompareStrings.java
2  // Program compares two strings.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class CompareStrings extends JFrame {
8     private JLabel prompt1, prompt2;
9     private JTextField inputField1, inputField2, outputField;
```

```
10
11    // constructor
12    public CompareStrings()
13    {
14       super( "Compare Strings" );
15
16       // create GUI components
17       prompt1 = new JLabel( "Enter first string:" );
18       prompt2 = new JLabel( "Enter second string:" );
19       inputField1 = new JTextField( 20 );
20       inputField2 = new JTextField( 20 );
21       outputField = new JTextField( 30 );
22       outputField.setEditable( false );
23
24       inputField2.addActionListener(
25
26          new ActionListener() {  // anonymous inner class
27
28             // compare Strings
29             public void actionPerformed( ActionEvent e )
30             {
31                String first = inputField1.getText();
32                String second = inputField2.getText();
33                int value = first.compareTo( second );
34
35                if ( value == 0 )
36                   outputField.setText( first + " == " + second );
37                else if ( value > 0 )
38                   outputField.setText( first + " > " + second );
39                else
40                   outputField.setText( first + " < " + second );
41             }
42
43          }  // end anonymous inner class
44
45       ); // end call to addActionListener
46
47       // add components to GUI
48       Container container = getContentPane();
49       container.setLayout( new FlowLayout() );
50       container.add( prompt1 );
51       container.add( inputField1 );
52       container.add( prompt2 );
53       container.add( inputField2 );
54       container.add( outputField );
55
56       setSize( 375, 150 );
57       setVisible( true );
58
59    }  // end constructor
60
61    // execute application
62    public static void main( String args[] )
63    {
```

```
64         CompareStrings application = new CompareStrings();
65         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
66      }
67
68   } // end class CompareStrings
```

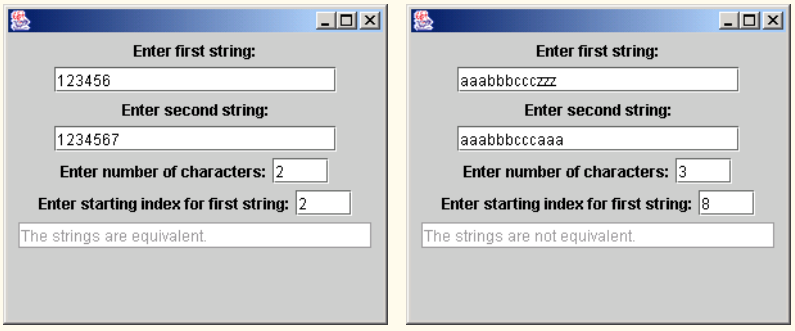


11.8 Write an application that uses `String` method `regionMatches` to compare two strings input by the user. The program should input the number of characters to be compared and the starting index of the comparison. The program should state whether the strings are equal. Ignore the case of the characters when performing the comparison.

ANS:

```
1  // Exercise 11.8 Solution: Compare.java
2  // Program compares two Strings.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Compare extends JFrame {
8     private JLabel prompt1, prompt2, promptNumber, promptIndex;
9     private JTextField input1, input2, inputNumber, inputIndex, status;
10
11    // set up GUI
12    public Compare()
13    {
14       prompt1 = new JLabel( "Enter first string:" );
15       prompt2 = new JLabel( "Enter second string:" );
16       promptNumber = new JLabel( "Enter number of characters:" );
17       promptIndex =
18          new JLabel( "Enter starting index for first string:" );
19
20       input1 = new JTextField( 20 );
21       input2 = new JTextField( 20 );
22       inputNumber = new JTextField( 4 );
23       inputIndex = new JTextField( 4 );
24       status = new JTextField( 25 );
25       status.setEnabled( false );
26
27       inputIndex.addActionListener(
28
29          new ActionListener() {  // anonymous inner class
30
```

```
31             // compare Strings
32             public void actionPerformed( ActionEvent event )
33             {
34                String first = input1.getText().toLowerCase();
35                String second = input2.getText().toLowerCase();
36                int number = Integer.parseInt( inputNumber.getText() );
37                int index = Integer.parseInt( inputIndex.getText() );
38                status.setText( "" );
39
40                if ( first.regionMatches(
41                   true, index, second, index, number ) )
42                   status.setText( "The strings are equivalent." );
43
44                else
45                   status.setText( "The strings are not equivalent." );
46             }
47
48          }  // end anonymous inner class
49
50       ); // end call to addActionListener
51
52       Container container = getContentPane();
53       container.setLayout( new FlowLayout() );
54       container.add( prompt1 );
55       container.add( input1 );
56       container.add( prompt2 );
57       container.add( input2 );
58       container.add( promptNumber );
59       container.add( inputNumber );
60       container.add( promptIndex );
61       container.add( inputIndex );
62       container.add( status );
63
64       setSize( 300, 250 );   // set the window size
65       setVisible( true );    // show the window
66    }
67
68    public static void main( String args[] )
69    {
70       Compare application = new Compare();
71       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
72    }
73
74 }  // end class Compare
```



11.9 Write an application that uses random number generation to create sentences. Use four arrays of strings called `article`, `noun`, `verb` and `preposition`. Create a sentence by selecting a word at random from each array in the following order: `article`, `noun`, `verb`, `preposition`, `article` and `noun`. As each word is picked, concatenate it to the previous words in the sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The program should generate 20 sentences and output them to a text area.

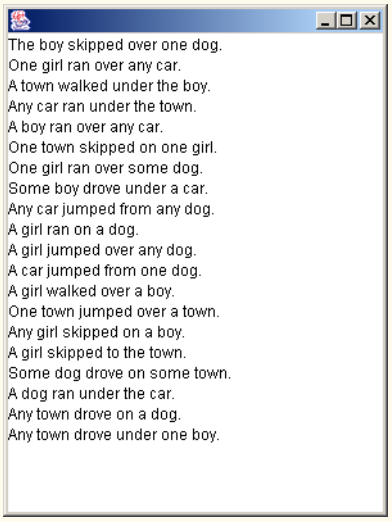
The article array should contain the articles `"the"`, `"a"`, `"one"`, `"some"` and `"any"`; the noun array should contain the nouns `"boy"`, `"girl"`, `"dog"`, `"town"` and `"car"`; the verb array should contain the verbs `"drove"`, `"jumped"`, `"ran"`, `"walked"` and `"skipped"`; the preposition array should contain the prepositions `"to"`, `"from"`, `"over"`, `"under"` and `"on"`.

After the preceding program is written, modify the program to produce a short story consisting of several of these sentences. (How about the possibility of a random term paper writer?)

ANS:

```
1  // Exercise 11.9 Solution: Sentences.java
2  // Program randomly generates sentences.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Sentences extends JFrame {
8     private JTextArea display;
9
10    // set up GUI and create sentence
11    public Sentences()
12    {
13       String article[] = { "the ", "a ", "one ", "some ", "any " },
14          noun[] = { "boy", "girl", "dog", "town", "car" },
15          verb[] = { " drove", " jumped", " ran", " walked", " skipped" },
16          preposition[] =
17             { " to ", " from ", " over ", " under ", " on " };
18
19       display = new JTextArea( 10, 30 );
20
21       // randomly create sentence
22       for ( int j = 1; j <= 20; j++ ) {
23          int article1 = ( int )( Math.random() * 5 );
24          int noun1 = ( int )( Math.random() * 5 );
25          int verb1 = ( int )( Math.random() * 5 );
```

```
26            int preposition2 = ( int )( Math.random() * 5 );
27            int article2 = ( int )( Math.random() * 5 );
28            int noun2 = ( int )( Math.random() * 5 );
29
30            StringBuffer buffer = new StringBuffer();
31
32            // concatenate words and add period
33            buffer.append( article[ article1 ] + noun[ noun1 ] +
34               verb[ verb1 ] + preposition[ preposition2 ] +
35               article[ article2 ] + noun[ noun2 ] + ".\n" );
36
37            // capitalize first letter
38            buffer.setCharAt(
39               0, Character.toUpperCase( buffer.charAt( 0 ) ) );
40            display.append( buffer.toString() );
41         }
42
43         Container container = getContentPane();
44         container.add( display );
45
46         setSize( 300, 400 );   // set the window size
47         setVisible( true );    // show the window
48      }
49
50      // execute application
51      public static void main( String args[] )
52      {
53         Sentences application = new Sentences();
54         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
55      }
56
57   }  // end class Sentences
```



11.10 *(Limericks)* A limerick is a humorous five-line verse in which the first and second lines rhyme with the fifth, and the third line rhymes with the fourth. Using techniques similar to those developed in Exercise 11.9, write a Java program that produces random limericks. Polishing this program to produce good limericks is a challenging problem, but the result will be worth the effort!

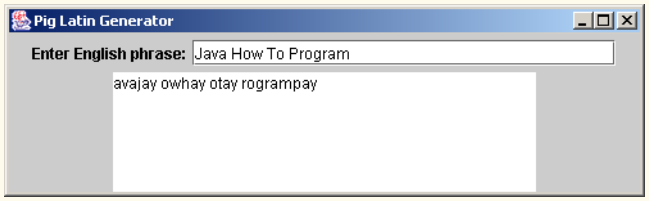
11.11 *(Pig Latin)* Write an application that encodes English language phrases into pig Latin. Pig Latin is a form of coded language. Many variations exist in the methods used to form pig Latin phrases. For simplicity, use the following algorithm:

To form a pig Latin phrase from an English language phrase, tokenize the phrase into words with an object of class `StringTokenizer`. To translate each English word into a pig Latin word, place the first letter of the English word at the end of the word and add the letters "ay." Thus, the word "jump" becomes "umpjay," the word "the" becomes "hetay," and the word "computer" becomes "omputercay." Blanks between words remain as blanks. Assume the following: The English phrase consists of words separated by blanks, there are no punctuation marks and all words have two or more letters. Method `printLatinWord` should display each word. Each token returned from `nextToken` is passed to method `printLatinWord` to print the pig Latin word. Enable the user to input the sentence. Keep a running display of all the converted sentences in a text area.

ANS:

```
1  // Exercise 11.11 Solution: PigLatin.java
2  // Program translates English to Pig Latin.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class PigLatin extends JFrame {
9     private JLabel prompt;
10    private JTextField inputField;
11    private JTextArea outputArea;
12    private int count;
13
14    // constructor
15    public PigLatin()
16    {
17       super( "Pig Latin Generator" );
18       prompt = new JLabel( "Enter English phrase:" );
19       inputField = new JTextField( 30 );
20       inputField.addActionListener(
21
22          new ActionListener() {  // anonymous inner class
23
24             // translate user input
25             public void actionPerformed( ActionEvent event )
26             {
27                String inputString = event.getActionCommand().toString();
28                StringTokenizer tokens =
29                   new StringTokenizer( inputString );
30
31                count = tokens.countTokens();
32
33                while ( tokens.hasMoreTokens() ) {
34                   count--;
```

```
35                      printLatinWord( tokens.nextToken() );
36                   }
37                }
38
39             }  // end anonymous inner class
40
41          ); // end call to addActionListener
42
43          outputArea = new JTextArea( 10, 30 );
44          outputArea.setEditable( false );
45
46          // add components to GUI
47          Container container = getContentPane();
48          container.setLayout( new FlowLayout() );
49          container.add( prompt );
50          container.add( inputField );
51          container.add( outputArea );
52
53          setSize( 500, 150 );
54          setVisible( true );
55
56       }  // end constructor
57
58       // translate English into Pig Latin
59       private void printLatinWord( String token )
60       {
61          char letters[] = token.toCharArray();
62          StringBuffer translation = new StringBuffer();
63
64          translation.append( letters, 1, letters.length - 1 ) ;
65          translation.append( Character.toLowerCase( letters[ 0 ] ) );
66          translation.append( "ay" );
67
68          outputArea.append( translation.toString() + " " );
69
70          if ( count == 0 )
71             outputArea.append( "\n" );
72
73       }  // end method printLatinWord
74
75       public static void main( String args[] )
76       {
77          PigLatin application = new PigLatin();
78          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
79       }
80
81    }  // end class PigLatin
```

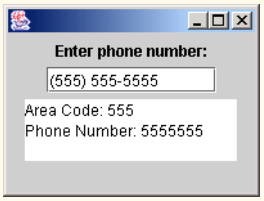


11.12 Write an application that inputs a telephone number as a string in the form `(555) 555-5555`. The program should use an object of class `StringTokenizer` to extract the area code as a token, the first three digits of the phone number as a token and the last four digits of the phone number as a token. The seven digits of the phone number should be concatenated into one string. The program should convert the area code string to `int` (remember `parseInt`!) and convert the phone number string to `long`. Both the area code and the phone number should be printed. Remember that you will have to change delimiter characters during the tokenization process.

ANS:

```
1  // Exercise 11.12 Solution: Phone.java
2  // Program separates phone area code from the other seven digits.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class Phone extends JFrame {
9     private JTextField inputField;
10    private JLabel prompt;
11    private JTextArea phoneOut;
12
13    // set up GUI
14    public Phone()
15    {
16       prompt = new JLabel( "Enter phone number:" );
17
18       inputField = new JTextField( 12 );
19       inputField.addActionListener(
20
21          new ActionListener() {  // anonymous inner class
22
23             // tokenize phone number
24             public void actionPerformed( ActionEvent event )
25             {
26                StringTokenizer tokenizer = new StringTokenizer(
27                   inputField.getText() );
28
29                int areaCode =
30                   Integer.parseInt( tokenizer.nextToken( "()" ) );
31                String number =
32                   new Long( tokenizer.nextToken( ") -") ).toString();
33                number += new Long( tokenizer.nextToken("-") ).toString();
34
35                long phoneNumber = Long.parseLong( number );
```

```
36
37                phoneOut.setText( "Area Code: " + areaCode +
38                   "\nPhone Number: " + phoneNumber );
39             }
40
41          }  // end anonymous inner class
42
43       ); // end call to addActionListener
44
45       phoneOut = new JTextArea( 3, 15 );
46
47       Container container = getContentPane();
48       container.setLayout( new FlowLayout() );
49
50       container.add( prompt );
51       container.add( inputField );
52       container.add( phoneOut );
53
54       setSize( 200, 150 );
55       setVisible( true );
56
57    } // end constructor
58
59    public static void main( String args[] )
60    {
61       Phone application = new Phone();
62       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
63    }
64
65 } // end class Phone
```



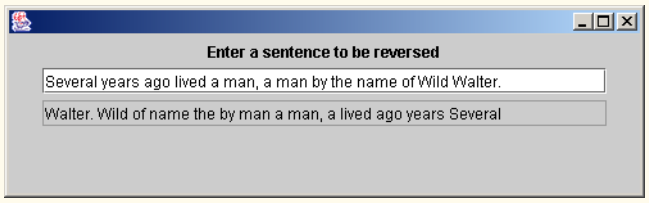
11.13 Write an application that inputs a line of text, tokenizes the line with an object of class `StringTokenizer` and outputs the tokens in reverse order. Use space characters as delimiters.

ANS:

```
1  // Exercise 11.13 Solution: TokenTest.java
2  // Program displays tokens in reverse order.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class TokenTest extends JFrame{
9     private JLabel prompt;
```

```
10    private JTextField inputField, outputField;
11
12    // set up GUI
13    public TokenTest()
14    {
15       prompt = new JLabel( "Enter a sentence to be reversed" );
16       inputField = new JTextField( 40 );
17       inputField.addActionListener(
18
19          new ActionListener() {  // anonymous inner class
20
21             // display tokens in reverse order
22             public void actionPerformed( ActionEvent event )
23             {
24                String stringToTokenize = inputField.getText();
25                StringTokenizer tokens =
26                   new StringTokenizer( stringToTokenize );
27                int count = tokens.countTokens();
28                String reverse[] = new String[ count ];
29                int index = count - 1;
30
31                while ( tokens.hasMoreTokens() )
32                   reverse[ index-- ] = tokens.nextToken();
33
34                outputField.setText( "" );
35
36                for ( int k = 0; k < count; k++ )
37                   outputField.setText( outputField.getText() +
38                      reverse[ k ] + " " );
39             }
40
41          }  // end anonymous inner class
42
43       ); // end call to addActionListener
44
45       Container container = getContentPane();
46       container.setLayout( new FlowLayout() );
47
48       outputField = new JTextField( 40 );
49       outputField.setEditable( false );
50       container.add( prompt );
51       container.add( inputField );
52       container.add( outputField );
53
54       setSize( 500, 150 );
55       setVisible( true );
56
57    } // end constructor
58
59    public static void main( String args[] )
60    {
61       TokenTest application = new TokenTest();
62       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
63    }
```

```
64
65 } // end class TokenTest
```



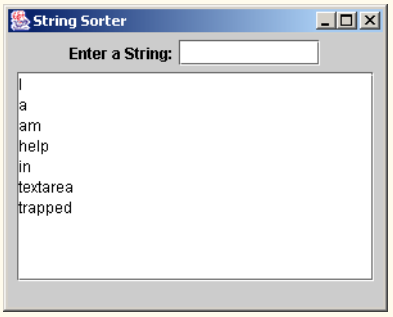
11.14 Use the string-comparison methods discussed in this chapter and the techniques for sorting arrays developed in Chapter 7 to write a program that alphabetizes a list of strings. Allow the user to enter the strings in a text field. Display the results in a text area.

ANS:

```
1  // Exercise 11.14 Solution: SortThem.java
2  // Program sorts strings.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class SortThem extends JFrame {
9     private JTextField inputField;
10    private JLabel prompt;
11    private JTextArea outputArea;
12    private JScrollPane scroller;
13
14    // sort Strings and set up GUI
15    public SortThem()
16    {
17       super( "String Sorter" );
18       inputField = new JTextField( 10 );
19       inputField.addActionListener(
20
21          new ActionListener() {  // anonymous inner class
22
23             // sort Strings
24             public void actionPerformed( ActionEvent e )
25             {
26                String newString = inputField.getText();
27                String oldString = outputArea.getText();
28                StringTokenizer tokenizer =
29                   new StringTokenizer( oldString, "\n" );
30                int number = tokenizer.countTokens(), count = 0;
31                String tokens[] = new String[ number + 1 ];
32
33                inputField.setText( "" );
34                outputArea.setText( "" );
35
```

```
36                while ( tokenizer.hasMoreTokens() && count < number )
37                   tokens[ count++ ] = tokenizer.nextToken();
38
39                tokens[ count ] = newString;
40                bubbleSort( tokens );
41
42                for ( int counter = 0; counter < tokens.length; counter++ )
43                   outputArea.append( tokens[ counter ] + "\n" );
44             }
45
46          }  // end anonymous inner class
47
48       ); // end call to addActionListener
49
50       // create GUI components...
51       outputArea = new JTextArea( 10, 25 );
52       scroller = new JScrollPane( outputArea );
53       prompt = new JLabel( "Enter a String:" );
54
55       // ...and add them to the GUI
56       Container container = getContentPane();
57       container.setLayout( new FlowLayout() );
58       container.add( prompt );
59       container.add( inputField );
60       container.add( scroller );
61
62       setSize( 300, 240 );
63       setVisible( true );
64
65    } // end constructor
66
67    // sort using bubble sort algorithm
68    private void bubbleSort( String strings[] )
69    {
70       for ( int pass = 1; pass < strings.length; pass++ )
71
72          for ( int comparison = 0; comparison < strings.length - pass;
73             comparison++ )
74
75             // starting at the first element in the array, if any two
76             // consecutive elements are not in ascending order switch them
77             if ( strings[ comparison ].compareTo(
78                strings[ comparison + 1 ] ) > 0 ) {
79                String temp = strings[ comparison ];
80                strings[ comparison ] = strings[ comparison + 1 ];
81                strings[ comparison + 1 ] = temp;
82             }
83    }
84
85    // execute application
86    public static void main( String args[] )
87    {
88       SortThem application = new SortThem();
89       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
90    }
```

```
91
92  }  // end class SortThem
```

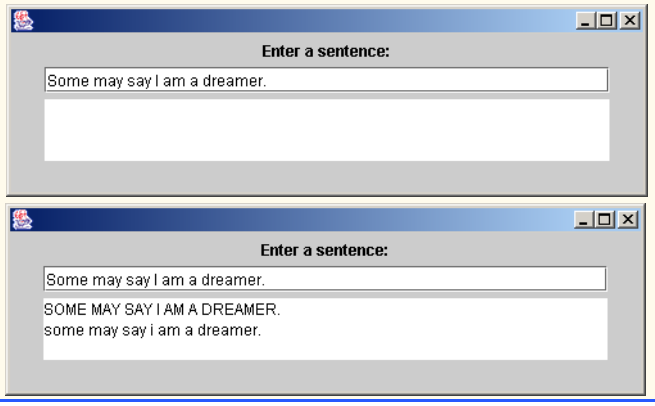


11.15 Write an application that inputs a line of text and outputs the text twice—once in all uppercase letters and once in all lowercase letters.

ANS:

```
1  // Exercise 11.15 Solution: Convert2.java
2  // Program outputs text in uppercase and lowercase.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Convert2 extends JFrame {
8     private JTextField inputField;
9     private JLabel prompt;
10    private JTextArea diplayArea;
11
12    // set up GUI
13    public Convert2()
14    {
15       inputField = new JTextField( 40 );
16       inputField.addActionListener(
17
18          new ActionListener() {  // anonymous inner class
19
20             // diplay converted text
21             public void actionPerformed( ActionEvent event )
22             {
23                diplayArea.setText( "" );
24                String uppercase = inputField.getText().toUpperCase();
25                String lowercase = inputField.getText().toLowerCase();
26                diplayArea.append( uppercase + "\n" );
27                diplayArea.append( lowercase + "\n" );
28             }
29
30          }  // end anonymous inner class
31
32       ); // end call to addActionListener
33
```

```
34          prompt = new JLabel( "Enter a sentence:" );
35          diplayArea = new JTextArea( 3, 40 );
36
37          Container container = getContentPane();
38          container.setLayout( new FlowLayout() );
39
40          container.add( prompt );
41          container.add( inputField );
42          container.add( diplayArea );
43
44          setSize( 500, 150 );
45          setVisible( true );
46
47       } // end constructor
48
49       public static void main( String args[] )
50       {
51          Convert2 application = new Convert2();
52          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
53       }
54
55    }  // end class Convert2
```



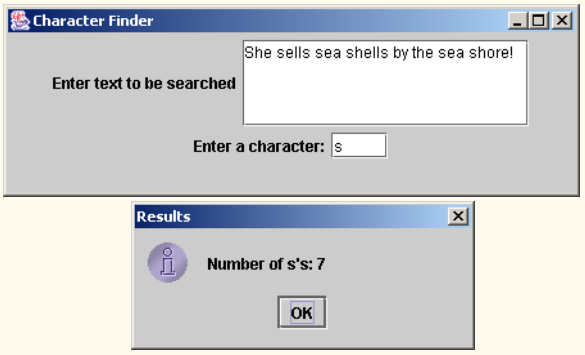
11.16 Write an application that inputs several lines of text and a search character and uses `String` method `indexOf` to determine the number of occurrences of the character in the text.

ANS:

```
1   // Exercise 11.16 Solution: Index.java
2   // Program outputs the number of times a search character was found.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class Index extends JFrame {
8      private JTextField inputKey;
9      private JLabel promptArea, promptKey;
10     private JTextArea inputArea;
```

```
11
12    // constructor
13    public Index()
14    {
15       super( "Character Finder" );
16
17       inputKey = new JTextField( 4 );
18       inputKey.addActionListener(
19
20          new ActionListener() {  // anonymous inner class
21
22             // search for input character
23             public void actionPerformed( ActionEvent event )
24             {
25                int count = 0, current = 0;
26                String inputKey = event.getActionCommand().toString();
27                char key = inputKey.charAt( 0 );
28                String inputString = inputArea.getText();
29                current = inputString.indexOf( key, 0 );
30
31                while ( current != -1 ) {
32                   count++;
33                   current = inputString.indexOf( key, current + 1 );
34                }
35
36                JOptionPane.showMessageDialog( null,
37                   "Number of " + key + "'s: " + count, "Results",
38                   JOptionPane.INFORMATION_MESSAGE );
39             }
40
41          }  // end anonymous inner class
42
43       ); // end call to addActionListener
44
45       promptArea = new JLabel( "Enter text to be searched" );
46       promptKey = new JLabel( "Enter a character:" );
47       inputArea = new JTextArea( 4, 20 );
48       JScrollPane scrollPane = new JScrollPane( inputArea );
49
50       // add components to GUI
51       Container container = getContentPane();
52       container.setLayout( new FlowLayout() );
53       container.add( promptArea );
54       container.add( scrollPane );
55       container.add( promptKey );
56       container.add( inputKey );
57
58       setSize( 450, 150 );
59       setVisible( true );
60
61    }  // end constructor
62
63    public static void main( String args[] )
64    {
```

```
65          Index application = new Index();
66          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
67       }
68
69    }  // end class Index
```



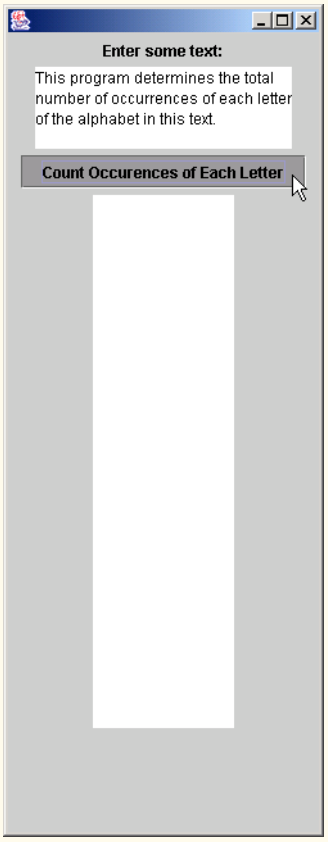
11.17 Write an application based on the program in Exercise 11.16 that inputs several lines of text and uses `String` method `indexOf` to determine the total number of occurrences of each letter of the alphabet in the text. Uppercase and lowercase letters should be counted together. Store the totals for each letter in an array, and print the values in tabular format after the totals have been determined.

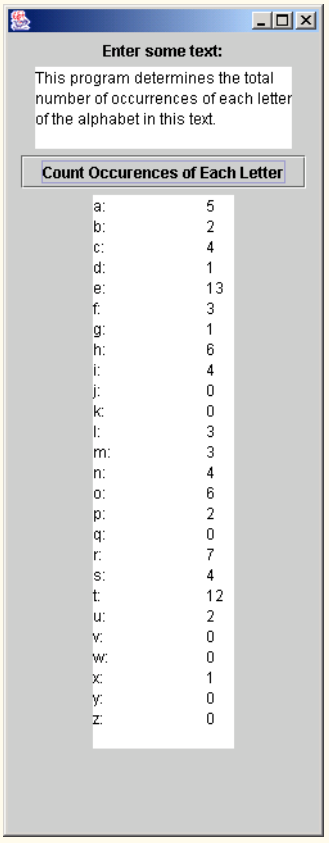
ANS:

```
1   // Exercise 11.17 Solution: Index2.java
2   // Program outputs the number of times each character of
3   // the alphabet was found.
4   import java.awt.*;
5   import java.awt.event.*;
6   import java.util.*;
7   import javax.swing.*;
8
9   public class Index2 extends JFrame {
10     private JLabel prompt;
11     private JTextArea inputArea, outputArea;
12     private JButton button;
13
14     // set up GUI
15     public Index2()
16     {
17        prompt = new JLabel( "Enter some text:" );
18        inputArea = new JTextArea( 4, 18 );
19
20        button = new JButton( "Count Occurences of Each Letter" );
21        button.addActionListener(
22
23           new ActionListener () {  // anonymous inner class
24
25              // count occurences of each letter
26              public void actionPerformed( ActionEvent event )
27              {
```

```
28                outputArea.setText( "" );
29
30                String alphabet = "abcdefghijklmnopqrstuvwxyz";
31                String inputString = inputArea.getText().toLowerCase();
32
33                int last = -2, current = 0;
34                int count[] = new int[ 26 ];
35
36                // count occurences
37                for ( int j = 0; j < alphabet.length(); j++ ) {
38
39                   for ( int k = 0; k < inputString.length() - 1; k++ ) {
40
41                      current = inputString.indexOf(
42                         alphabet.charAt( j ), k );
43
44                      if ( current != -1 && current != last ) {
45                         last = current;
46                         count[ j ]++;
47                         k = current;  // optimization
48                      }
49                   }
50
51                   last = -2;
52                }
53
54                // display results
55                for ( int y = 0; y < count.length; y++ )
56                   outputArea.append( alphabet.charAt( y ) +
57                      ":\t" + count[ y ] + "\n" );
58             }
59
60          }  // end anonymous inner class
61
62       ); // end call to addActionListener
63
64       outputArea = new JTextArea( 26, 10 );
65
66       Container container = getContentPane();
67       container.setLayout( new FlowLayout() );
68
69       container.add( prompt );
70       container.add( inputArea );
71       container.add( button );
72       container.add( outputArea );
73
74       setSize( 250, 650 );
75       setVisible( true );
76
77    } // end constructor
78
79    public static void main( String args[] )
80    {
81       Index2 application = new Index2();
```

```
82          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
83       }
84
85    }  // end class Index2
```





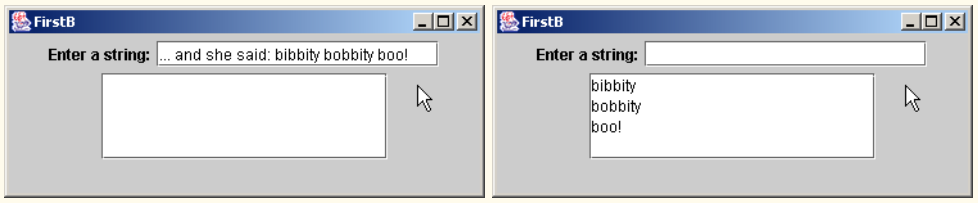
11.18 Write an application that reads a line of text, tokenizes the line using space characters as delimiters and outputs only those words beginning with the letter "b." The results should appear in a text area.

ANS:

```
1   // Exercise 11.18 Solution: FirstB.java
2   // Program outputs strings that begin with "b"
3   import java.awt.*;
4   import java.awt.event.*;
5   import java.util.*;
6   import javax.swing.*;
7
8   public class FirstB extends JFrame {
9      private JTextField inputField;
10     private JLabel prompt;
```

```
11    private JTextArea display;
12
13    // constructor
14    public FirstB()
15    {
16       super( "FirstB" );
17       inputField = new JTextField( 20 );
18       inputField.addActionListener(
19
20          new ActionListener() {  // anonymous inner class
21
22             // search for Strings beginning with "b"
23             public void actionPerformed( ActionEvent event )
24             {
25                String inputString = event.getActionCommand().toString();
26                StringTokenizer tokens =
27                   new StringTokenizer( inputString );
28
29                while ( tokens.hasMoreTokens() ) {
30                   String test = tokens.nextToken();
31
32                   if( test.startsWith( "b" ) )
33                      display.append( test + "\n" );
34                }
35
36                inputField.setText( "" );
37             }
38
39          }  // end anonymous inner class
40
41       ); // end call to addActionListener
42
43       prompt = new JLabel( "Enter a string:" );
44       display = new JTextArea( 4, 20 );
45       display.setEditable( false );
46
47       JScrollPane displayScrollPane = new JScrollPane( display );
48
49       // add components to GUI
50       Container container = getContentPane();
51       container.setLayout( new FlowLayout() );
52       container.add( prompt );
53       container.add( inputField );
54       container.add( displayScrollPane );
55
56       setSize( 375, 150 );
57       setVisible( true );
58
59    }  // end constructor
60
61    public static void main( String args[] )
62    {
63       FirstB application = new FirstB();
64       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
65    }
```

```
66
67   }  // end class FirstB
```

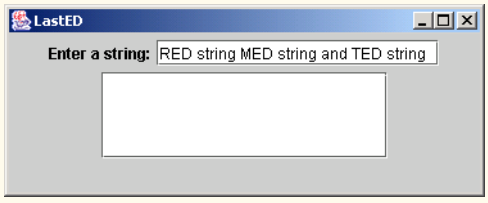


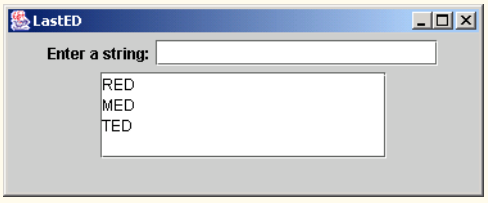
11.19 Write an application that reads a line of text, tokenizes it using space characters as delimiters and outputs only those words ending with the letters "ED." The results should appear in a text area.

ANS:

```
1   // Exercise 11.19 Solution: LastED.java
2   // Program outputs strings that end with "ED"
3   import java.awt.*;
4   import java.awt.event.*;
5   import java.util.*;
6   import javax.swing.*;
7
8   public class LastED extends JFrame {
9      private JTextField inputField;
10     private JLabel prompt;
11     private JTextArea display;
12
13     // constructor
14     public LastED()
15     {
16        super( "LastED" );
17        inputField = new JTextField( 20 );
18        inputField.addActionListener(
19
20           new ActionListener() {  // anonymous inner class
21
22              // search for Strings ending with "ED"
23              public void actionPerformed( ActionEvent event )
24              {
25                 String inputString = event.getActionCommand().toString();
26                 StringTokenizer tokens =
27                    new StringTokenizer( inputString );
28
29                 while ( tokens.hasMoreTokens() ) {
30                    String test = tokens.nextToken();
31
32                    if( test.endsWith( "ED" ) )
33                       display.append( test + "\n" );
34                 }
35
36                 inputField.setText( "" );
37              }
38
```

```
39           }  // end anonymous inner class
40
41        ); // end call to addActionListener
42
43        prompt = new JLabel( "Enter a string:" );
44        display = new JTextArea( 4, 20 );
45        display.setEditable( false );
46
47        JScrollPane displayScrollPane = new JScrollPane( display );
48
49        // add components to GUI
50        Container container = getContentPane();
51        container.setLayout( new FlowLayout() );
52        container.add( prompt );
53        container.add( inputField );
54        container.add( displayScrollPane );
55
56        setSize( 375, 150 );
57        setVisible( true );
58
59     }  // end constructor
60
61     public static void main( String args[] )
62     {
63        LastED application = new LastED();
64        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
65     }
66
67  }  // end class LastED
```



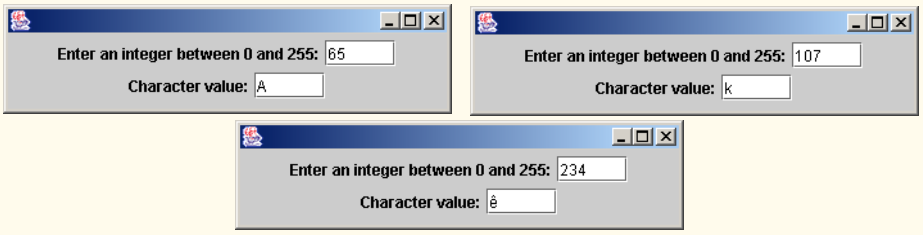


11.20 Write an application that inputs an integer code for a character and displays the corresponding character. Modify this program so that it generates all possible three-digit codes in the range from 000 to 255 and attempts to print the corresponding characters. Display the results in a text area.

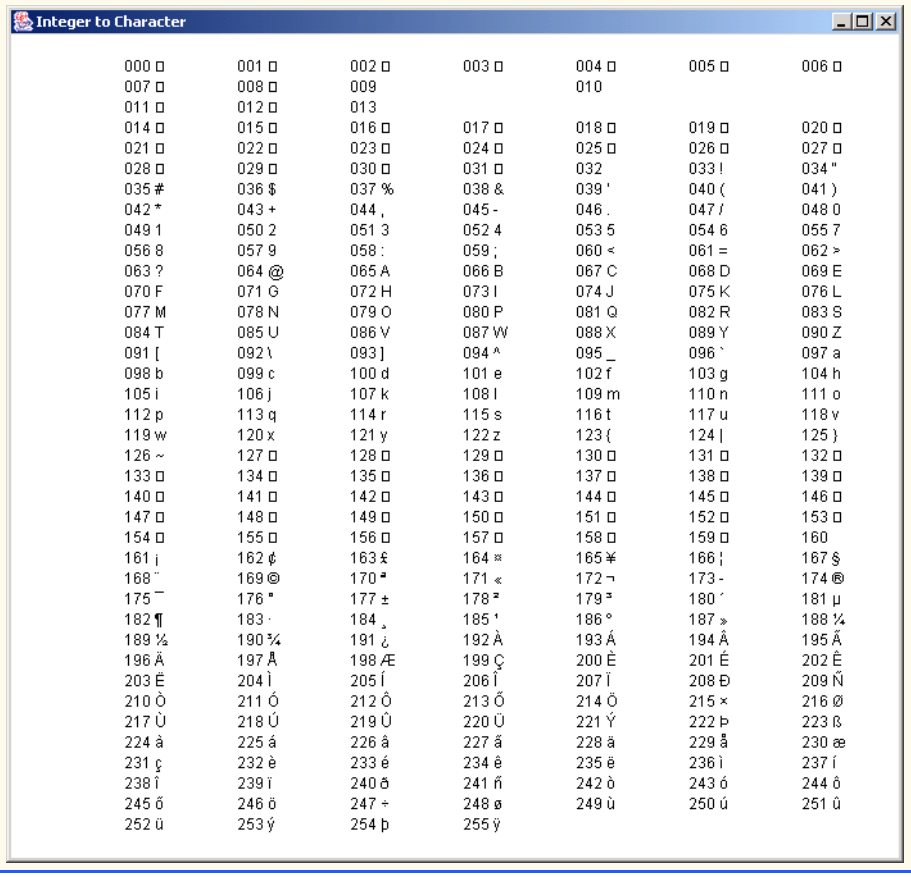
ANS:

```
1  // Exercise 11.20 Solution: IntegerToChar.java
2  // Program converts from an integer value to a character
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class IntegerToChar extends JFrame {
8     private JLabel prompt, value;
9     private JTextField inputField, charField;
```

```
10
11    // set up GUI
12    public IntegerToChar()
13    {
14       prompt = new JLabel( "Enter an integer between 0 and 255:" );
15       value = new JLabel( "Character value:" );
16       inputField = new JTextField( 5 );
17       inputField.addActionListener(
18
19          new ActionListener() {
20
21             public void actionPerformed( ActionEvent event )
22             {
23                int input = Integer.parseInt( inputField.getText() );
24                if ( 0 < input && input <= 255 )
25                   charField.setText( String.valueOf( (char)input ) );
26             }
27
28          } // end anonymous class
29
30       ); // end call to addActionListener
31
32       charField = new JTextField( 5 );
33
34       Container container = getContentPane();
35       container.setLayout( new FlowLayout() );
36       container.add( prompt );
37       container.add( inputField );
38       container.add( value );
39       container.add( charField );
40
41       setSize(350,85);
42       setVisible( true );
43
44    }  // end constructor
45
46    public static void main( String args[] )
47    {
48       IntegerToChar application = new IntegerToChar();
49       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
50    }
51
52 }  // end class IntegerToChar
```



```
1  // Exercise 11.20 Solution: IntegerToChar2.java
2  // Program converts from an integer value to a character
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.text.*;
6  import javax.swing.*;
7
8  public class IntegerToChar2 extends JFrame {
9     private JTextArea outputArea;
10
11    // set up GUI
12    public IntegerToChar2()
13    {
14       super( "Integer to Character" );
15
16       outputArea = new JTextArea();
17       DecimalFormat threeDigits = new DecimalFormat( "000" );
18       String output = "";
19
20       for ( int i = 0; i <= 255; i++ ) {
21
22          if ( i % 7 == 0 )
23             outputArea.append( "\n" );
24
25          outputArea.append( "\t" + threeDigits.format( i ) + " " +
26             String.valueOf( ( char ) i ) );
27       }
28
29       Container container = getContentPane();
30       container.add( outputArea );
31
32       setSize( 700, 670 );
33       setVisible( true );
34
35    }  // end constructor
36
37    public static void main( String args[] )
38    {
39       IntegerToChar2 application = new IntegerToChar2();
40       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
41    }
42
43 }  // end class IntegerToChar2
```

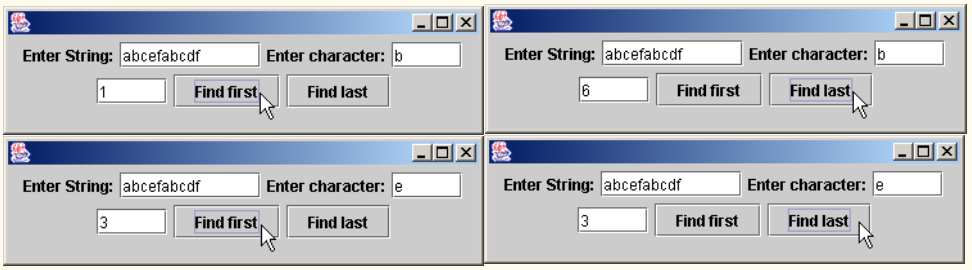


11.21 Write your own versions of `String` search methods `indexOf` and `lastIndexOf`.
ANS:

```
1  // Exercise 11.21 Solution: NewSearch.java
2  // Program uses our own versions of String indexing methods
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class NewSearch extends JFrame {
8     private JTextField inputString, inputChar, outputField;
9     private JButton indexOf, lastIndexOf;
10    private JLabel prompt1, prompt2;
11
12    // set up GUI
13    public NewSearch()
14    {
15       inputString = new JTextField( 10 );
16       inputChar = new JTextField( 5 );
17       outputField = new JTextField( 5 );
```

```
18
19          prompt1 = new JLabel( "Enter String:" );
20          prompt2 = new JLabel( "Enter character:" );
21
22          indexOf = new JButton( "Find first" );
23          indexOf.addActionListener(
24
25             new ActionListener() {  // anonymous inner class
26
27                public void actionPerformed( ActionEvent event ) {
28                   int result = ourIndexOf( inputString.getText(),
29                      inputChar.getText().charAt( 0 ) );
30                   outputField.setText( String.valueOf( result ) );
31                }
32             }
33          );
34
35          lastIndexOf = new JButton( "Find last" );
36          lastIndexOf.addActionListener(
37
38             new ActionListener() {  // anonymous inner class
39
40                public void actionPerformed( ActionEvent event ) {
41                   int result = ourLastIndexOf( inputString.getText(),
42                      inputChar.getText().charAt( 0 ) );
43                   outputField.setText( String.valueOf( result ) );
44                }
45             }
46          );
47
48          Container container = getContentPane();
49          container.setLayout( new FlowLayout() );
50          container.add( prompt1 );
51          container.add( inputString );
52          container.add( prompt2 );
53          container.add( inputChar );
54          container.add( outputField );
55          container.add( indexOf );
56          container.add( lastIndexOf );
57
58          setSize(375,100);
59          setVisible( true );
60
61       }  // end constructor
62
63       public int ourIndexOf( String input, char search )
64       {
65          // loop through the characters
66          for( int i = 0; i < input.length(); i++ ) {
67             if ( input.charAt( i ) == search )
68                return i;
69          }
70
```

```
71          // if we do not find it
72          return -1;
73       }
74
75       public int ourLastIndexOf( String input, char search )
76       {
77          // loop through the characters in reverse
78          for( int i = input.length() - 1; i >= 0; i-- ) {
79             if ( input.charAt( i ) == search )
80                return i;
81          }
82
83          // if we do not find it
84          return -1;
85       }
86
87       public static void main( String args[] )
88       {
89          NewSearch application = new NewSearch();
90          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
91       }
92
93    }  // end class NewSearch
```



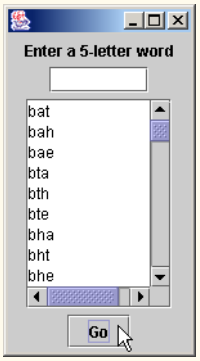
11.22 Write a program that reads a five-letter word from the user and produces all possible three-letter words that can be derived from the letters of the five-letter word. For example, the three-letter words produced from the word "bathe" include "ate," "bat," "bet," "tab," "hat," "the" and "tea."

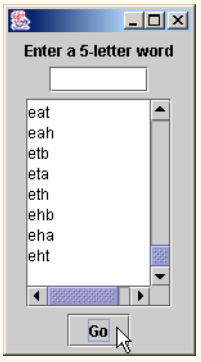
ANS:

```
1  // Exercise 11.22 Solution: ThreeLetter.java
2  // Finding all 3-letter words in a 5-letter word. There are
3  // 60 such combinations.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class ThreeLetter extends JFrame {
9     private JLabel label1;
10    private JTextField inputTextField;
11    private JTextArea outputTextArea;
12    private JButton goButton;
13
```

```
14    public ThreeLetter()
15    {
16       label1 = new JLabel( "Enter a 5-letter word" );
17
18       inputTextField = new JTextField( 7 );
19       inputTextField.addActionListener(
20
21          new ActionListener() {  // anonymous inner class
22
23             public void actionPerformed( ActionEvent event )
24             {
25                generate();
26             }
27          }
28       );
29
30       outputTextArea = new JTextArea( 10, 10 );
31       outputTextArea.setEditable( false );
32
33       goButton = new JButton( "Go" );
34       goButton.addActionListener(
35
36          new ActionListener() {  // anonymous inner class
37
38             public void actionPerformed( ActionEvent event )
39             {
40                generate();
41             }
42          }
43       );
44
45       JScrollPane displayScrollPane = new JScrollPane( outputTextArea );
46
47       // add components to GUI
48       Container container = getContentPane();
49       container.setLayout( new FlowLayout() );
50       container.add( label1 );
51       container.add( inputTextField );
52       container.add( displayScrollPane );
53       container.add( goButton );
54
55       setSize( 150, 275 );
56       setVisible( true );
57
58    } // end constructor
59
60    public void generate()
61    {
62       while ( inputTextField.getText().length() != 5 )
63          JOptionPane.showMessageDialog( this, "Enter a 5-letter word" );
64
65       String word = inputTextField.getText();
66       String temp = "";
67
```

```
68          // choose the first letter
69          for ( int first = 0; first <= 4; first++ ) {
70
71             // choose the second letter
72             for ( int second = 0; second <= 4; second++ ) {
73
74                // each letter can only be used once in a word
75                if ( second == first )
76                   continue;
77
78                // choose the third letter
79                for ( int third = 0; third <= 4; third++ ) {
80
81                   // each letter can only be used once in a word
82                   if ( third == first || third == second )
83                      continue;
84
85                   temp += String.valueOf( word.charAt( first ) ) +
86                      word.charAt( second ) + word.charAt( third ) + "\n";
87
88                } // end third for
89
90             } // end second for
91
92          } // end first for
93
94          outputTextArea.setText( temp );
95          inputTextField.setText( "" );
96
97       } // end method generate
98
99       public static void main( String args[] )
100      {
101         ThreeLetter application = new ThreeLetter();
102         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
103      }
104
105   } // end class ThreeLetter
```





SPECIAL SECTION: ADVANCED STRING MANIPULATION EXERCISES

The preceding exercises are keyed to the text and designed to test your understanding of fundamental string-manipulation concepts. This section includes a collection of intermediate and advanced string-manipulation exercises. You should find these problems challenging, yet entertaining. The problems vary considerably in difficulty. Some require an hour or two of program writing and implementation. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects.

11.23 *(Text Analysis)* The availability of computers with string-manipulation capabilities has resulted in some rather interesting approaches to analyzing the writings of great authors. Much attention has been focused on whether William Shakespeare ever lived. Some scholars believe there is substantial evidence indicating that Christopher Marlowe or other authors actually penned the masterpieces attributed to Shakespeare. Researchers have used computers to find similarities in the writings of these two authors. This exercise examines three methods for analyzing texts with a computer.

a) Write an application that reads several lines of text from the keyboard and prints a table indicating the number of occurrences of each letter of the alphabet in the text. For example, the phrase

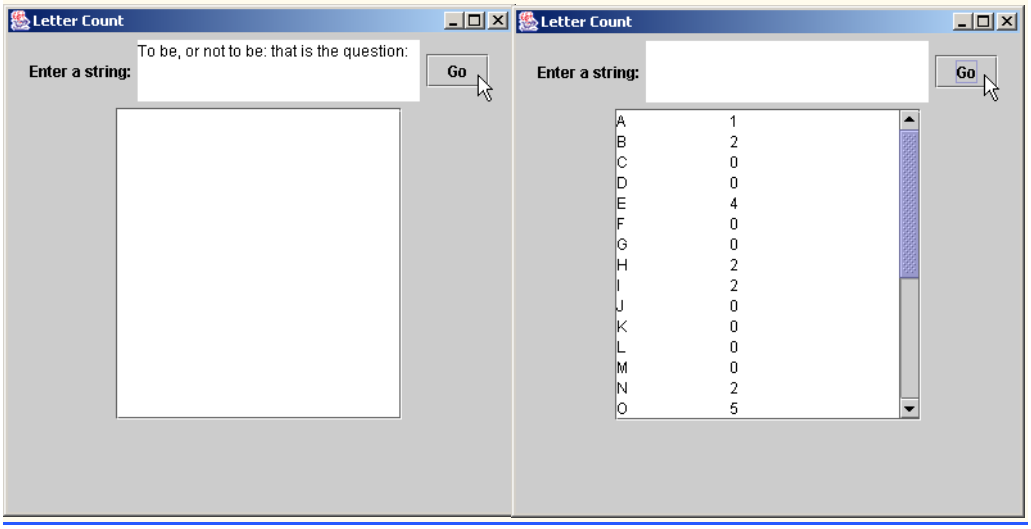
```
To be, or not to be: that is the question:
```

contains one "a," two "b's," no "c's," etc.

ANS:

```
1  // Exercise 11.23a Solution: LetterCount.java
2  // Program counts the occurances of each letter in a string
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class LetterCount extends JFrame {
9     private JTextArea inputField;
10    private JLabel prompt;
11    private JTextArea display;
12    private JButton goButton;
13
14    // constructor
15    public LetterCount()
16    {
17       super( "Letter Count" );
18       inputField = new JTextArea( 3, 20 );
19
20       goButton = new JButton( "Go" );
21       goButton.addActionListener(
22
23          new ActionListener() {  // anonymous inner class
24
25             public void actionPerformed( ActionEvent event )
26             {
27                String input = inputField.getText();
28                int[] letters = new int[26];
29
```

```
30                // loop through the string
31                for ( int i = 0; i < input.length(); i++ )
32
33                   if ( Character.isLetter( input.charAt( i ) ) )
34                      letters[ Character.toUpperCase(
35                         input.charAt( i ) ) - 65 ]++;
36
37                String output = "";
38
39                for ( int i = 0; i < 26; i++ )
40                   output += String.valueOf(
41                      ( char )( i + 65 ) ) + "\t" + letters[ i ] + "\n";
42
43                display.setText( output );
44                inputField.setText( "" );
45             }
46
47          }  // end anonymous inner class
48
49       ); // end call to addActionListener
50
51       prompt = new JLabel( "Enter a string:" );
52       display = new JTextArea( 15, 20 );
53       display.setEditable( false );
54
55       JScrollPane displayScrollPane = new JScrollPane( display );
56
57       // add components to GUI
58       Container container = getContentPane();
59       container.setLayout( new FlowLayout() );
60       container.add( prompt );
61       container.add( inputField );
62       container.add( goButton );
63       container.add( displayScrollPane );
64
65       setSize( 400, 400 );
66       setVisible( true );
67
68    }  // end constructor
69
70    public static void main( String args[] )
71    {
72       LetterCount application = new LetterCount();
73       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
74    }
75
76 }  // end class LetterCount
```



b) Write an application that reads several lines of text and prints a table indicating the number of one-letter words, two-letter words, three-letter words, etc., appearing in the text. For example, Fig. 11.32 shows the counts for the phrase

```
Whether 'tis nobler in the mind to suffer
```

Word length	Occurrences
1	0
2	2
3	1
4	2 (including 'tis)
5	0
6	2
7	1

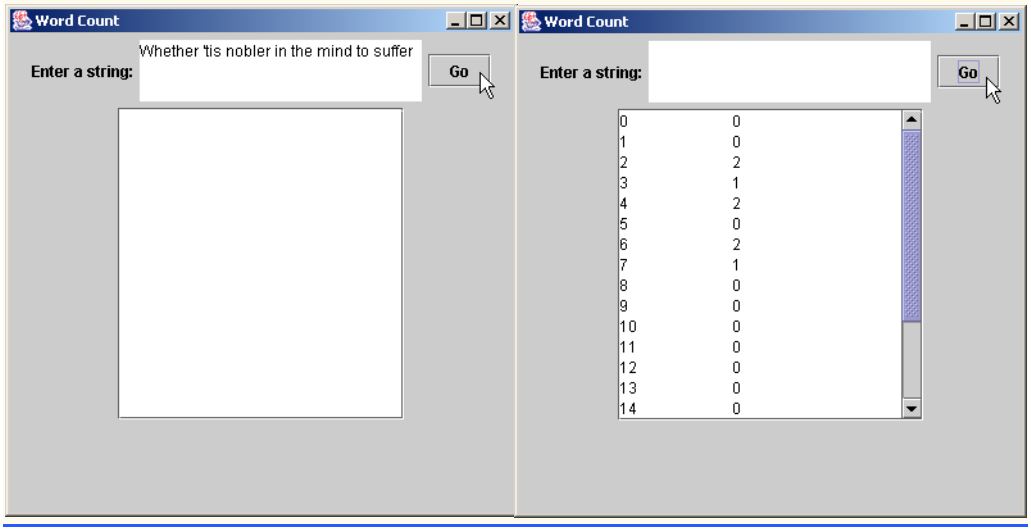
Fig. 11.32 Word-length counts for the string "Whether 'tis nobler in the mind to suffer".

ANS:

```
1  // Exercise 11.23b Solution: WordCount.java
2  // Program counts the number of words of each length in a string
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
```

```
 8  public class WordCount extends JFrame {
 9     private JTextArea inputField;
10     private JLabel prompt;
11     private JTextArea display;
12     private JButton goButton;
13
14     // constructor
15     public WordCount()
16     {
17        super( "Word Count" );
18        inputField = new JTextArea( 3, 20 );
19
20        goButton = new JButton( "Go" );
21        goButton.addActionListener(
22
23           new ActionListener() {  // anonymous inner class
24
25              public void actionPerformed( ActionEvent event )
26              {
27                 String input = inputField.getText();
28                 int[] words = new int[20];
29
30                 StringTokenizer token = new StringTokenizer( input, " " );
31
32                 // split the string into words
33                 while( token.hasMoreTokens() ) {
34                    String word = token.nextToken();
35                    words[ word.length() ]++;
36                 }
37
38                 String output = "";
39
40                 // output the results
41                 for( int i = 0; i < 20; i++ )
42                    output += String.valueOf( i ) + "\t" +
43                       words[ i ] + "\n";
44
45                 display.setText( output );
46                 inputField.setText( "" );
47              }
48
49           }  // end anonymous inner class
50
51        ); // end call to addActionListener
52
53        prompt = new JLabel( "Enter a string:" );
54        display = new JTextArea( 15, 20 );
55        display.setEditable( false );
56
57        JScrollPane displayScrollPane = new JScrollPane( display );
58
59        // add components to GUI
60        Container container = getContentPane();
61        container.setLayout( new FlowLayout() );
```

```
62          container.add( prompt );
63          container.add( inputField );
64          container.add( goButton );
65          container.add( displayScrollPane );
66
67          setSize( 400, 400 );
68          setVisible( true );
69
70       }  // end constructor
71
72       public static void main( String args[] )
73       {
74          WordCount application = new WordCount();
75          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
76       }
77
78    }  // end class WordCount
```



c) Write an application that reads several lines of text and prints a table indicating the number of occurrences of each different word in the text. The first version of your program should include the words in the table in the same order in which they appear in the text. For example, the lines

```
To be, or not to be: that is the question:
Whether 'tis nobler in the mind to suffer
```

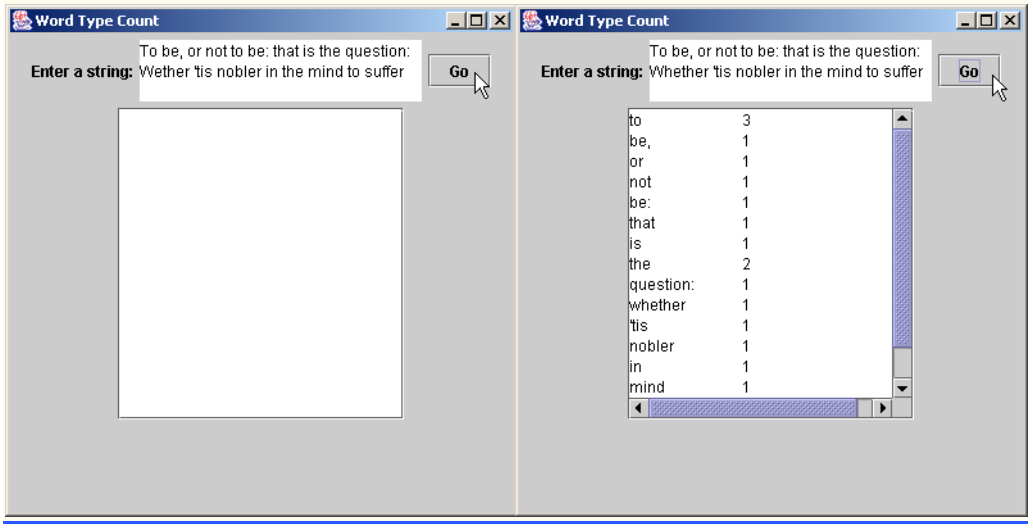
contain the word “to” three times, the word “be” two times, the word “or” once, etc. A more interesting (and useful) printout should then be attempted in which the words are sorted alphabetically.

ANS:

```
1  // Exercise 11.23c Solution: WordTypeCount.java
2  // Program counts the number of occurances of each word in a string
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class WordTypeCount extends JFrame {
9     private JTextArea inputField;
10    private JLabel prompt;
11    private JTextArea display;
12    private JButton goButton;
13
14    // constructor
15    public WordTypeCount()
16    {
17       super( "Word Type Count" );
18       inputField = new JTextArea( 3, 20 );
19
20       goButton = new JButton( "Go" );
21       goButton.addActionListener(
22
23          new ActionListener() {  // anonymous inner class
24
25             public void actionPerformed( ActionEvent event )
26             {
27                String input = inputField.getText();
28                WordCounter counter = new WordCounter();
29                StringTokenizer tokenizer =
30                   new StringTokenizer( input, " \n\r\t" );
31
32                while( tokenizer.hasMoreTokens() )
33                   counter.addWord( tokenizer.nextToken().toLowerCase() );
34
35                display.setText( counter.getWords() );
36             }
37
38          }  // end anonymous inner class
39
40       ); // end call to addActionListener
41
42       prompt = new JLabel( "Enter a string:" );
43       display = new JTextArea( 15, 20 );
44       display.setEditable( false );
45
46       JScrollPane displayScrollPane = new JScrollPane( display );
47
48       // add components to GUI
49       Container container = getContentPane();
50       container.setLayout( new FlowLayout() );
51       container.add( prompt );
52       container.add( inputField );
```

```
53          container.add( goButton );
54          container.add( displayScrollPane );
55
56          setSize( 400, 400 );
57          setVisible( true );
58
59       }  // end constructor
60
61       public static void main( String args[] )
62       {
63          WordTypeCount application = new WordTypeCount();
64          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
65       }
66
67    }  // end class WordTypeCount
68
69    class WordCounter {
70       private int currentWords;
71       private String words[];
72       private int count[];
73
74       // initialize instance variables
75       public WordCounter() {
76          currentWords = 0;
77          words = new String[ 80 ];
78          count = new int[ 80 ];
79       }
80
81       // add a word to our count
82       public void addWord( String input ) {
83
84          for ( int i = 0; i < currentWords; i++ ) {
85
86             // if we have seen the word already
87             if ( input.equals( words[ i ] ) ) {
88                count[i]++;
89                return;
90             }
91          }
92
93          // if we have not seen the word before
94          words[ currentWords ] = input;
95          count[ currentWords ] = 1;
96          currentWords++;
97
98       } // end method addWord
99
100      // create an output string
101      public String getWords() {
102         String output = "";
103
104         for( int i = 0; i < currentWords; i++ )
105            output += words[ i ] + "\t" + count[ i ] + "\n";
106
```

```
107         return output;
108      }
109
110   } // end class WordCounter
```



11.24 *(Printing Dates in Various Formats)* Dates are printed in several common formats. Two of the more common formats are

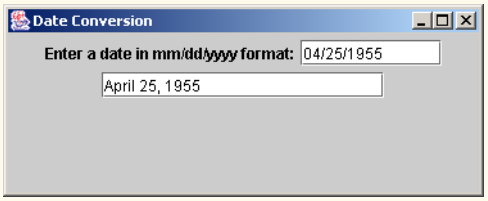
`04/25/1955` and `April 25, 1955`

Write an application that reads a date in the first format and prints that date in the second format.

ANS:

```
1   // Exercise 11.24 Solution: DateConversion.java
2   // Program to convert a date from mm/dd/yyyy format to
3   // month day, year format.
4   import java.awt.*;
5   import java.awt.event.*;
6   import java.util.*;
7   import javax.swing.*;
8
9   public class DateConversion extends JFrame {
10
11     // names of the months
12     private static final String[] months = { "", "January",
13        "February", "March", "April", "May", "June", "July",
14        "August", "September", "October", "November", "December" };
15
16     private JLabel prompt;
17     private JTextField inputField;
18     private JTextField display;
19
20     public DateConversion()
21     {
```

```
22         super( "Date Conversion" );
23
24         prompt = new JLabel( "Enter a date in mm/dd/yyyy format:" );
25         inputField = new JTextField( 10 );
26         inputField.addActionListener(
27
28            new ActionListener () {  // anonymous inner class
29
30               public void actionPerformed( ActionEvent event )
31               {
32                  // split the string into pieces
33                  String input = inputField.getText();
34                  StringTokenizer tokenize =
35                     new StringTokenizer( input, "/" );
36
37                  // get the day, month and year as integers
38                  int month = Integer.parseInt( tokenize.nextToken() );
39                  int day = Integer.parseInt( tokenize.nextToken() );
40                  int year = Integer.parseInt( tokenize.nextToken() );
41
42                  // get the name of the month
43                  String monthName = months[ month ];
44
45                  display.setText( monthName + " " + day + ", " + year );
46               }
47
48            } // end anonymous inner class
49
50         ); // end call to addActionListener
51
52         display = new JTextField( 20 );
53
54         // add components to GUI
55         Container container = getContentPane();
56         container.setLayout( new FlowLayout() );
57         container.add( prompt );
58         container.add( inputField );
59         container.add( display );
60
61         setSize( 375, 150 );
62         setVisible( true );
63
64      } // end constructor
65
66      public static void main( String args[] ) {
67         DateConversion application = new DateConversion();
68         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
69      }
70
71   } // end class DateConversion
```



11.25 *(Check Protection)* Computers are frequently employed in check-writing systems such as payroll and accounts payable applications. Many strange stories circulate regarding weekly paychecks being printed (by mistake) for amounts in excess of $1 million. Incorrect amounts are printed by computerized check-writing systems because of human error or machine failure. Systems designers build controls into their systems to prevent such erroneous checks from being issued.

Another serious problem is the intentional alteration of a check amount by someone who plans to cash a check fraudulently. To prevent a dollar amount from being altered, most computerized check-writing systems employ a technique called *check protection.* Checks designed for imprinting by computer contain a fixed number of spaces in which the computer may print an amount. Suppose a paycheck contains eight blank spaces in which the computer is supposed to print the amount of a weekly paycheck. If the amount is large, then all eight of those spaces will be filled. For example,

```
1,230.60 (check amount)
--------
12345678 (position numbers)
```

On the other hand, if the amount is less than $1000, then several of the spaces would ordinarily be left blank. For example,

```
   99.87
--------
12345678
```

contains three blank spaces. If a check is printed with blank spaces, it is easier for someone to alter the amount of the check. To prevent a check from being altered, many check-writing systems insert *leading asterisks* to protect the amount as follows:

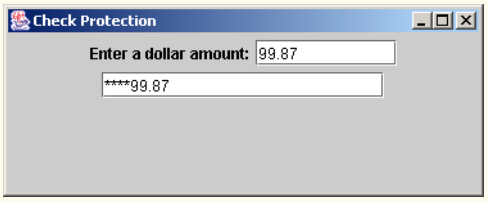
```
***99.87
--------
12345678
```

Write an application that inputs a dollar amount to be printed on a check, then prints the amount in check-protected format with leading asterisks if necessary. Assume that nine spaces are available for printing the amount.

ANS:

```
1  // Exercise 11.25 Solution: CheckProtection.java
2  // Program that protects number values on checks
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
```

```
 8  public class CheckProtection extends JFrame {
 9     private JLabel prompt;
10     private JTextField inputField;
11     private JTextField display;
12
13     public CheckProtection()
14     {
15        super( "Check Protection" );
16
17        prompt = new JLabel( "Enter a dollar amount:" );
18        inputField = new JTextField( 10 );
19        inputField.addActionListener(
20
21           new ActionListener () {  // anonymous inner class
22
23              public void actionPerformed( ActionEvent event )
24              {
25                 String input = inputField.getText();
26                 String temp = "";
27
28                 // add asterisks to fill up the 9 spaces
29                 for ( int i = 9; i > input.length(); i-- )
30                    temp += "*";
31
32                 temp += input;
33
34                 display.setText( temp );
35              }
36
37           } // end anonymous inner class
38
39        ); // end call to addActionListener
40
41        display = new JTextField( 20 );
42
43        // add components to GUI
44        Container container = getContentPane();
45        container.setLayout( new FlowLayout() );
46        container.add( prompt );
47        container.add( inputField );
48        container.add( display );
49
50        setSize( 375, 150 );
51        setVisible( true );
52
53     } // end constructor
54
55     public static void main( String args[] )
56     {
57        CheckProtection application = new CheckProtection();
58        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
59     }
60
61  } // end class CheckProtection
```



11.26 *(Writing the Word Equivalent of a Check Amount)* Continuing the discussion of Exercise 11.25, we reiterate the importance of designing check-writing systems to prevent alteration of check amounts. One common security method requires that the check amount be written in numbers and "spelled out" in words as well. Even if someone is able to alter the numerical amount of the check, it is extremely difficult to change the amount in words.

a) Many computerized check-writing systems do not print the check amount in words. Perhaps the main reason for this omission is that most high-level languages used in commercial applications do not contain adequate string-manipulation features. Another reason is that the logic for writing word equivalents of check amounts is somewhat involved.

b) Write an application that inputs a numeric check amount and writes the word equivalent of the amount. For example, the amount 112.43 should be written as

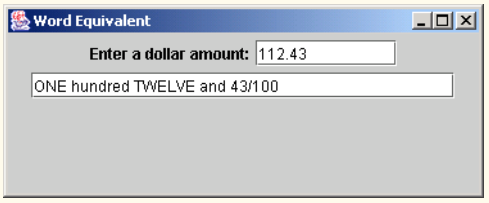
```
ONE HUNDRED TWELVE and 43/100
```

ANS:

```
1  // Exercise 11.26 Solution: WordEquivalent.java
2  // Program that outputs the word equivalent of a dollar amount
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class WordEquivalent extends JFrame {
9     private static final String[] ones = { "", "ONE ", "TWO ", "THREE ",
10       "FOUR ", "FIVE ", "SIX ", "SEVEN ", "EIGHT ", "NINE " };
11    private static final String[] teens = { "TEN ", "ELEVEN ", "TWELVE ",
12       "THIRTEEN ", "FOURTEEN ", "FIFTEEN ", "SIXTEEN ", "SEVENTEEN ",
13       "EIGHTEEN ", "NINETEEN "};
14    private static final String[] tens = { "", "TEN ", "TWENTY ",
15       "THIRTY ", "FORTY ", "FIFTY ", "SIXTY ", "SEVENTY ", "EIGHTY ",
16       "NINETY " };
17    private JLabel prompt;
18    private JTextField inputField;
19    private JTextField display;
20
21    public WordEquivalent()
22    {
23       super( "Word Equivalent" );
24
25       prompt = new JLabel( "Enter a dollar amount:" );
26       inputField = new JTextField( 10 );
27       inputField.addActionListener(
28
29          new ActionListener () {  // anonymous inner class
```

```
30
31             public void actionPerformed( ActionEvent event )
32             {
33                String temp = "";
34                String input = inputField.getText();
35
36                // get the dollar value
37                double amount = Double.parseDouble( input );
38                int dollar = (int) amount;
39
40                // get any change
41                double change = amount - dollar;
42
43                // if there is a hundreds component
44                if ( dollar >= 100 )
45                   temp += getMoney( dollar / 100 ) + "hundred ";
46
47                // output the rest of the dollars
48                temp += getMoney( dollar % 100 ) + "and ";
49
50                // output the change
51                temp += getChange( change );
52
53                display.setText( temp );
54             }
55
56          } // end anonymous inner class
57
58       ); // end addActionListener
59
60       display = new JTextField( 30 );
61
62       // add components to GUI
63       Container container = getContentPane();
64       container.setLayout( new FlowLayout() );
65       container.add( prompt );
66       container.add( inputField );
67       container.add( display );
68
69       setSize( 375, 150 );
70       setVisible( true );
71
72    } // end constructor
73
74    private String getMoney( int dollar ) {
75
76       // if there is less than ten dollars
77       if ( dollar < 10 )
78
79          // access the ones array
80          return ones[ dollar ];
81
82       // if the amount is in the teens
83       else if ( dollar < 20 )
```

```
84
85             // access the teens array
86             return teens[ dollar - 10 ];
87
88          // if the amount is more than 20
89          else
90
91             // create the amount with the tens and the ones arrays
92             return tens[ dollar / 10 ] + ones[ dollar % 10 ];
93
94       } // end method getMoney
95
96       private String getChange( double amount ) {
97
98          // calculate the change as an integer value
99          int change = (int)(amount * 100);
100
101         // output the change out of 100
102         return change + "/100";
103      }
104
105      public static void main( String args[] ) {
106         WordEquivalent application = new WordEquivalent();
107         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
108      }
109
110   } // end class WordEquivalent
```



11.27 *(Morse Code)* Perhaps the most famous of all coding schemes is the Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. The Morse code assigns a series of dots and dashes to each letter of the alphabet, each digit, and a few special characters (such as period, comma, colon and semicolon). In sound-oriented systems, the dot represents a short sound and the dash represents a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems. Separation between words is indicated by a space or, simply, the absence of a dot or dash. In a sound-oriented system, a space is indicated by a short time during which no sound is transmitted. The international version of the Morse code appears in Fig. 11.33.

Write an application that reads an English language phrase and encodes the phrase into Morse code. Also write a program that reads a phrase in Morse code and converts the phrase into the English language equivalent. Use one blank between each Morse-coded letter and three blanks between each Morse-coded word.

Character	Code	Character	Code
A	.-	T	-
B	-...	U	..-
C	-.-.	V	...-
D	-..	W	.--
E	.	X	-..-
F	..-.	Y	-.--
G	--.	Z	--..
H	....		
I	..	Digits	
J	.---	1	.----
K	-.-	2	..---
L	.-..	3	...--
M	--	4	....-
N	-.	5	.....
O	---	6	-....
P		7	--...
Q	--.-	8	---..
R	.-.	9	----.
S	...	0	-----

Fig. 11.33 The letters of the alphabet as expressed in international Morse code .

ANS:

```
1  // Exercise 11.27 Solution: MorseCode.java
2  // Program that outputs the morse code for a string
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class MorseCode extends JFrame {
8
9     // the numbers from 0 to 9
10    private static final String[] numbers = { "-----", ".----", "..---",
11       "...--", "....-", ".....", "-....", "--...", "---..", "----." };
12
13    // the letters from a to z
14    private static final String[] letters = { ".-", "-...", "-.-.", "-..",
15       ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..", "--",
16       "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-",
17       ".--", "-..-", "-.--", "--.." };
18    private JLabel prompt;
```

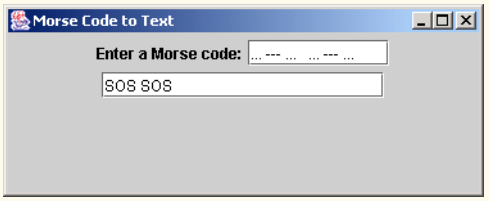
```
19    private JTextField inputField;
20    private JTextField display;
21
22    public MorseCode()
23    {
24       super( "Morse Code" );
25
26       prompt = new JLabel( "Enter a phrase:" );
27       inputField = new JTextField( 10 );
28       inputField.addActionListener(
29
30          new ActionListener () {  // anonymous inner class
31
32             public void actionPerformed( ActionEvent event )
33             {
34                String temp = "";
35                String input = inputField.getText();
36
37                // loop through the string
38                for ( int i = 0; i < input.length(); i++ ) {
39                   char alpha = input.charAt( i );
40
41                   // if the character is a number, access the number array
42                   if ( Character.isDigit( alpha ) )
43                      temp += numbers[ alpha - 48 ] + " ";
44
45                   // if the character is a letter, access the letter array
46                   if ( Character.isLetter( alpha ) )
47                      temp += letters[
48                         Character.toUpperCase( alpha ) - 65 ] + " ";
49
50                   // if the character is a space, output two extra spaces
51                   if ( alpha == ' ' )
52                      temp += "   ";
53                }
54
55                display.setText( temp );
56             }
57
58          } // end anonymous inner class
59
60       ); // end addActionListener
61
62       display = new JTextField( 20 );
63
64       // add components to GUI
65       Container container = getContentPane();
66       container.setLayout( new FlowLayout() );
67       container.add( prompt );
68       container.add( inputField );
69       container.add( display );
70
71       setSize( 375, 150 );
72       setVisible( true );
```

```
73
74       } // end constructor
75
76       public static void main( String args[] ) {
77          MorseCode application = new MorseCode();
78          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
79       }
80
81    } // end class MorseCode
```


```
1  // Exercise 11.27 Solution: MorseToNormalText.java
2  // Program that outputs the normal text for a morse code.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class MorseToNormalText extends JFrame{
9
10    // morse code numbers and letters
11    String[] morseCharacters = { "-----", ".----", "..---", "...--",
12       "....-", ".....", "-....", "--...", "---..", "----.", ".-",
13       "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---",
14       "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...",
15       "-", "..-", "...-", ".--", "-..-", "-.--", "--.." };
16
17    // normal English characters
18    String[] normalCharacters = { "0", "1", "2", "3", "4", "5", "6", "7",
19       "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K",
20       "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X",
21       "Y", "Z" };
22    private JLabel prompt;
23    private JTextField inputField;
24    private JTextField display;
25
26    public MorseToNormalText()
27    {
28       super( "Morse Code to Text" );
29
30       prompt = new JLabel( "Enter a Morse code:" );
31       inputField = new JTextField( 10 );
32       inputField.addActionListener(
33
34          new ActionListener () {  // anonymous inner class
35
```

```
36             public void actionPerformed( ActionEvent event )
37             {
38                String input = inputField.getText();
39                String output = translate( input );
40
41                display.setText( output );
42             }
43          }
44       );
45
46       display = new JTextField( 20 );
47
48       // add components to GUI
49       Container container = getContentPane();
50       container.setLayout( new FlowLayout() );
51       container.add( prompt );
52       container.add( inputField );
53       container.add( display );
54
55       setSize( 375, 150 );
56       setVisible( true );
57
58    } // end constructor
59
60    // translate morse code phrase to normal text
61    private String translate( String morseCode )
62    {
63       String result = "";
64       int start = 0, length = 0;
65       int threeSpaces = morseCode.indexOf( "   " );
66       String word;
67
68       // while not reach the end of morse code
69       while ( length < morseCode.length() ) {
70
71          if ( threeSpaces != -1 ) {
72             word = morseCode.substring( start, threeSpaces );
73             length = threeSpaces;
74          }
75          else {
76             word = morseCode.substring( start, morseCode.length() );
77             length = morseCode.length();
78          }
79
80          StringTokenizer letters = new StringTokenizer( word );
81
82          // decode letter
83          while ( letters.hasMoreTokens() )
84             result += decode( letters.nextToken() );
85
86          result += " ";
87          start = threeSpaces + 3;
88          threeSpaces = morseCode.indexOf( "   ", start );
89
90       } // end while
```

```
91
92          return result;
93
94       } // end method translate
95
96       // decode morse code letter
97       private String decode( String morseCode )
98       {
99          for ( int i = 0; i < morseCharacters.length; i++ )
100
101            if ( morseCode.equals( morseCharacters[ i ] ) )
102               return normalCharacters[ i ];
103
104         return "";
105
106      } // end method decode
107
108      public static void main(String[] args)
109      {
110         MorseToNormalText application = new MorseToNormalText();
111         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
112      }
113
114   } // end class MorseToNormalText
```



11.28 *(Metric Conversion Program)* Write an application that will assist the user with metric conversions. Your program should allow the user to specify the names of the units as strings (i.e., centimeters, liters, grams, etc., for the metric system and inches, quarts, pounds, etc., for the English system) and should respond to simple questions such as

```
"How many inches are in 2 meters?"
"How many liters are in 10 quarts?"
```

Your program should recognize invalid conversions. For example, the question

```
"How many feet are in 5 kilograms?"
```

is not meaningful because `"feet"` is a unit of length while `"kilograms"` is a unit of mass.

ANS:

```
1  // Exercise 11.28 Solution: Convert.java
2  // Program converts from selected units to other selected units.
3  // NOTE: unit names must always be entered in the plural form.
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
```

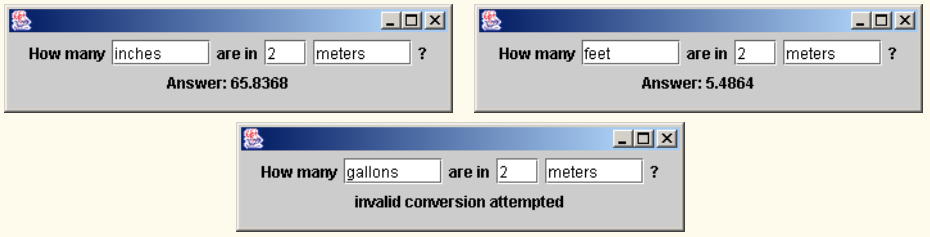
```
 7
 8  public class Convert extends JFrame {
 9     private JLabel prompt, prompt2, prompt3, status;
10     private JTextField fromUnits, number, toUnits;
11
12     // set up GUI
13     public Convert()
14     {
15        prompt = new JLabel( "How many" );
16        prompt2 = new JLabel( "are in" );
17        prompt3 = new JLabel( "?" );
18        status = new JLabel();
19
20        toUnits = new JTextField( 7 );
21        number = new JTextField( 3 );
22        fromUnits = new JTextField( 7 );
23        fromUnits.addActionListener(
24
25           new ActionListener() {  // anonymous inner class
26
27              // perform conversion
28              public void actionPerformed( ActionEvent event )
29              {
30                 status.setText( "" );
31
32                 String lengthUnits[] =
33                    { "inches", "meters", "feet", "yards" };
34                 String massUnits[] =
35                    { "grams", "carats", "ounces", "slugs" };
36                 String volumeUnits[] =
37                    { "liters", "gallons", "pints", "pecks" };
38
39                 // unit to be converted to
40                 String to = toUnits.getText().toLowerCase();
41
42                 // starting unit
43                 String from = fromUnits.getText().toLowerCase();
44
45                 // starting amount
46                 int num = Integer.parseInt( number.getText() );
47
48                 // length conversions
49                 if ( isMatch( to, from, lengthUnits ) == true ) {
50                    int inchesPerFoot = 12, feetPerYard = 3;
51                    double yardPerMeter = 0.9144, value = num;
52
53                    if ( to.equals( "inches" ) && from.equals( "meters" ) )
54                       value = num * feetPerYard *
55                          inchesPerFoot * yardPerMeter;
56
57                    else if(
58                       to.equals( "meters" ) && from.equals( "inches" ) )
59                       value =
60                          num / yardPerMeter * feetPerYard / inchesPerFoot;
```

```
61
62                   else if(
63                      to.equals( "feet" ) && from.equals( "meters" ) )
64                      value = num * yardPerMeter * feetPerYard;
65
66                   else if(
67                      to.equals( "meters" ) && from.equals( "feet" ) )
68                      value = num * yardPerMeter / feetPerYard;
69
70                   else if( to.equals( "feet" ) && from.equals( "yards" ) )
71                      value = num * feetPerYard;
72
73                   else if( to.equals( "yards" ) && from.equals( "feet" ) )
74
75                      value = num / feetPerYard;
76
77                   else if(
78                      to.equals( "feet" ) && from.equals( "inches" ) )
79                      value = num / inchesPerFoot;
80
81                   else if(
82                      to.equals( "inches" ) && from.equals( "feet" ) )
83                      value = num * inchesPerFoot;
84
85                   else if(
86                      to.equals( "yards" ) && from.equals( "inches" ) )
87                      value = num / inchesPerFoot * feetPerYard;
88
89                   else if(
90                      to.equals( "inches" ) && from.equals( "yards" ) )
91                      value = num * inchesPerFoot * feetPerYard;
92
93                   else if(
94                      to.equals( "meters" ) && from.equals( "yards" ) )
95                      value = num / yardPerMeter;
96
97                   else if(
98                      to.equals( "yards" ) && from.equals( "meters" ) )
99                      value = num * yardPerMeter;
100
101                  status.setText( "Answer: " + value );
102
103               } // end if
104
105               // mass conversions
106               else if ( isMatch( to, from, massUnits ) == true ) {
107                  double kiloPerSlug = 14.5939, kiloPerCarat = 0.0002,
108                     kiloPerOunce = 0.0283495, value = num;
109                  int gramsPerKilo = 1000;
110
111                  if ( to.equals( "grams" ) && from.equals( "carats" ) )
112                     value = num / ( gramsPerKilo * kiloPerCarat );
113
```

```
114                   else if (
115                      to.equals( "carats" ) && from.equals( "grams" ) )
116                      value = num * gramsPerKilo * kiloPerCarat;
117
118                   else if (
119                      to.equals( "grams" ) && from.equals( "ounces" ) )
120                      value = num * kiloPerOunce * gramsPerKilo;
121
122                   else if (
123                      to.equals( "ounces" ) && from.equals( "grams" ) )
124                      value = num / ( kiloPerOunce * gramsPerKilo );
125
126                   else if (
127                      to.equals( "grams" ) && from.equals( "slugs" ) )
128                      value = num * kiloPerSlug * gramsPerKilo;
129
130                   else if (
131                      to.equals( "slugs" ) && from.equals( "grams" ) )
132                      value = num / ( kiloPerSlug * gramsPerKilo );
133
134                   else if (
135                      to.equals( "carats" ) && from.equals( "ounces" ) )
136                      value = num * kiloPerOunce / kiloPerCarat;
137
138                   else if (
139                      to.equals( "ounces" ) && from.equals( "carats" ) )
140                      value = num / kiloPerOunce * kiloPerCarat;
141
142                   else if (
143                      to.equals( "carats" ) && from.equals( "slugs" ) )
144                      value = num * kiloPerSlug / kiloPerCarat;
145
146                   else if (
147                      to.equals( "slugs" ) && from.equals( "carats" ) )
148                      value = num / kiloPerSlug * kiloPerCarat;
149
150                   else if (
151                      to.equals( "ounces" ) && from.equals( "slugs" ) )
152                      value = num * kiloPerOunce / kiloPerSlug;
153
154                   else if (
155                      to.equals( "slugs" ) && from.equals( "ounces" ) )
156                      value = num / kiloPerOunce * kiloPerSlug ;
157
158                   status.setText( "Answer: " + value );
159
160                } // end else if
161
162                // volume conversions
163                else if ( isMatch( to, from, volumeUnits ) == true ) {
164                   double meters3PerPeck = 0.0088097675,
165                      meters3PerPint = 0.0004731764,
166                      meters3PerGallon = 0.0037854117,
167                      meters3PerLiter = 0.001, value = num;
```

```
168
169                if ( to.equals( "liters" ) && from.equals( "gallons" ) )
170                   value = num * meters3PerGallon / meters3PerLiter;
171
172                else if ( to.equals( "gallons" ) &&
173                   from.equals( "liters" ) )
174                   value = num / meters3PerGallon * meters3PerLiter;
175
176                else if ( to.equals( "liters" ) &&
177                   from.equals( "pints" ) )
178                   value = num * meters3PerPint / meters3PerLiter;
179
180                else if ( to.equals( "pints" ) &&
181                   from.equals( "liters" ) )
182                   value = num / meters3PerPint * meters3PerLiter;
183
184                else if ( to.equals( "liters" ) &&
185                   from.equals( "pecks" ) )
186                   value = num * meters3PerPeck / meters3PerLiter;
187
188                else if ( to.equals( "pecks" ) &&
189                   from.equals( "liters" ) )
190                   value = num / meters3PerPeck * meters3PerLiter;
191
192                else if ( to.equals( "gallons" ) &&
193                   from.equals( "pints" ) )
194                   value = num * meters3PerPint / meters3PerGallon;
195
196                else if ( to.equals( "pints" ) &&
197                   from.equals( "gallons" ) )
198                   value = num / meters3PerPint * meters3PerGallon;
199
200                else if ( to.equals( "gallons" ) &&
201                   from.equals( "pecks" ) )
202                   value = num * meters3PerPeck / meters3PerGallon;
203
204                else if ( to.equals( "pecks" ) &&
205                   from.equals( "gallons" ) )
206                   value = num / meters3PerPeck * meters3PerGallon;
207
208                else if ( to.equals( "pints" ) &&
209                   from.equals( "pecks" ) )
210                   value = num / meters3PerPint * meters3PerPeck;
211
212                else if ( to.equals( "pecks" ) &&
213                   from.equals( "pints" ) )
214                   value = num * meters3PerPint / meters3PerPeck;
215
216                status.setText( "Answer: " + value );
217
218             } // end else if
219
220             // incompatible units
221             else
222                status.setText( "invalid conversion attempted" );
```

```
223
224            } // end method actionPerformed
225
226         }  // end anonymous inner class
227
228      ); // end call to addActionListener
229
230      Container container = getContentPane();
231      container.setLayout( new FlowLayout() );
232      container.add( prompt );
233      container.add( toUnits );
234      container.add( prompt2 );
235      container.add( number );
236      container.add( fromUnits );
237      container.add( prompt3 );
238      container.add( status);
239
240      setSize(350,85);
241      setVisible( true );
242
243   }  // end constructor
244
245   // check if both units are of same type (i.e. length)
246   private boolean isMatch( String firstUnit,
247      String secondUnit, String unit[] )
248   {
249      boolean flag = false, flag2 = false;
250
251      for ( int x = 0; x < unit.length; x++ ) {
252
253         if ( firstUnit.equals( unit[ x ] ) )
254            flag = true;
255
256         if ( secondUnit.equals( unit[ x ] ) )
257            flag2 = true;
258
259         // same units
260         if ( flag == true && flag2 == true )
261            return true;
262      }
263
264      return false;
265   }
266
267   public static void main( String args[] )
268   {
269      Convert application = new Convert();
270      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
271   }
272
273 }  // end class Convert
```



SPECIAL SECTION: CHALLENGING STRING MANIPULATION PROJECTS

11.29 *(Project: A Spelling Checker)* Many popular word processing software packages have built-in spell checkers. In this project, you are asked to develop your own spell-checker utility. We make suggestions to help get you started. You should then consider adding more capabilities. Use a computerized dictionary (if you have access to one) as a source of words.

Why do we type so many words with incorrect spellings? In some cases, it is because we simply do not know the correct spelling, so we make a best guess. In some cases, it is because we transpose two letters (e.g., "defualt" instead of "default"). Sometimes we double-type a letter accidentally (e.g., "hanndy" instead of "handy"). Sometimes we type a nearby key instead of the one we intended (e.g., "biryhday" instead of "birthday"), and so on.

Design and implement a spell-checker application in Java. Your program should maintain an array `wordList` of strings. Enable the user to enter these strings. [*Note*: In Chapter 17, we will introduce file processing. Once you have this capability, you can obtain the words for the spell checker from a computerized dictionary stored in a file.]

Your program should ask a user to enter a word. The program should then look up that word in the `wordList` array. If the word is in the array, your program should print "`Word is spelled correctly`." If the word is not in the array, your program should print "`word is not spelled correctly`." Then your program should try to locate other words in `wordList` that might be the word the user intended to type. For example, you can try all possible single transpositions of adjacent letters to discover that the word "default" is a direct match to a word in `wordList`. Of course, this implies that your program will check all other single transpositions, such as "edfault," "dfeault," "deafult," "defalut" and "defautl." When you find a new word that matches one in `wordList`, print that word in a message, such as "`Did you mean "default?"`."

Implement other tests, such as replacing each double letter with a single letter and any other tests you can develop to improve the value of your spell checker.

11.30 *(Project: A Crossword Puzzle Generator)* Most people have worked a crossword puzzle, but few have ever attempted to generate one. Generating a crossword puzzle is suggested here as a string-manipulation project requiring substantial sophistication and effort.

There are many issues the programmer must resolve to get even the simplest crossword puzzle-generator program working. For example, how do you represent the grid of a crossword puzzle inside the computer? Should you use a series of strings or two-dimensional arrays?

The programmer needs a source of words (i.e., a computerized dictionary) that can be directly referenced by the program. In what form should these words be stored to facilitate the complex manipulations required by the program?

If you are really ambitious, you will want to generate the "clues" portion of the puzzle, in which the brief hints for each "across" word and each "down" word are printed. Merely printing a version of the blank puzzle itself is not a simple problem.

12

Graphics and Java2D

Objectives

- To understand graphics contexts and graphics objects.
- To understand and be able to manipulate colors.
- To understand and be able to manipulate fonts.
- To use `Graphics` methods to draw lines, rectangles, rectangles with rounded corners, three-dimensional rectangles, ovals, arcs and polygons.
- To use methods of class `Graphics2D` from the Java2D API to draw lines, rectangles, rectangles with rounded corners, ellipses, arcs and general paths.
- To be able to specify `Paint` and `Stroke` characteristics of shapes displayed with `Graphics2D`.

One picture is worth ten thousand words.
Chinese proverb

Treat nature in terms of the cylinder, the sphere, the cone, all in perspective.
Paul Cezanne

Nothing ever becomes real till it is experienced—even a proverb is no proverb to you till your life has illustrated it.
John Keats

A picture shows me at a glance what it takes dozens of pages of a book to expound.
Ivan Sergeyevich Turgenev

SELF-REVIEW EXERCISES

12.1 Fill in the blanks in each of the following statements:

a) In Java2D, method _____ of class _____ sets the characteristics of a line used to draw a shape.

ANS: `setStroke`, `Graphics2D`

b) Class _____ helps specify the fill for a shape such that the fill gradually changes from one color to another.

ANS: `GradientPaint`

c) The _____ method of class `Graphics` draws a line between two points.

ANS: `drawLine`

d) RGB is short for _____, _____ and _____.

ANS: red, green, blue

e) Font sizes are measured in units called _____.

ANS: points

f) Class _____ helps specify the fill for a shape using a pattern drawn in a `BufferedImage`.

ANS: `TexturePaint`

12.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The first two arguments of `Graphics` method drawOval specify the center coordinate of the oval.

ANS: False. The first two arguments specify the upper-left corner of the bounding rectangle.

b) In the Java coordinate system, *x* values increase from left to right.

ANS: True.

c) Method `fillPolygon` draws a solid polygon in the current color.

ANS: True.

d) Method `drawArc` allows negative angles.

ANS: True.

e) Method `getSize` returns the size of the current font in centimeters.

ANS: False. Font sizes are measured in points.

f) Pixel coordinate *(0, 0)* is located at the exact center of the monitor.

ANS: False. The coordinate *(0,0)* corresponds to the upper-left corner of a GUI component on which drawing occurs.

12.3 Find the error(s) in each of the following and explain how to correct the error(s). Assume that `g` is a `Graphics` object.

a)
```
g.setFont( "SansSerif" );
```
ANS: The `setFont` method takes a `Font` object as an argument—not a `String`.

b)
```
g.erase( x, y, w, h );    // clear rectangle at (x, y)
```
ANS: The `Graphics` class does not have an `erase` method. The `clearRect` method should be used.

c)
```
Font f = new Font( "Serif", Font.BOLDITALIC, 12 );
```
ANS: `Font.BOLDITALIC` is not a valid font style. To get a bold italic font, use `Font.BOLD + Font.ITALIC`.

d)
```
g.setColor( Color.Yellow );  // change color to yellow
```
ANS: `Yellow` should be all uppercase letters as in: `g.setColor( Color.YELLOW );`.

EXERCISES

12.4 Fill in the blanks in each of the following statements:

a) Class _____ of the Java2D API is used to draw ovals.

ANS: `Ellipse2D`.

b) Methods `draw` and `fill` of class `Graphics2D` require an object of type _____ as their argument.
ANS: `Shape`.
c) The three constants that specify font style are _____, _____ and _____.
ANS: `Font.PLAIN`, `Font.BOLD` and `Font.ITALIC`.
d) `Graphics2D` method _____ sets the painting color for Java2D shapes.
ANS: `setColor`.

12.5 State whether each of the following is *true* or *false*. If *false*, explain why.
a) The `drawPolygon` method automatically connects the endpoints of the polygon.
ANS: True.
b) The `drawLine` method draws a line between two points.
ANS: True.
c) The `fillArc` method uses degrees to specify the angle.
ANS: True.
d) In the Java coordinate system, *y* values increase from top to bottom.
ANS: False. In the Java coordinate system, *y* values increase from top to bottom.
e) The `Graphics` class inherits directly from class `Object`.
ANS: True.
f) The `Graphics` class is an `abstract` class.
ANS: True.
g) The `Font` class inherits directly from class `Graphics`.
ANS: False. Class `Font` inherits directly from class `Object`.

12.6 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawOval` method of class `Graphics`.
ANS:

```
1  // Exercise 12.6 Solution: Concentric.java
2  // Program draws concentric circles.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Concentric extends JFrame {
7
8     private int screenOffset = 150;
9
10    // constructor sets window's title bar string and dimensions
11    public Concentric()
12    {
13       super( "Concentric" );
14
15       setSize( 500, 500 );
16       setVisible( true );
17    }
18
19    // draw concentric ovals
20    public void paint( Graphics g )
21    {
22       super.paint( g );
23
24       for ( int i = 1; i <= 8; i++ ) {
25
```

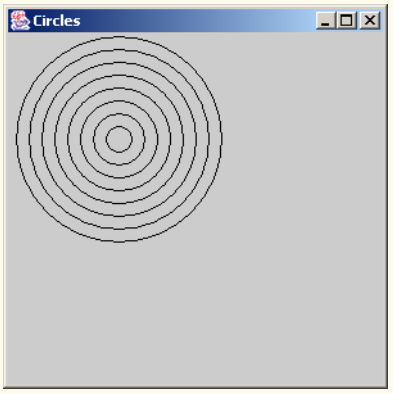
```
26            int origin = screenOffset + 80 - i * 10;
27            g.drawOval( origin, origin, i * 20, i * 20);
28         }
29      }
30
31      public static void main( String args[] )
32      {
33         Concentric application = new Concentric();
34         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
35      }
36
37   }  // end class Concentric
```

12.7 Write a program that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use the `drawArc` method.

ANS:

```
1  // Exercise 12.7 Solution: Circles.java
2  // This program draws concentric circles
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Circles extends JFrame {
8
9     // constructor
10    public Circles()
11    {
```

```
12          super( "Circles" );
13          setSize( 300, 300 );
14          setVisible( true );
15       }
16
17       // draw eight circles separated by 10 pixels
18       public void paint( Graphics g )
19       {
20          super.paint( g );
21
22          // create 8 concentric circles
23          for ( int topLeft = 0; topLeft < 80; topLeft += 10 ) {
24             int radius = 160 - ( topLeft * 2 );
25             g.drawArc( topLeft + 10, topLeft + 25, radius, radius, 0, 360 );
26          }
27       }
28
29       public static void main( String args[] )
30       {
31          Circles app = new Circles();
32          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
33       }
34
35    }  // end class Circles
```

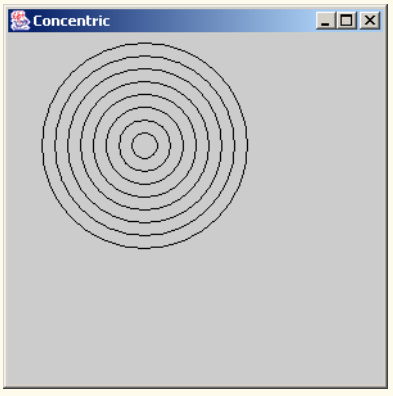


12.8 Modify your solution to Exercise 12.6 to draw the ovals by using instances of class `Ellipse2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.8 Solution: Concentric.java
2  // This program draws concentric circles using Graphics2D
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import java.awt.image.*;
7  import javax.swing.*;
8
```

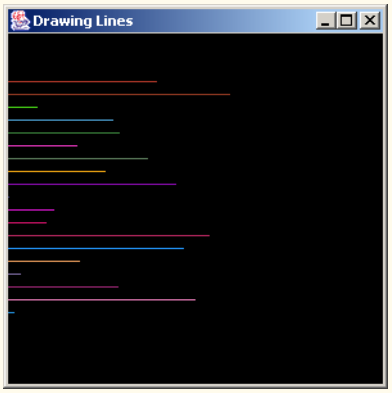
```
 9  public class Concentric extends JFrame {
10
11     // constructor
12     public Concentric()
13     {
14        super( "Concentric" );
15        setSize( 300, 300 );
16        setVisible( true );
17     }
18
19     // draw eight concentric circles separated by 10 pixels
20     public void paint( Graphics g )
21     {
22        super.paint( g );
23
24         // create 2D by casting g to Graphics 2D
25         Graphics2D g2d = ( Graphics2D ) g;
26
27         for ( int x = 0; x < 80; x += 10 ) {
28             int y = 160 - ( x * 2 );
29             g2d.draw( new Ellipse2D.Double( x + 30, x + 30, y, y ) );
30         }
31     }
32
33     public static void main( String args[] )
34     {
35        Concentric app = new Concentric();
36        app.setDefaultCloseOperation( EXIT_ON_CLOSE );
37     }
38
39  }  // end class Concentric
```



12.9 Write a program that draws lines of random lengths in random colors.
ANS:

```
1  // Exercise 12.9 Solution: Lines1.java
2  // This program draws lines of random sizes and colors
```

```
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Lines1 extends JFrame {
8
9     // constructor
10    public Lines1()
11    {
12       super( "Drawing Lines" );
13
14       Container container = getContentPane();
15
16       // set background to make lines more visible
17       container.setBackground( Color.black );
18
19       setSize( 300, 300 );
20       setVisible( true );
21    }
22
23    // draw 20 randomly-colored and -sized horizontal lines
24    public void paint( Graphics g )
25    {
26       super.paint( g );
27
28       for ( int y = 60; y < 250; y += 10 ) {
29
30          // create a color with three random floats
31          g.setColor( new Color( ( float ) Math.random(),
32             ( float ) Math.random(), ( float ) Math.random() ) );
33
34          // create a random length
35          int x1 = ( int ) ( 1 + Math.random() * 199 );
36          g.drawLine( 1, y, x1, y );
37       }
38    }
39
40    public static void main( String args[] )
41    {
42       Lines1 app = new Lines1();
43       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
44    }
45
46 }  // end class Lines1
```

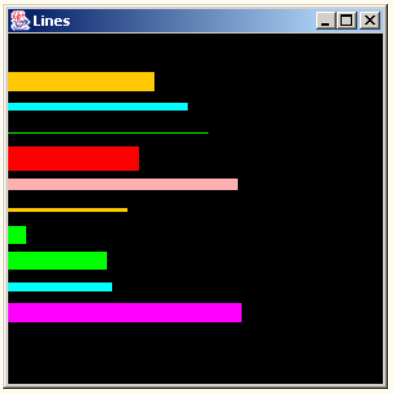


12.10 Modify your solution to Exercise 12.9 to draw random lines, in random colors and random line thicknesses. Use class `Line2D.Double` and method `draw` of class `Graphics2D` to draw the lines.

ANS:

```
1  // Exercise 12.10 Solution: Lines2.java
2  // This program draws lines of different colors
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Lines2 extends JFrame {
9     private Color colors[] = { Color.green, Color.cyan, Color.yellow,
10            Color.darkGray, Color.red, Color.orange,
11            Color.gray, Color.pink, Color.magenta };
12
13    // constructor
14    public Lines2()
15    {
16       super( "Lines" );
17
18       Container container = getContentPane();
19
20       // set background to make lines more visible
21       container.setBackground( Color.black );
22
23       setSize( 300, 300 );
24       setVisible( true );
25    }
26
27    // create 10 lines
28    public void paint( Graphics g )
29    {
30       super.paint( g );
31
32       // create 2D by casting g to Graphics 2D
33       Graphics2D g2d = ( Graphics2D ) g;
```

```
34
35          for ( int y = 60; y < 250; y += 20 ) {
36
37             // choose a random color from array
38             int color = ( int ) ( Math.random() * 9 );
39             g2d.setColor( colors[ color ] );
40
41             // choose a random thickness from 1-20
42             int thickness = ( int ) ( Math.random() * 20 + 1 );
43             g2d.setStroke( new BasicStroke( thickness ) );
44
45             // choose a random length and draw line
46             int x1 = ( int ) ( 1 + Math.random() * 199 );
47             g2d.draw( new Line2D.Double( 1, y, x1, y ) );
48          }
49       }
50
51       public static void main( String args[] )
52       {
53          Lines2 app = new Lines2();
54          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
55       }
56
57    }  // end class Lines2
```



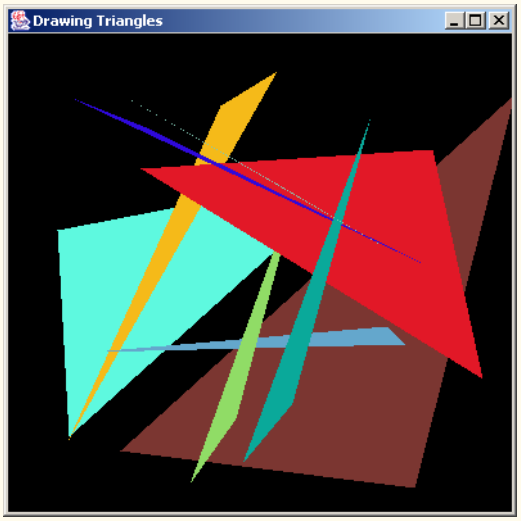
12.11 Write a program that displays randomly generated triangles in different colors. Each triangle should be filled with a different color. Use class `GeneralPath` and method `fill` of class `Graphics2D` to draw the triangles.

ANS:

```
1  // Exercise 12.11 Solution: Triangles.java
2  // Displays randomly generated triangles in different colors.
3  import java.awt.event.*;
4  import java.awt.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
```

```
 8  public class Triangles extends JFrame {
 9
10     // constructor
11     public Triangles()
12     {
13        super( "Drawing Triangles" );
14
15        Container container = getContentPane();
16
17        // set background to make lines more visible
18        container.setBackground( Color.black );
19
20        setSize( 400, 400 );
21        setVisible( true );
22     }
23
24     // draw ten triangles
25     public void paint( Graphics g )
26     {
27        super.paint( g );
28
29        Graphics2D g2d = ( Graphics2D ) g; // cast graphics object
30
31        // create a triangle from three random points
32        for ( int i = 0; i < 10; i++ ) {
33
34           // create the object which will be the triangle
35           GeneralPath triangle = new GeneralPath();
36
37           // use method moveTo to start the triangle
38           int x = ( int ) ( Math.random() * 375 + 25 );
39           int y = ( int ) ( Math.random() * 375 + 25 );
40           triangle.moveTo( x, y );
41
42           // draw a line to the second point
43           x = ( int ) ( Math.random() * 375 + 25 );
44           y = ( int ) ( Math.random() * 375 + 25 );
45           triangle.lineTo( x, y );
46
47           // draw a line to the third point
48           x = ( int ) ( Math.random() * 375 + 25 );
49           y = ( int ) ( Math.random() * 375 + 25 );
50           triangle.lineTo( x, y );
51
52           triangle.closePath(); // draw a line back to the initial point
53
54           // choose a random color
55           g2d.setColor(new Color( ( int ) ( Math.random() * 256 ),
56              ( int ) ( Math.random() * 256 ),
57              ( int ) ( Math.random() * 256 ) ) );
58
59           g2d.fill( triangle ); // color the interior of the triangle
60        }
61     }
```

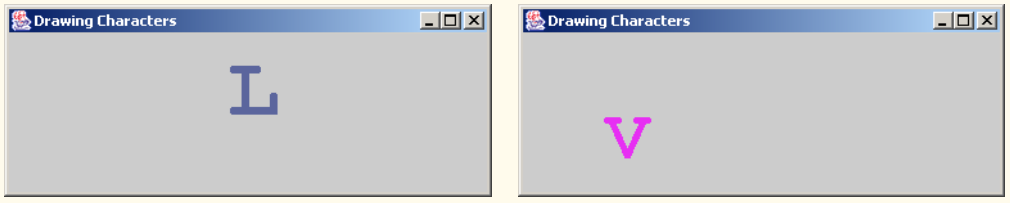
```
62
63     public static void main( String args[] )
64     {
65        Triangles app = new Triangles();
66        app.setDefaultCloseOperation( EXIT_ON_CLOSE );
67     }
68
69  }  // end class Triangles
```



12.12 Write a program that randomly draws characters in different font sizes and colors.
ANS:

```
1  // Exercise 12.12 Solution: Draw.java
2  // Program randomly draws characters
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Draw extends JFrame {
8     private final int DELAY = 999999;
9
10    // constructor sets window's title bar string and dimensions
11    public Draw()
12    {
13       super( "Drawing Characters" );
14
15       setSize( 380, 150 );
16       setVisible( true );
17    }
18
19    // draw characters
20    public void paint( Graphics g )
21    {
```

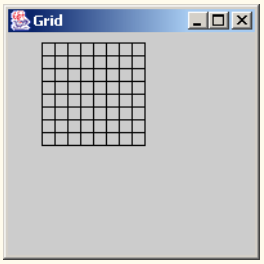
```
22          super.paint( g );
23
24          int fontSize = ( int ) ( 10 + Math.random() * 63 );
25          int x = ( int ) ( Math.random() * 380 );
26          int y = ( int ) ( 50 + Math.random() * 95 );
27          char letters[] = { 'V', 'O', 'L', 'S', '8', '7' };
28          Font font = new Font( "Monospaced", Font.BOLD, fontSize );
29
30          g.setColor( new Color( ( float ) Math.random(),
31             ( float ) Math.random(), ( float ) Math.random() ) );
32          g.setFont( font );
33          g.drawChars( letters, ( int ) ( Math.random() * 6 ), 1, x, y );
34
35          // adding delay is optional the body of the for loop is empty
36          for ( int h = 1; h < DELAY; h++ ) ;
37
38          repaint();
39       }
40
41       public static void main( String args[] )
42       {
43          Draw application = new Draw();
44          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
45       }
46
47    }  // end class Draw
```



12.13 Write a program that draws an 8-by-8 grid. Use the `drawLine` method.
ANS:

```
1  // Exercise 12.13 Solution: Grid1.java
2  // This program draws an 8 x 8 grid
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Grid1 extends JFrame {
8
9     public Grid1()
10    {
11       super( "Grid" );
12
13       setSize( 200, 200 );
14       setVisible( true );
15    }
```

```
16
17       // draw a grid using the drawLine method
18       public void paint( Graphics g )
19       {
20          super.paint( g );
21
22          int y = 30, x1 = 30;
23
24          for ( int row = 0; row <= 8; row++, y += 10 )
25             g.drawLine( 30, y, 110, y );
26
27          for ( int column = 0; column <= 8; column++, x1 += 10 )
28             g.drawLine( x1, 30, x1, 110 );
29       }
30
31       public static void main( String args[] )
32       {
33          Grid1 app = new Grid1();
34          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
35       }
36
37    }  // end class Grid1
```

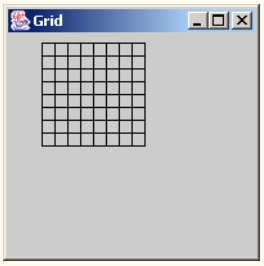


12.14 Modify your solution to Exercise 12.13 to draw the grid using instances of class `Line2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.14 Solution: Grid2.java
2  // This program draws an 8 x 8 grid
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Grid2 extends JFrame {
9
10    // constructor
11    public Grid2()
12    {
13       super( "Grid" );
14
15       setSize( 200, 200 );
```

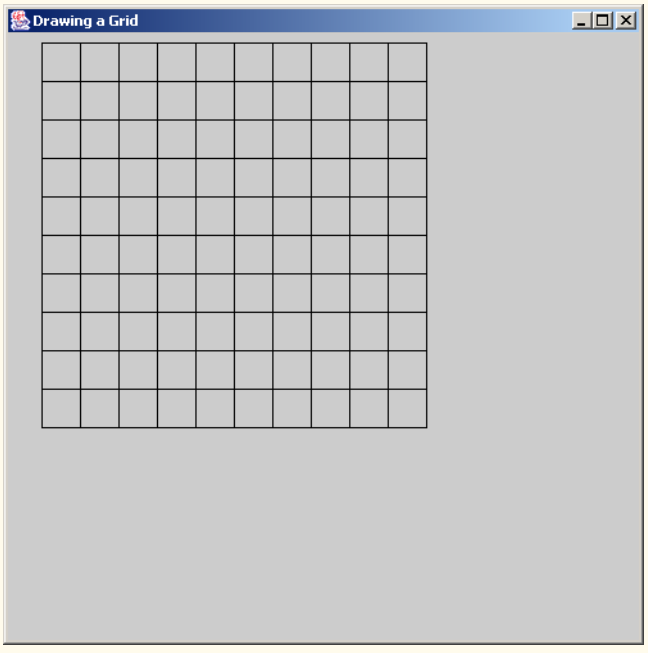
```
16          setVisible( true );
17       }
18
19       // draw an 8x8 grid
20       public void paint( Graphics g )
21       {
22          super.paint( g );
23
24          int y = 30, x1 = 30;
25          Graphics2D g2d = ( Graphics2D ) g;
26
27          for ( int row = 0; row <= 8; row++, y += 10 )
28             g2d.draw( new Line2D.Double( 30, y, 110, y ) );
29
30          for ( int column = 0; column <= 8; column++, x1 += 10 )
31             g2d.draw( new Line2D.Double(  x1, 30, x1, 110 ) );
32       }
33
34       public static void main( String args[] )
35       {
36          Grid2 app = new Grid2();
37          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
38       }
39
40    }  // end class Grid2
```



12.15 Write a program that draws a 10-by-10 grid. Use the `drawRect` method.
ANS:

```
1  // Exercise 12.15 Solution: Grid.java
2  // Program draws a 10x10 grid using drawRect().
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Grid extends JFrame {
7
8     // constructor sets window's title bar string and dimensions
9     public Grid()
10    {
11       super( "Drawing a Grid" );
12
13       setSize( 500, 500 );
```

```
14          setVisible( true );
15       }
16
17       // draw grid
18       public void paint( Graphics g )
19       {
20          super.paint( g );
21
22          for ( int x = 30; x <= 300; x += 30 )
23             for ( int y = 30; y <= 300; y += 30 )
24                g.drawRect( x, y, 30, 30 );
25       }
26
27       public static void main( String args[] )
28       {
29          Grid application = new Grid();
30          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
31       }
32
33    }  // end class Grid
```

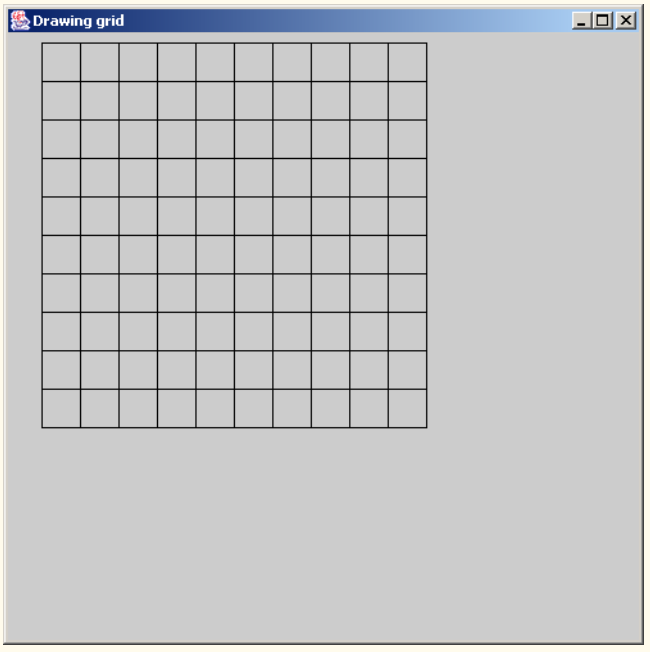


12.16 Modify your solution to Exercise 12.15 to draw the grid by using instances of class `Rectangle2D.Double` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.16 Solution: Grid2.java
2  // Program draws 10x10 grid using draw().
```

```
3  import java.awt.*;
4  import java.awt.geom.*;
5  import javax.swing.*;
6
7  public class Grid2 extends JFrame {
8
9     // constructor sets window's title bar string and dimensions
10    public Grid2()
11    {
12       super( "Drawing grid" );
13
14       setSize( 500, 500 );
15       setVisible( true );
16    }
17
18    // draw grid
19    public void paint( Graphics g )
20    {
21       super.paint( g );
22
23       Graphics2D g2d = ( Graphics2D ) g;
24
25       for ( int x = 30; x <= 300; x += 30 )
26          for ( int y = 30; y <= 300; y += 30 )
27             g2d.draw( new Rectangle2D.Double( x, y, 30, 30 ) );
28    }
29
30    public static void main( String args[] )
31    {
32       Grid2 application = new Grid2();
33       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
34    }
35
36 }  // end class Grid2
```

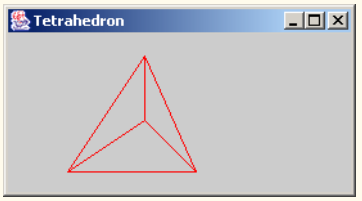


12.17 Write a program that draws a tetrahedron (a three-dimensional shape with four triangular faces). Use class `GeneralPath` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.17 Solution: Tetrahedron.java
2  // Program draws a tetrahedron.
3  import java.awt.*;
4  import java.awt.geom.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Tetrahedron extends JFrame {
9
10    // constructor
11    public Tetrahedron()
12    {
13       super( "Tetrahedron" );
14
15       setSize( 275, 150 );
16       setVisible( true );
17    }
18
19    // draw tetrahedron
20    public void paint( Graphics g )
21    {
22       super.paint( g );
23
```

```
24          int baseX[] = { 110, 150, 50, 110 };
25          int baseY[] = { 90, 130, 130, 90 };
26          int x = 110, y = 40;
27
28          Graphics2D g2d = ( Graphics2D ) g;
29          g2d.setColor( Color.red );
30
31          GeneralPath tetrahedron = new GeneralPath();
32          tetrahedron.moveTo( baseX[ 0 ], baseY[ 0 ] );
33
34          for ( int i = 1; i < 4; i++ ) {
35             tetrahedron.lineTo( x, y );
36             tetrahedron.moveTo( baseX[ i - 1 ], baseY[ i - 1 ] );
37             tetrahedron.lineTo( baseX[ i ], baseY[ i ] );
38          }
39
40          tetrahedron.closePath();
41          g2d.draw( tetrahedron );
42       }
43
44       public static void main( String args[] )
45       {
46          Tetrahedron application = new Tetrahedron();
47          application.setDefaultCloseOperation( EXIT_ON_CLOSE );
48       }
49
50    }  // end class Pyramid
```

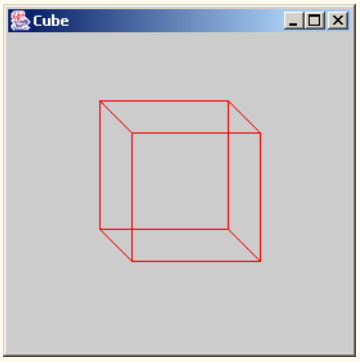


12.18 Write a program that draws a cube. Use class `GeneralPath` and method `draw` of class `Graphics2D`.

ANS:

```
1  // Exercise 12.18 Solution: Cube.java
2  // Program draws a cube.
3  import java.awt.*;
4  import java.awt.geom.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Cube extends JFrame {
9
10    // constructor
11    public Cube()
12    {
```

```
13          super( "Cube" );
14
15          setSize( 275, 275 );
16          setVisible( true );
17       }
18
19       // draw cube
20       public void paint( Graphics g )
21       {
22          super.paint( g );
23
24          // one base
25          int base1X[] = { 100, 100, 200, 200, 100 };
26          int base1Y[] = { 100, 200, 200, 100, 100 };
27
28          // second base
29          int base2X[] = { 75, 75, 175, 175, 75 };
30          int base2Y[] = { 75, 175, 175 ,75, 75 };
31
32          Graphics2D g2d = ( Graphics2D ) g;
33          g2d.setColor( Color.red );
34
35          GeneralPath cube = new GeneralPath();
36
37          for ( int i = 1; i <= 4; i++ ) {
38             // create the first base
39             cube.moveTo( base1X[ i - 1 ], base1Y[ i - 1 ] );
40             cube.lineTo( base1X[ i ], base1Y[ i ] );
41
42             // create the second base
43             cube.moveTo( base2X[ i - 1 ], base2Y[ i - 1 ] );
44             cube.lineTo( base2X[ i ], base2Y[ i ] );
45
46             // create the lines between the bases
47             cube.moveTo( base1X[ i ], base1Y[ i ] );
48             cube.lineTo( base2X[ i ], base2Y[ i ] );
49          }
50
51          g2d.draw( cube );
52       }
53
54       public static void main( String args[] )
55       {
56          Cube application = new Cube();
57          application.setDefaultCloseOperation( EXIT_ON_CLOSE );
58       }
59
60    }  // end class Cube
```

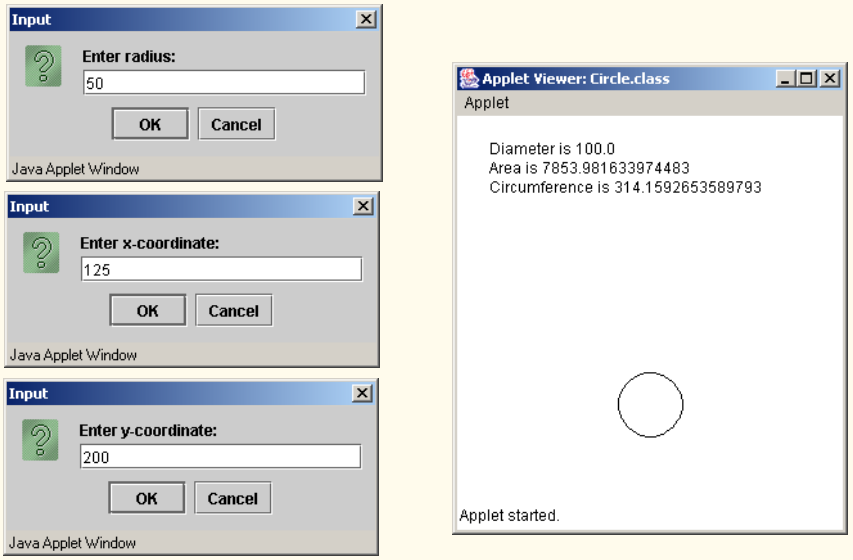


12.19 In Exercise 3.10, you wrote an applet that input the radius of a circle from the user and displayed the circle's diameter, circumference and area. Modify your solution to Exercise 3.10 to read a set of coordinates in addition to the radius. Then draw the circle, and display the circle's diameter, circumference and area, using an `Ellipse2D.Double` object to represent the circle and method `draw` of class `Graphics2D` to display the circle.

ANS:

```
1  // Exercise 12.19 Solution: Circle.java
2  // Program calculates the area, circumference
3  // and diameter for a circle and draws the circle
4  import java.awt.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Circle extends JApplet {
9     private double radius;
10    private int x, y;
11
12    // initialize applet by obtaining values from user
13    public void init()
14    {
15       String inputRadius, inputX, inputY;
16
17       // read from user as String
18       inputRadius = JOptionPane.showInputDialog( "Enter radius:" );
19       inputX = JOptionPane.showInputDialog( "Enter x-coordinate:" );
20       inputY = JOptionPane.showInputDialog( "Enter y-coordinate:" );
21
22       // convert number from type String to type int
23       radius = Double.parseDouble( inputRadius );
24       x = Integer.parseInt( inputX );
25       y = Integer.parseInt( inputY );
26
27    }  // end method init
28
29    // draw results on applet's background
30    public void paint( Graphics g )
31    {
32       Graphics2D g2d = ( Graphics2D ) g;
```

```
33
34          g.drawString( "Diameter is " + ( 2 * radius ), 25, 30 );
35          g.drawString( "Area is " + ( Math.PI * radius * radius ), 25, 45 );
36          g.drawString( "Circumference is " +
37             ( 2 * Math.PI * radius ), 25, 60 );
38
39          g2d.draw( new Ellipse2D.Double( x, y, radius, radius ) );
40
41       }  // end method paint
42
43    }  // end class Circle
```

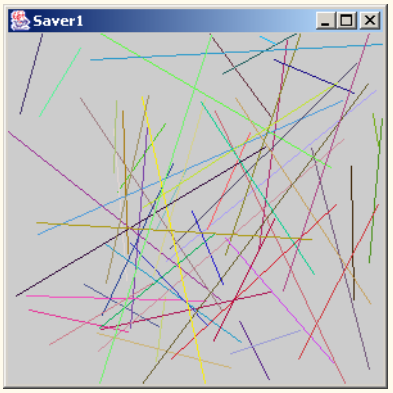


12.20 Write an application that simulates a screen saver. The application should randomly draw lines using method `drawLine` of class `Graphics`. After drawing 100 lines, the application should clear itself and start drawing lines again. To allow the program to draw continuously, place a call to `repaint` as the last line in method `paint`. Do you notice any problems with this on your system?

ANS:

```
1  // Exercise 12.20 Solution: Saver1.java
2  // Program simulates a simple screen saver
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Saver1 extends JFrame {
9     private final int DELAY = 9999999;
10
11    // constructor sets window's title bar string and dimensions
12    public Saver1()
13    {
```

```
14          super( "Saver1" );
15
16          setSize( 300, 300 );
17          setVisible( true );
18       }
19
20       // draw lines
21       public void paint( Graphics g )
22       {
23          super.paint( g );
24
25          int x, y, x1, y1;
26
27          // draw 100 random lines
28          for ( int i = 0; i < 100; i++ ) {
29
30             x = ( int ) ( Math.random() * 300 );
31             y = ( int ) ( Math.random() * 300 );
32             x1 = ( int ) ( Math.random() * 300 );
33             y1 = ( int ) ( Math.random() * 300 );
34
35             g.setColor( new Color( ( float ) Math.random(),
36                ( float ) Math.random(), ( float ) Math.random() ) );
37             g.drawLine( x, y, x1, y1 );
38
39             // slow the drawing down. the body of the for loop is empty
40             for ( int q = 1; q < DELAY; q++ ) ;
41          }
42
43          repaint();
44
45       }  // end method paint
46
47       public static void main( String args[] )
48       {
49          Saver1 application = new Saver1();
50          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
51       }
52
53    }  // end class Saver1
```



12.21 Here is a peek ahead. Package `javax.swing` contains a class called `Timer` that is capable of calling method `actionPerformed` of interface `ActionListener` at a fixed time interval (specified in milliseconds). Modify your solution to Exercise 12.20 to remove the call to `repaint` from method `paint`. Declare your class so it implements `ActionListener`. (The `actionPerformed` method should simply call `repaint`.) Declare an instance variable of type `Timer` called `timer` in your class. In the constructor for your class, write the following statements:

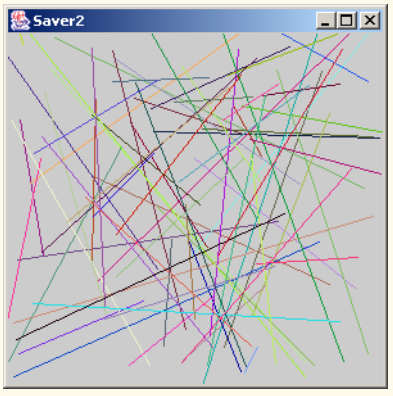
```
timer = new Timer( 1000, this );
timer.start();
```

This creates an instance of class `Timer` that will call `this` object's `actionPerformed` method every 1000 milliseconds (i.e., every second).

ANS:

```
1  // Exercise 12.21 Solution: Saver2.java
2  // Program simulates a simple screen saver
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Saver2 extends JFrame implements ActionListener {
9     private final int DELAY = 9999999;
10    private Timer timer;
11
12    // constructor sets window's title bar string and dimensions
13    public Saver2()
14    {
15       super( "Saver2" );
16
17       timer = new Timer( 1000, this ); // create the timer
18       timer.start();
19
20       setSize( 300, 300 );
21       setVisible( true );
22    }
23
```

```
24      // draw lines
25      public void paint( Graphics g )
26      {
27         super.paint( g );
28
29         int x, y, x1, y1;
30
31         for ( int i = 0; i < 100; i++ ) {
32
33            x = ( int ) ( Math.random() * 300 );
34            y = ( int ) ( Math.random() * 300 );
35            x1 = ( int ) ( Math.random() * 300 );
36            y1 = ( int ) ( Math.random() * 300 );
37
38            g.setColor( new Color( ( float ) Math.random(),
39               ( float ) Math.random(), ( float ) Math.random() ) );
40            g.drawLine( x, y, x1, y1 );
41
42            // slow the drawing down. the body of the for loop is empty
43            for ( int q = 1; q < DELAY; q++ ) ;
44         }
45
46      }  // end method paint
47
48      public static void main( String args[] )
49      {
50         Saver2 application = new Saver2();
51         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
52      }
53
54      public void actionPerformed( ActionEvent actionEvent )
55      {
56         repaint();
57      }
58
59   }  // end class Saver2
```

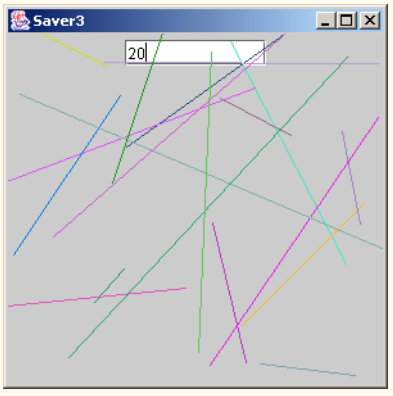


12.22 Modify your solution to Exercise 12.21 to enable the user to enter the number of random lines that should be drawn before the application clears itself and starts drawing lines again. Use a `JTextField` to obtain the value. The user should be able to type a new number into the `JTextField` at any time during the program's execution. Use an inner class to perform event handling for the `JTextField`.

ANS:

```
// Exercise 12.22 Solution: Saver3.java
// Program simulates a simple screen saver
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class Saver3 extends JFrame implements ActionListener {
   private final int DELAY = 9999999;
   private Timer timer;
   private int numberLines;

   private JTextField input;

   // constructor sets window's title bar string and dimensions
   public Saver3()
   {
      super( "Saver3" );

      numberLines = 100; // initialize the number of lines to draw

      timer = new Timer( 1000, this ); // create the timer
      timer.start();

      input = new JTextField( 10 );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      container.add( input );

      input.addActionListener(

         new ActionListener() { // anonymous inner class

            public void actionPerformed( ActionEvent event ) {

               numberLines = Integer.parseInt( input.getText() );
            }

         } // end anonymous inner class

      ); // end call to addActionListner

      setSize( 300, 300 );
      setVisible( true );
   }

```

```
48      // draw lines
49      public void paint( Graphics g )
50      {
51         super.paint( g );
52
53         int x, y, x1, y1;
54
55         for ( int i = 0; i < numberLines; i++ ) {
56
57            x = ( int ) ( Math.random() * 300 );
58            y = ( int ) ( Math.random() * 300 );
59            x1 = ( int ) ( Math.random() * 300 );
60            y1 = ( int ) ( Math.random() * 300 );
61
62            g.setColor( new Color( ( float ) Math.random(),
63               ( float ) Math.random(), ( float ) Math.random() ) );
64            g.drawLine( x, y, x1, y1 );
65
66            // slow the drawing down. the body of the for loop is empty
67            for ( int q = 1; q < DELAY; q++ ) ;
68         }
69
70      }  // end method paint
71
72      public static void main( String args[] )
73      {
74         Saver3 application = new Saver3();
75         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
76      }
77
78      public void actionPerformed( ActionEvent actionEvent )
79      {
80         repaint();
81      }
82
83   }  // end class Saver3
```



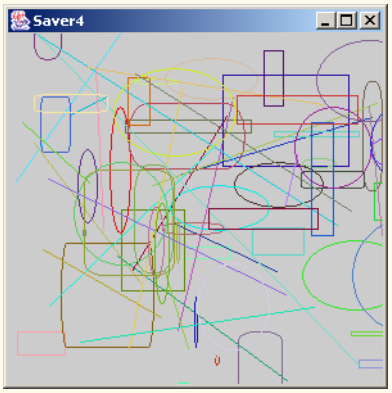
12.23 Modify your solution to Exercise 12.21 such that it uses random number generation to choose different shapes to display. Use methods of class `Graphics`.

ANS:

```
// Exercise 12.23 Solution: Saver4.java
// Program simulates a simple screen saver
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;

public class Saver4 extends JFrame implements ActionListener {
   private final int DELAY = 9999999;
   private Timer timer;

   // constructor sets window's title bar string and dimensions
   public Saver4()
   {
      super( "Saver4" );

      timer = new Timer( 1000, this ); // create the timer
      timer.start();

      setSize( 300, 300 );
      setVisible( true );
   }

   // draw shapes
   public void paint( Graphics g )
   {
      super.paint( g );

      int shape;

      for ( int i = 0; i < 100; i++ ) {

         shape = ( int ) ( Math.random() * 4 ); // pick a random shape

         switch( shape ) {
             case 0:
               makeLine( g );
               break;
            case 1:
               makeRect( g );
               break;
            case 2:
               makeOval( g );
               break;
            case 3:
               makeRoundRect( g );
               break;
         }

```

```
50            // slow the drawing down. the body of the for loop is empty
51            for ( int q = 1; q < DELAY; q++ ) ;
52         }
53
54      }  // end method paint
55
56      public static void main( String args[] )
57      {
58         Saver4 application = new Saver4();
59         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
60      }
61
62      public void actionPerformed( ActionEvent actionEvent )
63      {
64         repaint();
65      }
66
67      // draw a random lines
68      private void makeLine( Graphics g ) {
69
70         int x = ( int ) ( Math.random() * 300 );
71         int y = ( int ) ( Math.random() * 300 );
72         int x1 = ( int ) ( Math.random() * 300 );
73         int y1 = ( int ) ( Math.random() * 300 );
74
75         g.setColor( new Color( ( float ) Math.random(),
76            ( float ) Math.random(), ( float ) Math.random() ) );
77         g.drawLine( x, y, x1, y1 );
78      }
79
80      // draw a random rectangle
81      private void makeRect( Graphics g ) {
82
83         int x = ( int ) ( Math.random() * 300 );
84         int y = ( int ) ( Math.random() * 300 );
85         int width = ( int ) ( Math.random() * 100 );
86         int height = ( int ) ( Math.random() * 100 );
87
88         g.setColor( new Color( ( float ) Math.random(),
89            ( float ) Math.random(), ( float ) Math.random() ) );
90         g.drawRect( x, y, width, height );
91      }
92
93      // draw a random oval
94      private void makeOval( Graphics g ) {
95
96         int x = ( int ) ( Math.random() * 300 );
97         int y = ( int ) ( Math.random() * 300 );
98         int width = ( int ) ( Math.random() * 100 );
99         int height = ( int ) ( Math.random() * 100 );
100
101        g.setColor( new Color( ( float ) Math.random(),
102           ( float ) Math.random(), ( float ) Math.random() ) );
103        g.drawOval( x, y, width, height );
104     }
```

```
105
106    // draw a random rounded rectangle
107    private void makeRoundRect( Graphics g ) {
108
109       int x = ( int ) ( Math.random() * 300 );
110       int y = ( int ) ( Math.random() * 300 );
111       int width = ( int ) ( Math.random() * 100 );
112       int height = ( int ) ( Math.random() * 100 );
113       int arcWidth = ( int ) ( Math.random() * width );
114       int arcHeight = ( int ) ( Math.random() * height );
115
116       g.setColor( new Color( ( float ) Math.random(),
117          ( float ) Math.random(), ( float ) Math.random() ) );
118       g.drawRoundRect( x, y, width, height, arcWidth, arcHeight );
119    }
120
121 }  // end class Saver4
```



12.24 Modify your solution to Exercise 12.23 to use classes and drawing capabilities of the Java2D API. For shapes such as rectangles and ellipses, draw them with randomly generated gradients. Use class `GradientPaint` to generate the gradient.

ANS:

```
1  // Exercise 12.24 Solution: Saver5.java
2  // Program simulates a simple screen saver
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Saver5 extends JFrame implements ActionListener {
9     private final int DELAY = 9999999;
10    private Timer timer;
11
12    // constructor sets window's title bar string and dimensions
13    public Saver5()
14    {
```

```
15          super( "Saver5" );
16
17          timer = new Timer( 1000, this ); // create a new timer
18          timer.start();
19
20          setSize( 300, 300 );
21          setVisible( true );
22       }
23
24       // draw shapes
25       public void paint( Graphics g )
26       {
27          super.paint( g );
28
29          int shape;
30
31          for ( int i = 0; i < 100; i++ ) {
32
33             shape = ( int ) ( Math.random() * 4 ); // pick a random shape
34
35             switch( shape ) {
36                case 0:
37                   makeLine( g );
38                   break;
39                case 1:
40                   makeRect( g );
41                   break;
42                case 2:
43                   makeOval( g );
44                   break;
45                case 3:
46                   makeRoundRect( g );
47                   break;
48             }
49
50             // slow the drawing down. the body of the for loop is empty
51             for ( int q = 1; q < DELAY; q++ ) ;
52          }
53
54       }  // end method paint
55
56       public static void main( String args[] )
57       {
58          Saver5 application = new Saver5();
59          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
60       }
61
62       public void actionPerformed( ActionEvent actionEvent )
63       {
64          repaint();
65       }
66
67       // draw a random lines
68       private void makeLine( Graphics g ) {
```

```
69
70          Graphics2D g2d = ( Graphics2D ) g;
71
72          double x = Math.random() * 300;
73          double y = Math.random() * 300;
74          double x1 = Math.random() * 300;
75          double y1 = Math.random() * 300;
76
77          g2d.setPaint( new GradientPaint( (int)x, (int)y,
78             new Color( ( float ) Math.random(), ( float ) Math.random(),
79             ( float ) Math.random() ), (int)x1, (int)y1,
80             new Color( ( float ) Math.random(), ( float ) Math.random(),
81             ( float ) Math.random() ), true ) );
82
83          g2d.draw( new Line2D.Double( x, y, x1, y1 ) );
84       }
85
86       // draw a random rectangle
87       private void makeRect( Graphics g ) {
88
89          Graphics2D g2d = ( Graphics2D ) g;
90
91          double x = Math.random() * 300;
92          double y = Math.random() * 300;
93          double width = Math.random() * 100;
94          double height = Math.random() * 100;
95
96          g2d.setPaint( new GradientPaint( (int)x, (int)y,
97             new Color( ( float ) Math.random(), ( float ) Math.random(),
98             ( float ) Math.random() ), (int)(x + width), (int)(y + height),
99             new Color( ( float ) Math.random(), ( float ) Math.random(),
100            ( float ) Math.random() ), true ) );
101
102         g2d.draw( new Rectangle2D.Double( x, y, width, height ) );
103      }
104
105      // draw a random oval
106      private void makeOval( Graphics g ) {
107
108         Graphics2D g2d = ( Graphics2D ) g;
109
110         double x = Math.random() * 300;
111         double y = Math.random() * 300;
112         double width = Math.random() * 100;
113         double height = Math.random() * 100;
114
115         g2d.setPaint( new GradientPaint( (int)x, (int)y,
116            new Color( ( float ) Math.random(), ( float ) Math.random(),
117            ( float ) Math.random() ), (int)(x + width), (int)(y + height),
118            new Color( ( float ) Math.random(), ( float ) Math.random(),
119            ( float ) Math.random() ), true ) );
120
121         g2d.draw( new Ellipse2D.Double( x, y, width, height ) );
122      }
```

```
123
124    // create a random rounded rectangle
125    private void makeRoundRect( Graphics g ) {
126
127       Graphics2D g2d = ( Graphics2D ) g;
128
129       double x = Math.random() * 300;
130       double y = Math.random() * 300;
131       double width = Math.random() * 100;
132       double height = Math.random() * 100;
133       double arcWidth = Math.random() * width;
134       double arcHeight = Math.random() * height;
135
136       g2d.setPaint( new GradientPaint( (int)x, (int)y,
137          new Color( ( float ) Math.random(), ( float ) Math.random(),
138          ( float ) Math.random() ), (int)(x + width), (int)(y + height),
139          new Color( ( float ) Math.random(), ( float ) Math.random(),
140          ( float ) Math.random() ), true ) );
141
142       g2d.draw( new RoundRectangle2D.Double( x, y, width, height,
143          arcWidth, arcHeight ) );
144    }
145
146 }  // end class Saver5
```

12.25 Modify your solution to Exercise 7.21—*Turtle Graphics*—to add a graphical user interface using `JTextFields` and `JButtons`. Also, draw lines rather than drawing asterisks (*). When the turtle graphics program specifies a move, translate the number of positions into a number of pixels on the screen by multiplying the number of positions by 10 (or any value you choose). Implement the drawing with Java2D API features.

ANS:

```
1  // Exercise 12.25: TurtleGraphics.java
2  // Drawing turtle graphics based on turtle commands.
3  import java.awt.*;
4  import java.awt.event.*;
```

```
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class TurtleGraphics extends JApplet {
9
10    // an array of turtle commands
11    private static final int MAXCOMMANDS = 10;
12    private int commandArray[][] = new int [ MAXCOMMANDS ][ 2 ];
13
14    private int direction, count, xPos, yPos; // state variables
15    private boolean penDown;
16
17    private JTextField input;
18    private JButton down, up, move, turnRight, turnLeft, print, clear;
19
20    private static final int PEN_UP = 1, PEN_DOWN = 2,
21       TURN_RIGHT = 3, TURN_LEFT = 4, MOVE = 5;
22
23    private Graphics2D g2d;
24
25    // set up GUI components and initialize instance variables
26    public void init()
27    {
28       up = new JButton( "Pen Up" );
29       up.addActionListener(
30
31          new ActionListener () { // anonymous inner class
32
33             public void actionPerformed( ActionEvent event ) {
34                commandArray[ count ][ 0 ] = PEN_UP;
35                count++;
36
37                if ( count == MAXCOMMANDS ) {
38                   up.setEnabled( false );
39                   down.setEnabled( false );
40                   move.setEnabled( false );
41                   input.setEnabled( false );
42                   turnRight.setEnabled( false );
43                   turnLeft.setEnabled( false );
44                }
45             }
46
47          } // end anonymous inner class
48
49       );
50
51       down = new JButton( "Pen Down" );
52       down.addActionListener(
53
54          new ActionListener () { // anonymous inner class
55
56             public void actionPerformed( ActionEvent event ) {
57                commandArray[ count ][ 0 ] = PEN_DOWN;
58                count++;
```

```
59
60                if ( count == MAXCOMMANDS ) {
61                   up.setEnabled( false );
62                   down.setEnabled( false );
63                   move.setEnabled( false );
64                   input.setEnabled( false );
65                   turnRight.setEnabled( false );
66                   turnLeft.setEnabled( false );
67                }
68             }
69
70          } // end anonymous inner class
71
72       );
73
74       move = new JButton( "Move forward" );
75       move.addActionListener(
76
77          new ActionListener () { // anonymous inner class
78
79             public void actionPerformed( ActionEvent event ) {
80
81                // get the distance to move
82                int spaces = Integer.parseInt( input.getText() );
83                input.setText( "" );
84
85                commandArray[ count ][ 0 ] = MOVE;
86                commandArray[ count ][ 1 ] = spaces;
87
88                count++;
89
90                if ( count == MAXCOMMANDS ) {
91                   up.setEnabled( false );
92                   down.setEnabled( false );
93                   move.setEnabled( false );
94                   input.setEnabled( false );
95                   turnRight.setEnabled( false );
96                   turnLeft.setEnabled( false );
97                }
98             }
99
100         } // end anonymous inner class
101
102      );
103
104      input = new JTextField( 4 );
105      input.addActionListener(
106
107         new ActionListener () { // anonymous inner class
108
109            public void actionPerformed( ActionEvent event ) {
110
111               // get the distance to move
112               int spaces = Integer.parseInt( input.getText() );
```

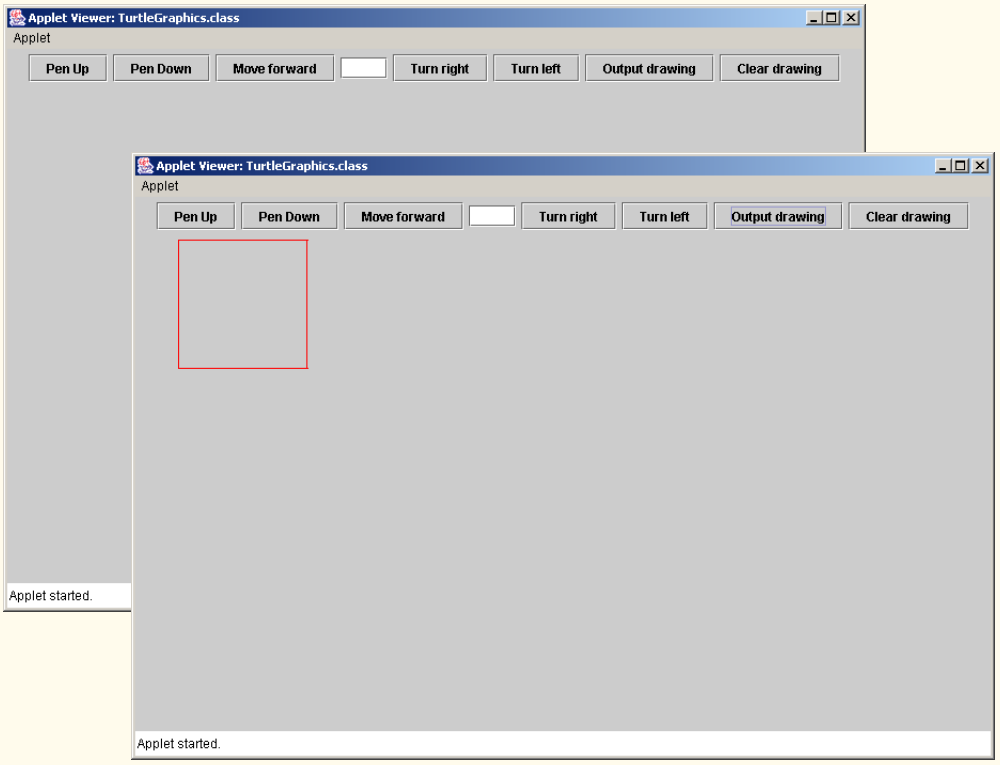
```
113                input.setText( "" );
114
115                commandArray[ count ][ 0 ] = MOVE;
116                commandArray[ count ][ 1 ] = spaces;
117
118                count++;
119
120                if ( count == MAXCOMMANDS ) {
121                   up.setEnabled( false );
122                   down.setEnabled( false );
123                   move.setEnabled( false );
124                   input.setEnabled( false );
125                   turnRight.setEnabled( false );
126                   turnLeft.setEnabled( false );
127                }
128             }
129
130          } // end anonymous inner class
131
132       );
133
134       turnRight = new JButton( "Turn right" );
135       turnRight.addActionListener(
136
137          new ActionListener () { // anonymous inner class
138
139             public void actionPerformed( ActionEvent event ) {
140                commandArray[ count ][ 0 ] = TURN_RIGHT;
141
142                count++;
143
144                if ( count == MAXCOMMANDS ) {
145                   up.setEnabled( false );
146                   down.setEnabled( false );
147                   move.setEnabled( false );
148                   input.setEnabled( false );
149                   turnRight.setEnabled( false );
150                   turnLeft.setEnabled( false );
151                }
152             }
153
154          } // end anonymous inner class
155
156       );
157
158       turnLeft = new JButton( "Turn left" );
159       turnLeft.addActionListener(
160
161          new ActionListener () { // anonymous inner class
162
163             public void actionPerformed( ActionEvent event ) {
164                commandArray[ count ][ 0 ] = TURN_LEFT;
165
166                count++;
```

```
167
168                if ( count == MAXCOMMANDS ) {
169                   up.setEnabled( false );
170                   down.setEnabled( false );
171                   move.setEnabled( false );
172                   input.setEnabled( false );
173                   turnRight.setEnabled( false );
174                   turnLeft.setEnabled( false );
175                }
176             }
177
178          } // end anonymous inner class
179
180       );
181
182       print = new JButton( "Output drawing" );
183       print.addActionListener(
184
185          new ActionListener () { // anonymous inner class
186
187             public void actionPerformed( ActionEvent event ) {
188
189                executeCommands(); // run the commands
190             }
191
192          } // end anonymous inner class
193
194       );
195
196       clear = new JButton( "Clear drawing" );
197       clear.addActionListener(
198
199          new ActionListener() { // anonymous inner class
200
201             public void actionPerformed( ActionEvent event ) {
202
203                repaint(); // clear the applet
204
205                clearCommands(); // clear the command array
206
207                up.setEnabled( true );
208                down.setEnabled( true );
209                move.setEnabled( true );
210                input.setEnabled( true );
211                turnRight.setEnabled( true );
212                turnLeft.setEnabled( true );
213             }
214
215          } // end anonymous inner class
216
217       );
218
219       g2d = (Graphics2D) getGraphics();
220       g2d.setColor( Color.red );
```

```
221
222        Container container = getContentPane();
223        container.setLayout( new FlowLayout() );
224        container.add( up );
225        container.add( down );
226        container.add( move );
227        container.add( input );
228        container.add( turnRight );
229        container.add( turnLeft );
230        container.add( print );
231        container.add( clear );
232
233        clearCommands();
234
235        setSize( 800, 500 );
236
237     }  // end method init
238
239     public void clearCommands() {
240        count = 0;
241
242        // initialize array commandArray
243        for ( int i = 0; i < MAXCOMMANDS; i++ )
244           for ( int j = 0; j < 2; j++ )
245              commandArray[ i ][ j ] = 0;
246     }
247
248     public void executeCommands()
249     {
250        // reset the state variables
251        direction = 0;
252        xPos = 40;
253        yPos = 40;
254        penDown = false;
255
256        // continue executing commands until reach the end
257        for ( int commandNumber = 0; commandNumber < count;
258           commandNumber++ ) {
259
260           int command = commandArray[ commandNumber ][ 0 ];
261
262           // determine what command was entered  and peform desired action
263           switch ( command ) {
264              case PEN_UP:
265                 penDown = false;
266                 break;
267              case PEN_DOWN:
268                 penDown = true;
269                 break;
270              case TURN_RIGHT:
271                 direction = turnRight( direction );
272                 break;
273              case TURN_LEFT:
274                 direction = turnLeft( direction );
```

```
275                break;
276             case MOVE:
277                int distance = commandArray[ commandNumber ][ 1 ];
278                movePen( penDown, direction, distance );
279                break;
280
281          }  // end switch
282
283       }  // end for
284
285    } // end method executeCommands
286
287    // method to turn turtle to the right
288    public int turnRight( int d )
289    {
290       return --d < 0 ? 3 : d;
291    }
292
293    // method to turn turtle to the left
294    public int turnLeft( int d )
295    {
296       return ++d > 3 ? 0 : d;
297    }
298
299    // method to move the pen
300    public void movePen( boolean down, int dir, int dist )
301    {
302       // determine which way to move pen
303       switch ( dir ) {
304
305          case 0:   // move down
306             if ( down )
307                g2d.draw( new Line2D.Double(
308                   xPos, yPos, xPos, yPos + dist * 10 ) );
309             yPos += dist * 10;
310             break;
311
312          case 1:   // move right
313             if ( down )
314                g2d.draw( new Line2D.Double(
315                   xPos, yPos, xPos + dist * 10, yPos ) );
316             xPos += dist * 10;
317             break;
318
319          case 2:   // move up
320             if ( down )
321                g2d.draw( new Line2D.Double(
322                   xPos, yPos, xPos, yPos - dist * 10 ) );
323             yPos -= dist * 10;
324             break;
325
326          case 3:   // move left
327             if ( down )
328                g2d.draw( new Line2D.Double(
```

```
                    xPos, yPos, xPos - dist * 10, yPos ) );
            xPos -= dist * 10;
            break;

         } // end switch

      } // end method movePen

} // end class TurtleGraphics
```



12.26 Produce a graphical version of the Knight's Tour problem (Exercise 7.22, Exercise 7.23 and Exercise 7.26). As each move is made, the appropriate cell of the chessboard should be updated with the proper move number. If the result of the program is a *full tour* or a *closed tour*, the program should display an appropriate message. If you would like, use class `Timer` (see Exercise 11.24) to help animate the Knight's Tour. Every second, the next move should be made.

ANS:

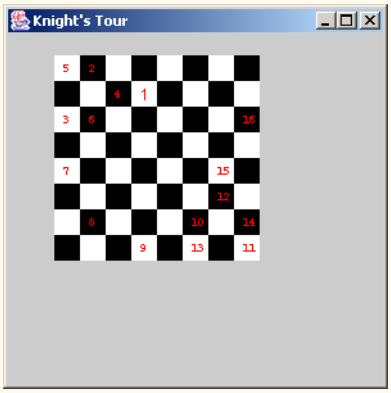
```
// Exercise 12.26 Solution: Knight.java
// Knight's Tour - access version runs one tour
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import javax.swing.*;
```

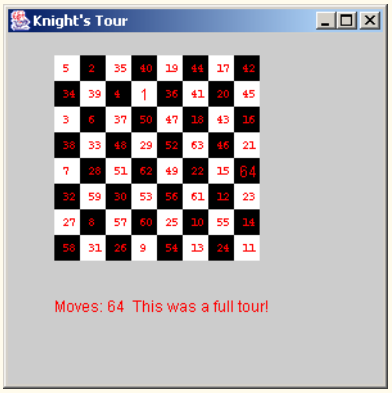
```
 7
 8  public class Knight extends JFrame {
 9     private final int DELAY = 8000000, INSET = 40;
10     private boolean done, closed;
11     private int board[][], currentRow, currentColumn, moveNumber, testRow,
12        testColumn, count, minRow, minColumn, minAccess, accessNumber,
13        firstMoveRow, firstMoveColumn;
14     private int access[][] = { { 2, 3, 4, 4, 4, 4, 3, 2 },
15                                { 3, 4, 6, 6, 6, 6, 4, 3 },
16                                { 4, 6, 8, 8, 8, 8, 6, 4 },
17                                { 4, 6, 8, 8, 8, 8, 6, 4 },
18                                { 4, 6, 8, 8, 8, 8, 6, 4 },
19                                { 4, 6, 8, 8, 8, 8, 6, 4 },
20                                { 3, 4, 6, 6, 6, 6, 4, 3 },
21                                { 2, 3, 4, 4, 4, 4, 3, 2 } },
22        horizontal[] = { 2, 1, -1, -2, -2, -1, 1, 2 },
23        vertical[] = { -1, -2, -2, -1, 1, 2, 2, 1 };
24
25     // constructor sets up the play
26     public Knight()
27     {
28        super( "Knight's Tour" );
29
30        minAccess = 9;
31        board = new int[ 8 ][ 8 ];
32        currentRow = ( int ) ( Math.random() * 8 );
33        currentColumn = ( int ) ( Math.random() * 8 );
34        firstMoveRow = currentRow;
35        firstMoveColumn = currentColumn;
36
37        setSize( 300, 300 );
38        setVisible( true );
39     }
40
41     // returns if spot is within range and empty
42     public boolean validMove( int row, int column )
43     {
44        return ( row >= 0 && row < 8 && column >= 0 && column < 8
45                 && board[ row ][ column ] == 0 );
46     }
47
48     // draw chess board with tour
49     public void paint( Graphics g )
50     {
51        super.paint( g );
52
53        int x1 = 0, y1 = 0;
54
55        // draw black and white patterned chess board
56        for ( int j = 0; j <= 7; j++ ) {
57
58           for ( int k = 0; k <= 7; k++ ) {
59
60              if ( ( k + j ) % 2 == 1 )
```

```
61                g.setColor( Color.black );
62
63             else
64                g.setColor( Color.white );
65
66             g.fillRect( x1 + INSET, y1 + INSET, 20, 20 );
67             x1 += 20;
68          }
69
70          y1 += 20;
71          x1 = 0;
72       }
73
74       // place first move on board the red font is used for emphasis
75       board[ currentRow ][ currentColumn ] = ++moveNumber;
76       g.setColor( Color.red );
77       g.drawString( "1", INSET + 7 + 20 * currentRow,
78          INSET + 15 + 20 * currentColumn );
79
80       while ( !done ) {
81          accessNumber = minAccess;
82
83          // cycle through each column of the current row and
84          // determine the move with the minimum access number
85          for ( int moveType = 0; moveType < board.length; moveType++ ) {
86
87             testRow = currentRow + vertical[ moveType ];
88             testColumn = currentColumn + horizontal[ moveType ];
89
90             if ( validMove( testRow, testColumn ) ) {
91
92                if ( access[ testRow ][ testColumn ] < accessNumber ) {
93
94                   accessNumber = access[ testRow ][ testColumn ];
95                   minRow = testRow;
96                   minColumn = testColumn;
97                }
98
99                --access[ testRow ][ testColumn ];
100            }
101         }
102
103         // determine whether the next move has been found
104         if ( accessNumber == minAccess )
105            done = true;
106
107         else {
108            currentRow = minRow;
109            currentColumn = minColumn;
110            board[ currentRow ][ currentColumn ] = ++moveNumber;
111
112            Font oldFont = g.getFont();
113            Font newFont = new Font( "Monospaced", Font.BOLD, 9 );
114
```

```
115             // slow animation
116             for ( int h = 1; h <= DELAY; h++ )
117                ;     // do nothing
118
119             // draw the new move onto the board
120             if ( moveNumber != 64 ) {
121                g.setFont( newFont );
122                g.drawString( String.valueOf(
123                   board[ currentRow ][ currentColumn ] ),
124                   INSET + 7 + 20 * currentRow, INSET + 13 + 20 *
125                   currentColumn );
126             }
127
128             // emphasize move 64 (final one in board)
129             else {
130                g.setFont( oldFont );
131                g.drawString( String.valueOf(
132                   board[ currentRow ][ currentColumn ] ),
133                   INSET + 4 + 20 * currentRow, INSET + 15 + 20 *
134                   currentColumn );
135             }
136
137             g.setFont( oldFont );
138          }
139       }
140
141       // if this is the final move, determine whether it is possible to
142       // reach the first square and close the knights tour
143       if ( moveNumber == 64 )
144          for ( int moveType = 0; moveType < 8; moveType++ ) {
145             testRow = currentRow + vertical[ moveType ];
146             testColumn = currentColumn + horizontal[ moveType ];
147
148             if ( testRow == firstMoveRow &&
149                testColumn == firstMoveColumn ) {
150                closed = true;
151                break;
152             }
153          }
154
155       // draw result
156       if ( moveNumber == 64 && closed == true )
157          g.drawString( "Moves: " + moveNumber +
158          "  This was a CLOSED tour!", INSET + 0,
159             INSET + 200 );
160       else if ( moveNumber == 64 )
161          g.drawString( "Moves: " + moveNumber +
162             "  This was a full tour!", INSET + 0,
163             INSET + 200 );
164       else
165          g.drawString( "Moves: " + moveNumber +
166             "  This was not a full tour.", INSET + 0,
167             INSET + 200 );
168    }
```

```
169
170    public static void main( String args[] )
171    {
172       Knight app = new Knight();
173       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
174    }
175
176 }  // end class Knight
```





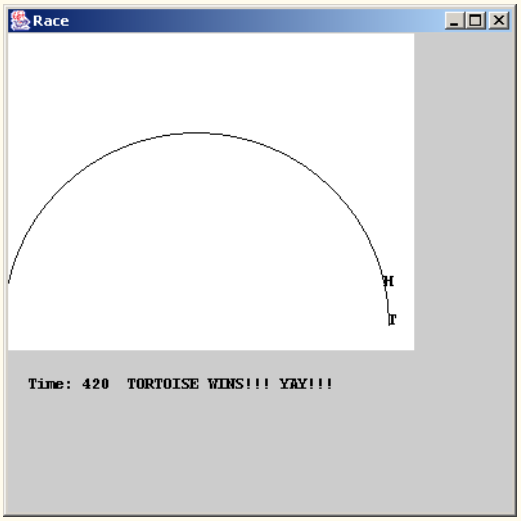
12.27 Produce a graphical version of the *Tortoise and the Hare* simulation (Exercise 7.15). Simulate the mountain by drawing an arc that extends from the bottom-left of the window to the top-right of the window. The tortoise and the hare should race up the mountain. Implement the graphical output so the tortoise and the hare are actually printed on the arc every move. [*Note:* Extend the length of the race from 70 to 300 to allow yourself a larger graphics area.]

ANS:

```
1  // Exercise 12.27 Solution: Race2.java
2  // Program simulates the race between the tortoise and the hare.For
3  // graphical purposes the race has been extended to 300. The mountain is
4  // represented by a semicircle using the equation for a
5  // circle: (x-h)^2 + (y-k)^2 = r^2. A slight adjustment was made in the
6  // equation to help compensate for the different directions of the
7  // coordinate systems.
8  import javax.swing.*;
9  import java.awt.*;
10 import java.awt.event.*;
11 import java.awt.geom.*;
12
13 public class Race2 extends JFrame {
14    private final int RACE_END = 300;
15    private int tortoise = 1, hare = 1, timer = 0;
16    private Font f;
17
18    public Race2()
19    {
20       super( "Race" );
21       setSize( 400, 400 );
```

```
22          f = new Font( "Monospaced", Font.BOLD, 12 );
23          setVisible( true );
24       }
25
26       public void paint( Graphics g )
27       {
28          super.paint( g );
29
30          g.setFont( f );
31
32          while ( tortoise != RACE_END && hare != RACE_END ) {
33
34             // slow animation
35             for ( int k = 1; k <= 100000; k++ )
36                ;   // do nothing
37
38             g.setColor( Color.white );
39             g.fillRect( 0, 0, 320, 270 );  // html size
40             g.setColor( Color.black );
41             g.drawArc( 0, 100, 300, 300, 0, 180 );
42
43             moveHare();
44             moveTortoise();
45             printCurrentPositions( g );
46             ++timer;
47          }
48
49          if ( tortoise >= hare )
50             g.drawString( "Time: " + timer + "  TORTOISE WINS!!! YAY!!!",
51                20, 300 );
52          else
53             g.drawString( "Time: " + timer + "  Hare wins. Yuch!", 20, 300 );
54       }
55
56       public void printCurrentPositions( Graphics g2 )
57       {
58          int yHare, yTortoise;
59
60          yHare = ( int ) ( 250 - Math.sqrt( 150 * 150 -
61                                Math.pow( hare - 150, 2 ) ) );
62
63          yTortoise = ( int ) ( 250 - Math.sqrt( 150 * 150 -
64                                Math.pow( tortoise - 150, 2 ) ) );
65
66          if ( yHare == yTortoise && hare == tortoise ) {
67             g2.drawString( "OUCH!", hare, yHare - 60 );
68             g2.drawString( "H", hare, yHare - 20 );  // make hare jump
69          }
70          else
71             g2.drawString( "H", hare, yHare );
72
73          g2.drawString( "T", tortoise, yTortoise );
74       }
75
```

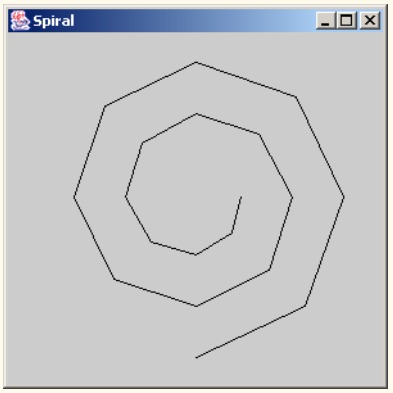
```
76     public void moveTortoise()
77     {
78        int x = ( int ) ( 1 + Math.random() * 10 );
79        int tortoiseMoves[] = { 3, 6 };
80
81        if ( x >= 1 && x <= 5 )             // fast plod
82           tortoise += tortoiseMoves[ 0 ];
83        else if ( x == 6 || x == 7 )        // slip
84           tortoise -= tortoiseMoves[ 1 ];
85        else                                // slow plod
86           ++tortoise;
87
88        if ( tortoise < 1 )
89           tortoise = 1;
90        else if ( tortoise > RACE_END )
91           tortoise = RACE_END;
92     }
93
94     public void moveHare()
95     {
96         int y = ( int ) ( 1 + Math.random() * 10 );
97         int hareMoves[] = { 9, 12, 2 };
98
99         if ( y == 3 || y == 4 )         // big hop
100           hare += hareMoves[ 0 ];
101        else if ( y == 5 )              // big slip
102           hare -= hareMoves[ 1 ];
103        else if ( y >= 6 && y <= 8 )    // small hop
104           ++hare;
105        else if ( y > 8 )               // small slip
106           hare -= hareMoves[ 2 ];
107
108        if ( hare < 1 )
109           hare = 1;
110        else if ( hare > RACE_END )
111           hare = RACE_END;
112    }
113
114    public static void main( String args[] )
115    {
116       Race2 application = new Race2();
117       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
118    }
119
120 } // end class Race2
```



12.28 Write a program that uses method `drawPolyline` to draw a spiral.
ANS:

```
1  // Exercise 12.28 Solution: Spiral.java
2  // Program creates a spiral.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class Spiral extends JFrame {
9     private int[] x = { 150, 235, 265, 228, 150,  79,  55,  86,
10                        150, 207, 225, 199, 150, 108,  95, 115,
11                        150, 178, 185, 171, 150, 136, 135, 143,
12                        150 };
13    private int[] y = { 275, 235, 150,  72,  45,  79, 150, 214,
14                        235, 207, 150, 101,  85, 108, 150, 185,
15                        195, 178, 150, 129, 125, 136, 150, 157,
16                        155 };
17
18    // constructor sets window's title bar string and dimensions
19    public Spiral()
20    {
21       super( "Spiral" );
22
23       setSize( 300, 300 );
24       setVisible( true );
25    }
26
27    public void paint( Graphics g ) {
28       super.paint( g );
29
```

```
30          g.drawPolyline( x, y, 19 );
31       }
32
33       public static void main( String args[] )
34       {
35          Spiral application = new Spiral();
36          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
37       }
38
39    }  // end class Spiral
```

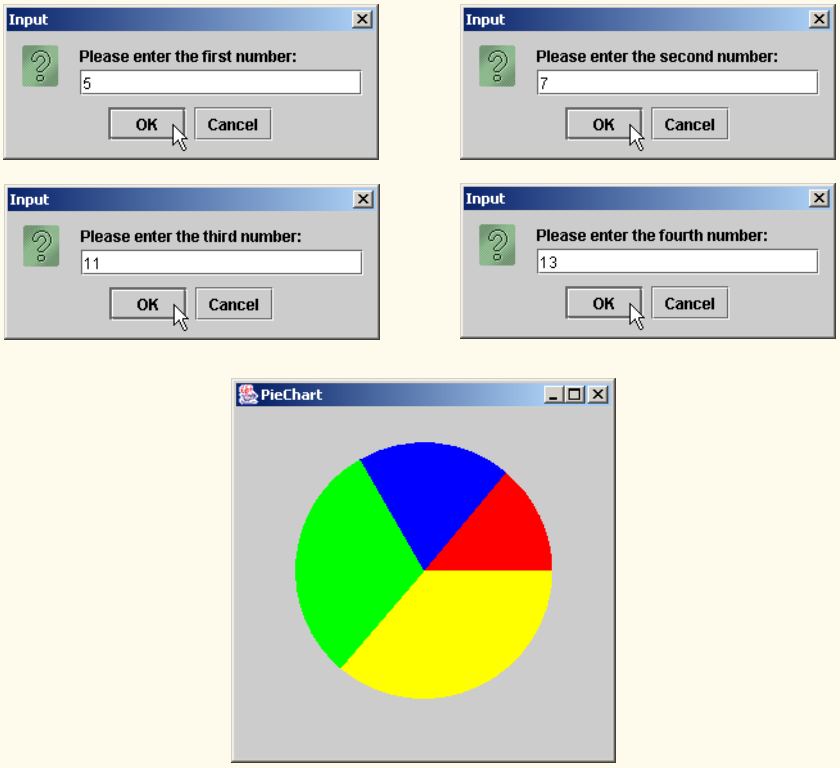


12.29 Write a program that inputs four numbers and graphs the numbers as a pie chart. Use class `Arc2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each piece of the pie in a separate color.

ANS:

```
1  // Exercise 12.29 Solution: PieChart.java
2  // Program creates a pie chart.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class PieChart extends JFrame {
9     private int one, two, three, four, sum;
10
11    // constructor sets window's title bar string and dimensions
12    public PieChart()
13    {
14       super( "PieChart" );
15
16       // input the four values
17       String first = JOptionPane.showInputDialog(
18          this, "Please enter the first number:" );
19       String second = JOptionPane.showInputDialog(
20          this, "Please enter the second number:" );
21       String third = JOptionPane.showInputDialog(
22          this, "Please enter the third number:" );
```

```
23          String fourth = JOptionPane.showInputDialog(
24             this, "Please enter the fourth number:" );
25
26          one = Integer.parseInt( first );
27          two = Integer.parseInt( second );
28          three = Integer.parseInt( third );
29          four = Integer.parseInt( fourth );
30          sum = one + two + three + four;
31
32          setSize( 300, 300 );
33          setVisible( true );
34       }
35
36       // scale the arc by the sum of the values
37       public void paint( Graphics g ) {
38          super.paint( g );
39
40          Graphics2D g2d = ( Graphics2D )g;
41
42          g2d.setColor( Color.red );
43          g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 0, 360.0 *
44             (double)one / sum, Arc2D.PIE ) );
45
46          g2d.setColor( Color.blue );
47          g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
48             (double)one / sum, 360.0 * (double)two / sum, Arc2D.PIE ) );
49
50          g2d.setColor( Color.green );
51          g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
52             (double)( two + one ) / sum, 360.0 *
53             (double)three / sum, Arc2D.PIE ) );
54
55          g2d.setColor( Color.yellow );
56          g2d.fill( new Arc2D.Double( 50, 50, 200, 200, 360.0 *
57             (double)( three + two + one ) / sum, 360.0 *
58             (double)four / sum, Arc2D.PIE ) );
59
60       }
61
62       public static void main( String args[] )
63       {
64          PieChart application = new PieChart();
65          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
66       }
67
68    }  // end class PieChart
```

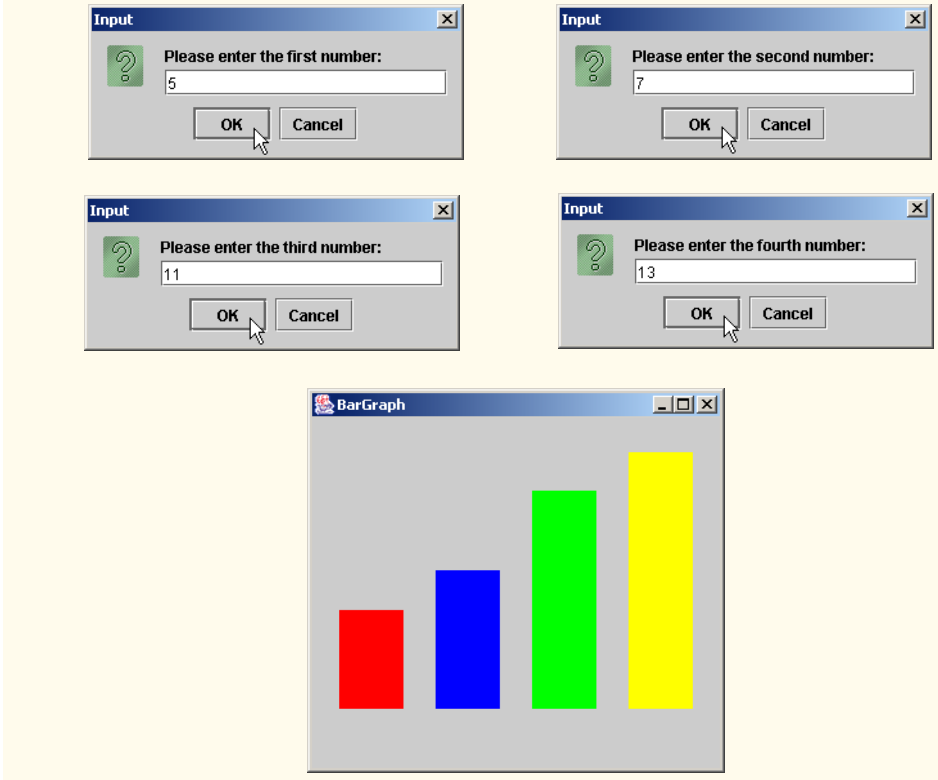


12.30 Write an applet that inputs four numbers and graphs the numbers as a bar graph. Use class `Rectangle2D.Double` and method `fill` of class `Graphics2D` to perform the drawing. Draw each bar in a different color.

ANS:

```
1  // Exercise 12.30 Solution: BarGraph.java
2  // Program creates a bar graph.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.awt.geom.*;
6  import javax.swing.*;
7
8  public class BarGraph extends JFrame {
9     private int one, two, three, four, max;
10
11    // constructor sets window's title bar string and dimensions
12    public BarGraph()
13    {
14       super( "BarGraph" );
15
16       // input the four numbers
17       String first = JOptionPane.showInputDialog(
18          this, "Please enter the first number:" );
```

```
19          String second = JOptionPane.showInputDialog(
20             this, "Please enter the second number:" );
21          String third = JOptionPane.showInputDialog(
22             this, "Please enter the third number:" );
23          String fourth = JOptionPane.showInputDialog(
24             this, "Please enter the fourth number:" );
25
26          one = Integer.parseInt( first );
27          two = Integer.parseInt( second );
28          three = Integer.parseInt( third );
29          four = Integer.parseInt( fourth );
30
31          // find the maximum
32          max = ( one > two ? one : two );
33          max = ( max > three ? max : three );
34          max = ( max > four ? max : four );
35
36          setSize( 325, 300 );
37          setVisible( true );
38       }
39
40       // scale each bar by the maximum value
41       public void paint( Graphics g ) {
42          super.paint( g );
43
44          Graphics2D g2d = ( Graphics2D )g;
45
46          g2d.setColor( Color.red );
47          g2d.fill( new Rectangle2D.Double( 25, 250 - 200 *
48             (double) one / max, 50, 200 * (double) one / max ) );
49
50          g2d.setColor( Color.blue );
51          g2d.fill( new Rectangle2D.Double( 100, 250 - 200 *
52             (double) two / max, 50, 200 * (double) two / max ) );
53
54          g2d.setColor( Color.green );
55          g2d.fill( new Rectangle2D.Double( 175, 250 - 200 *
56             (double) three / max, 50, 200 * (double) three / max ) );
57
58          g2d.setColor( Color.yellow );
59          g2d.fill( new Rectangle2D.Double( 250, 250 - 200 *
60             (double) four / max, 50, 200 * (double) four / max ) );
61
62       }
63
64       public static void main( String args[] )
65       {
66          BarGraph application = new BarGraph();
67          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
68       }
69
70    }  // end class BarGraph
```



13

Graphical User Interface Components: Part 1

Objectives

- To understand the design principles of graphical user interfaces (GUI).
- To be able to build graphical user interfaces.
- To understand the packages containing GUI-related components, event-handling classes and interfaces.
- To be able to create and manipulate buttons, labels, lists, text fields and panels.
- To understand mouse events and keyboard events.
- To understand and be able to use layout managers.

… the wisest prophets make sure of the event first.
Horace Walpole

Do you think I can listen all day to such stuff?
Lewis Carroll

Speak the affirmative; emphasize your choice by utter ignoring of all that you reject.
Ralph Waldo Emerson

You pays your money and you takes your choice.
Punch

Guess if you can, choose if you dare.
Pierre Corneille

All hope abandon, ye who enter here!
Dante Alighieri

Exit, pursued by a bear.
William Shakespeare

SELF-REVIEW EXERCISES

13.1 Fill in the blanks in each of the following statements:

a) Method _____ is called when the mouse is moved with no buttons pressed and an event listener is registered to handle the event.

ANS: `mouseMoved`.

b) Text that cannot be modified by the user is called _____ text.

ANS: uneditable (read-only)

c) A(n) _____ arranges GUI components in a `Container`.

ANS: layout manager.

d) The `add` method for attaching GUI components is a method of class _____.

ANS: `Container`

e) GUI is an acronym for _____.

ANS: graphical user interface

f) Method _____ is used to specify the layout manager for a container.

ANS: `setLayout`

g) A `mouseDragged` method call is preceded by a(n) _____ method call and followed by a(n) _____ method call.

ANS: `mousePressed`, `mouseReleased`

13.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) `BorderLayout` is the default layout manager for a content pane.

ANS: True.

b) When the mouse cursor is moved into the bounds of a GUI component, method `mouseOver` is called.

ANS: False. Method `mouseEntered` is called.

c) A `JPanel` cannot be added to another `JPanel`.

ANS: False. A `JPanel` can be added to another `JPanel`, because `JPanel` is an indirect subclass of `Component`. Therefore, a `JPanel` is a `Component`. Any `Component` can be added to a `Container`.

d) In a `BorderLayout`, two buttons added to the `NORTH` region will be placed side by side.

ANS: False. Only the last button added will be displayed. Remember that only one component should be added to each region in a `BorderLayout`.

e) When one is using `BorderLayout`, a maximum of five components should be displayed.

ANS: True.

13.3 Find the error(s) in each of the following and explain how to correct it (them).

a)
```
buttonName = JButton( "Caption" );
```

ANS: `new` is needed to create an object.

b)
```
JLabel aLabel, JLabel;      // create references
```

ANS: `JLabel` is a class name and cannot be used as a variable name.

c)
```
txtField = new JTextField( 50, "Default Text" );
```

ANS: The arguments passed to the constructor are reversed. The `String` must be passed first.

d)
```
Container container = getContentPane();
setLayout( new BorderLayout() );
button1 = new JButton( "North Star" );
button2 = new JButton( "South Pole" );
container.add( button1 );
container.add( button2 );
```

ANS: `BorderLayout` has been set, and components are being added without specifying the region so both are added to the center region. Proper `add` statements might be

```
container.add( button1, BorderLayout.NORTH );
container.add( button2, BorderLayout.SOUTH );
```

EXERCISES

13.4 Fill in the blanks in each of the following statements:

a) The `JTextField` class directly extends class _____.

ANS: `JTextComponent`.

b) List three layout managers from this chapter: _____, _____ and _____.

ANS: `FlowLayout`, `BorderLayout` and `GridLayout`.

c) `Container` method _____ attaches a GUI component to a container.

ANS: `add`.

d) Method _____ is called when a mouse button is released (without moving the mouse).

ANS: `mouseClicked`.

e) The _____ class is used to create a group of `JRadioButton`s.

ANS: `ButtonGroup`.

13.5 Determine whether each statement is *true* or *false*. If *false*, explain why.

a) Only one layout manager can be used per `Container`.

ANS: True.

b) GUI components can be added to a `Container` in any order in a `BorderLayout`.

ANS: True.

c) `JRadioButton`s provide a series of mutually exclusive options (i.e., only one can be `true` at a time).

ANS: True.

d) `Graphics` method `setFont` is used to set the font for text fields.

ANS: False. `Component` method `setFont` is used.

e) A `JList` displays a scrollbar if there are more items in the list than can be displayed.

ANS: False. A `JList` never provides a scrollbar.

f) A `Mouse` object has a method called `mouseDragged`.

ANS: False. A `Mouse` object is not provided by Java.

13.6 State whether each of the following is *true* or *false*. If *false*, explain why.

a) A `JApplet` does not have a content pane.

ANS: False. A `JApplet` does have a content pane.

b) A `JPanel` is a `JComponent`.

ANS: True.

c) A `JPanel` is a `Component`.

ANS: True.

d) A `JLabel` is a `Container`.

ANS: True.

e) A `JList` is a `JPanel`.

ANS: False. A `JList` is a `JComponent`.

f) An `AbstractButton` is a `JButton`.

ANS: False. A `JButton` is an `AbstractButton`.

g) A `JTextField` is an `Object`.

ANS: True.

h) `ButtonGroup` is a subclass of `JComponent`.

ANS: False. `ButtonGroup` inherits from `Object`.

13.7 Find any error(s) in each of the following and explain how to correct it (them).

a) `import javax.swing.*  // include swing package`

ANS: Semicolon is missing after the asterisk.

b) `panelObject.GridLayout( 8, 8 ); // set GridLayout`

ANS: The `GridLayout` constructor cannot be used in this manner. The correct statement should be:

```
panelObject.getContentPane().setLayout(
   new GridLayout( 8, 8 ) );
```

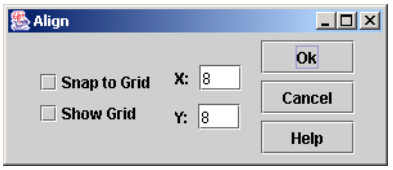
c) `container.setLayout( new FlowLayout( FlowLayout.DEFAULT ) );`

ANS: Class `FlowLayout` does not contain **`static`** constant `DEFAULT`.

d) `container.add( eastButton, EAST );  // BorderLayout`

ANS: `EAST` should be `BorderLayout.EAST`.

13.8 Create the following GUI. You do not have to provide any functionality.

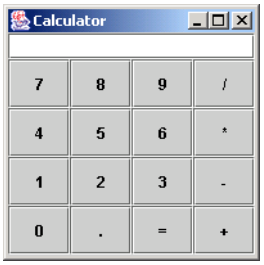


ANS:

```
1  // Exercise 13.8 Solution: Align.java
2  // Program creates a simple GUI.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Align extends JFrame {
7     private JButton okButton, cancelButton, helpButton;
8     private JTextField xValue, yValue;
9     private JCheckBox snapBox, showBox;
10    private JLabel xLabel, yLabel;
11    private JPanel checkPanel, buttonPanel, fieldPanel1,
12       fieldPanel2, fieldPanel;
13
14    // constructor sets up GUI
15    public Align()
16    {
17       super( "Align" );
18
19       // build checkPanel
20       snapBox = new JCheckBox( "Snap to Grid" );
21       showBox = new JCheckBox( "Show Grid" );
22       checkPanel = new JPanel();
23       checkPanel.setLayout( new GridLayout( 2 , 1 ) );
24       checkPanel.add( snapBox );
25       checkPanel.add( showBox );
26
27       // build field panel1
28       xLabel = new JLabel( "X: " );
29       xValue = new JTextField( "8", 3 );
30       fieldPanel1 = new JPanel();
31       fieldPanel1.setLayout( new FlowLayout() );
```

```
32          fieldPanel1.add( xLabel );
33          fieldPanel1.add( xValue );
34
35          // build field panel2
36          yLabel = new JLabel( "Y: " );
37          yValue = new JTextField( "8", 3 );
38          fieldPanel2 = new JPanel();
39          fieldPanel2.setLayout( new FlowLayout() );
40          fieldPanel2.add( yLabel );
41          fieldPanel2.add( yValue );
42
43          // build field panel
44          fieldPanel = new JPanel();
45          fieldPanel.setLayout( new BorderLayout() );
46          fieldPanel.add( fieldPanel1, BorderLayout.NORTH );
47          fieldPanel.add( fieldPanel2, BorderLayout.SOUTH );
48
49          // build button panel
50          okButton = new JButton( "Ok" );
51          cancelButton = new JButton( "Cancel" );
52          helpButton = new JButton( "Help" );
53          buttonPanel = new JPanel();
54          buttonPanel.setLayout( new GridLayout( 3, 1, 10, 5 ) );
55          buttonPanel.add( okButton );
56          buttonPanel.add( cancelButton );
57          buttonPanel.add( helpButton );
58
59          // set layout for applet
60          Container container = getContentPane();
61          container.setLayout(
62             new FlowLayout( FlowLayout.CENTER, 10, 5 ) );
63          container.add( checkPanel );
64          container.add( fieldPanel );
65          container.add( buttonPanel );
66
67          setSize( 300, 125 );
68          setVisible( true );
69
70       }  // end Align constructor
71
72       // execute application
73       public static void main( String args[] )
74       {
75          Align application = new Align();
76          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
77       }
78
79    }  // end class Align
```

13.9 Create the following GUI. You do not have to provide any functionality.

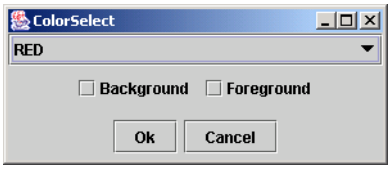


ANS:

```
1  // Solution exercise 13.9: Calculator.java
2  // Program creates a GUI that resembles a calculator.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class Calculator extends JFrame {
7     private JButton keys[];
8     private JPanel keyPad;
9     private JTextField lcd;
10
11    // constructor sets up GUI
12    public Calculator()
13    {
14       super( "Calculator" );
15
16       lcd = new JTextField( 20 );
17       lcd.setEditable( true );
18
19       keys = new JButton[ 16 ];
20
21       // initialize all non-digit key Buttons
22       for ( int i = 0; i <= 9; i++ )
23          keys[ i ] = new JButton( String.valueOf( i ) );
24
25       // initialize all digit key Buttons
26       keys[ 10 ] = new JButton( "/" );
27       keys[ 11 ] = new JButton( "*" );
28       keys[ 12 ] = new JButton( "-" );
29       keys[ 13 ] = new JButton( "+" );
30       keys[ 14 ] = new JButton( "=" );
31       keys[ 15 ] = new JButton( "." );
32
33       // set keyPad layout to grid layout
34       keyPad = new JPanel();
35       keyPad.setLayout( new GridLayout( 4, 4 ) );
36
37       // add buttons to keyPad panel
38       // 7, 8, 9, divide
39       for ( int i = 7; i <= 10; i++ )
40          keyPad.add( keys[ i ] );
41
```

```
42          // 4, 5, 6
43          for ( int i = 4; i <= 6; i++ )
44             keyPad.add( keys[ i ] );
45
46          // multiply
47          keyPad.add( keys[ 11 ] );
48
49          // 1, 2, 3
50          for ( int i = 1; i <= 3; i++ )
51             keyPad.add( keys[ i ] );
52
53          // subtract
54          keyPad.add( keys[ 12 ] );
55
56          // 0
57          keyPad.add( keys[ 0 ] );
58
59          // ., =, add
60          for ( int i = 15; i >= 13; i-- )
61             keyPad.add( keys[ i ] );
62
63          // add components to (default) border layout
64          Container container = getContentPane();
65          container.add( lcd, BorderLayout.NORTH );
66          container.add( keyPad, BorderLayout.CENTER );
67
68          setSize( 200, 200 );
69          setVisible( true );
70
71       }  // end Calculator constructor
72
73       // execute application
74       public static void main( String args[] )
75       {
76          Calculator application = new Calculator();
77          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
78       }
79
80    }  // end class Calculator
```

13.10 Create the following GUI. You do not have to provide any functionality.

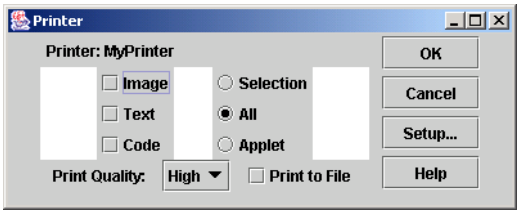


ANS:

```
1  // Exercise 13.10 Solution: ColorSelect.java
2  // This program creates a simple GUI
```

```
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class ColorSelect extends JFrame {
7     private JButton ok, cancel;
8     private JCheckBox background, foreground;
9     private JComboBox colorList;
10    private JPanel panel, panel2;
11
12    // create components and add them to applet
13    public ColorSelect()
14    {
15       super( "ColorSelect" );
16
17       getContentPane().setLayout( new BorderLayout() );
18
19       // north
20       colorList = new JComboBox();
21       colorList.addItem( "RED" );
22       getContentPane().add( colorList, BorderLayout.NORTH );
23
24       // center
25       panel = new JPanel();
26       background = new JCheckBox( "Background" );
27       foreground = new JCheckBox( "Foreground" );
28       panel.add( background );
29       panel.add( foreground );
30       getContentPane().add( panel, BorderLayout.CENTER );
31
32       // south
33       ok = new JButton( "Ok" );
34       cancel = new JButton( "Cancel" );
35       panel2 = new JPanel();
36       panel2.add( ok );
37       panel2.add( cancel );
38       getContentPane().add( panel2, BorderLayout.SOUTH );
39
40       setSize( 300, 125 );
41       setVisible( true );
42
43    } // end constructor
44
45    // execute application
46    public static void main ( String args[] )
47    {
48       ColorSelect app = new ColorSelect();
49       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
50    }
51
52 }  // end class ColorSelect
```

13.11 Create the following GUI. You do not have to provide any functionality.



ANS:

```
// Exercise 13.11 Solution: Printer.java
// This program creates a simple Printer GUI
import java.awt.*;
import javax.swing.*;

public class Printer extends JFrame {
   private JButton button1, button2, button3, button4;
   private JCheckBox check1, check2, check3, check4;
   private JRadioButton radio1, radio2, radio3;
   private ButtonGroup radioGroup;
   private JComboBox comboBox;
   private JLabel label1, label2;
   private JPanel panel1, panel2, panel3, panel4, panel5,
      panel6, panel7, panel8;

   // constructor sets up GUI
   public Printer()
   {
      super( "Printer" );

      // build left north panel
      label1 = new JLabel( "Printer: MyPrinter" );
      panel1 = new JPanel();
      panel1.setLayout( new FlowLayout( FlowLayout.LEFT ) );
      panel1.add( label1 );

      // build right east panel
      button1 = new JButton( "OK" );
      button2 = new JButton( "Cancel" );
      button3 = new JButton( "Setup..." );
      button4 = new JButton( "Help" );
      panel2 = new JPanel();
      panel2.setLayout( new GridLayout( 4, 1, 5, 5 ) );
      panel2.add( button1 );
      panel2.add( button2 );
      panel2.add( button3 );
      panel2.add( button4 );

      // build left south panel
      label2 = new JLabel( "Print Quality: " );
      comboBox = new JComboBox();
      comboBox.addItem( "High" );
```

```
43          check1 = new JCheckBox( "Print to File" );
44          panel3 = new JPanel();
45          panel3.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
46          panel3.add( label2 );
47          panel3.add( comboBox );
48          panel3.add( check1 );
49
50          // build left east panel
51          check2 = new JCheckBox( "Image" );
52          check3 = new JCheckBox( "Text" );
53          check4 = new JCheckBox( "Code" );
54          panel4 = new JPanel();
55          panel4.setLayout( new BorderLayout( ) );
56          panel4.add( check2, BorderLayout.NORTH );
57          panel4.add( check3, BorderLayout.CENTER );
58          panel4.add( check4, BorderLayout.SOUTH );
59
60          // build left west panel
61          panel5 = new JPanel();
62          panel5.setLayout( new BorderLayout() );
63          panel5.add( radio1 = new JRadioButton( "Selection",
64             false ), BorderLayout.NORTH );
65          panel5.add( radio2 = new JRadioButton( "All", true ),
66             BorderLayout.CENTER );
67          panel5.add( radio3 = new JRadioButton( "Applet", false ),
68             BorderLayout.SOUTH );
69
70          // group the radio buttons
71          radioGroup = new ButtonGroup();
72          radioGroup.add( radio1 );
73          radioGroup.add( radio2 );
74          radioGroup.add( radio3 );
75
76          // build left center
77          panel8 = new JPanel();
78          panel8.setLayout( new FlowLayout( FlowLayout.CENTER, 30, 0 ) );
79          panel8.setBackground( Color.white );
80          panel8.add( panel4 );
81          panel8.add( panel5 );
82
83          // setup left panel
84          panel6 = new JPanel();
85          panel6.setLayout( new BorderLayout() );
86          panel6.add( panel1, BorderLayout.NORTH );
87          panel6.add( panel8, BorderLayout.CENTER );
88          panel6.add( panel3, BorderLayout.SOUTH );
89
90          // setup applet layout
91          panel7 = new JPanel();
92          panel7.setLayout( new FlowLayout( FlowLayout.CENTER, 10, 0 ) );
93          panel7.add( panel6 );
94          panel7.add( panel2 );
95          getContentPane().add( panel7 );
96
```

```
97          setSize( 400, 160 );
98          setVisible( true );
99
100      } // end constructor
101
102      public static void main( String args[] )
103      {
104         Printer app = new Printer();
105         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
106      }
107
108   }  // end class Printer
```

13.12 Write a temperature conversion program that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via a `JTextField`). A `JLabel` should be used to display the converted temperature. Use the following formula for the conversion:

$$Celsius = \frac{5}{9} \times (Fahrenheit - 32)$$

ANS:

```
1  // Exercise 13.12 Solution: Convert.java
2  // Temperature conversion program
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Convert extends JFrame {
8     private JLabel label1, label2;
9     private JTextField temperatureF;
10
11    // constructer sets up GUI
12    public Convert()
13    {
14       super( "Temperature converter" );
15
16       label1 = new JLabel( "Enter Fahrenheit temperature:" );
17       temperatureF = new JTextField( 10 );
18
19       // register anonymous action listener
20       temperatureF.addActionListener(
21
22          new ActionListener() {
23
24             public void actionPerformed(ActionEvent e)
25             {
26                int temp = Integer.parseInt( temperatureF.getText() );
27                int celcius = ( int ) ( 5.0f / 9.0f * ( temp - 32 ) );
28                temperatureC.setText( String.valueOf( celcius ) );
29             }
30          }
31       );
```

```
32
33          label2 = new JLabel( "Temperature in Celcius is:" );
34
35          Container container = getContentPane();
36          container.setLayout( new BorderLayout() );
37          container.add( label1, BorderLayout.NORTH );
38          container.add( temperatureF, BorderLayout.CENTER );
39          container.add( label2, BorderLayout.SOUTH );
40
41          setSize( 225, 80 );
42          setVisible( true );
43
44       } // end Convert constructor
45
46       public static void main ( String args[] )
47       {
48          Convert app = new Convert();
49          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
50       }
51
52    }  // end class Convert
```

13.13 Enhance the temperature conversion program of Exercise 13.12 by adding the Kelvin temperature scale. The program should also allow the user to make conversions between any two scales. Use the following formula for the conversion between Kelvin and Celsius (in addition to the formula in Exercise 13.12):

Kelvin = *Celsius* + 273.15

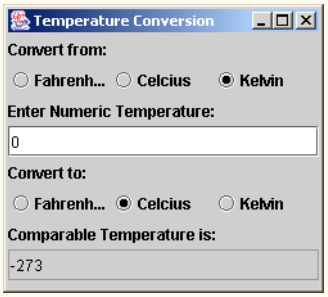
ANS:

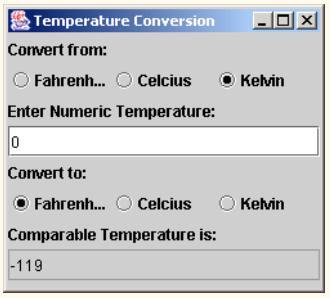
```
1  // Exercise 13.13 Solution: Convert.java
2  // Program converts temperatures.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Convert extends JFrame {
8     private JPanel convertFrom, convertTo;
9     private JLabel label1, label2, label3, label4;
10    private JTextField temperature1, temperature2;
11    private ButtonGroup radioFrom, radioTo;
12    private JRadioButton celciusBoxTo, fahrenheitBoxTo,
13       kelvinBoxTo, celciusBoxFrom, fahrenheitBoxFrom, kelvinBoxFrom;
14
15    // set up GUI
16    public Convert()
17    {
```

```
18         super( "Temperature Conversion" );
19
20         fahrenheitBoxFrom = new JRadioButton( "Fahrenheit", true );
21         celciusBoxFrom = new JRadioButton( "Celcius", false );
22         kelvinBoxFrom = new JRadioButton( "Kelvin",   false );
23         radioFrom = new ButtonGroup();
24         radioFrom.add( fahrenheitBoxFrom );
25         radioFrom.add( celciusBoxFrom );
26         radioFrom.add( kelvinBoxFrom );
27
28         fahrenheitBoxTo = new JRadioButton( "Fahrenheit", false );
29         celciusBoxTo = new JRadioButton( "Celcius", true );
30         kelvinBoxTo = new JRadioButton( "Kelvin", false );
31         radioTo = new ButtonGroup();
32         radioTo.add( fahrenheitBoxTo );
33         radioTo.add( celciusBoxTo );
34         radioTo.add( kelvinBoxTo );
35
36         convertFrom = new JPanel();
37         convertFrom.setLayout( new GridLayout( 1, 3 ) );
38         convertFrom.add( fahrenheitBoxFrom );
39         convertFrom.add( celciusBoxFrom );
40         convertFrom.add( kelvinBoxFrom );
41
42         convertTo = new JPanel();
43         convertTo.setLayout( new GridLayout( 1, 3 ) );
44         convertTo.add( fahrenheitBoxTo );
45         convertTo.add( celciusBoxTo );
46         convertTo.add( kelvinBoxTo );
47
48         label1 = new JLabel( "Convert from:" );
49         label2 = new JLabel( "Convert to:" );
50         label3 = new JLabel( "Enter Numeric Temperature: " );
51         label4 = new JLabel( "Comparable Temperature is: " );
52
53         temperature1 = new JTextField( 10 );
54         temperature1.addActionListener(
55
56            new ActionListener() {  // anonymous inner class
57
58               // perform conversions
59               public void actionPerformed( ActionEvent event )
60               {
61                  int convertTemp, temp;
62
63                  temp = Integer.parseInt( ( ( JTextField )
64                     event.getSource() ).getText() );
65
66                  // fahrenheit to celcius
67                  if ( fahrenheitBoxFrom.isSelected() &&
68                     celciusBoxTo.isSelected() ) {
69                     convertTemp = ( int ) ( 5.0f / 9.0f * ( temp - 32 ) );
70                     temperature2.setText( String.valueOf( convertTemp ) );
71                  }
```

```
72
73                // fahrenheit to kelvin
74                else if ( fahrenheitBoxFrom.isSelected() &&
75                   kelvinBoxTo.isSelected() ) {
76                   convertTemp = ( int )
77                      ( 5.0f / 9.0f * ( temp - 32 ) + 273 );
78                   temperature2.setText( String.valueOf( convertTemp ) );
79                }
80
81                // celcius to fahrenheit
82                else if ( celciusBoxFrom.isSelected() &&
83                   fahrenheitBoxTo.isSelected() ) {
84                   convertTemp = ( int ) ( 9.0f / 5.0f * temp + 32 );
85                   temperature2.setText( String.valueOf( convertTemp ) );
86                }
87
88                // celcius to kelvin
89                else if ( celciusBoxFrom.isSelected() &&
90                   kelvinBoxTo.isSelected() ) {
91                   convertTemp = temp + 273;
92                   temperature2.setText( String.valueOf( convertTemp ) );
93                }
94
95                // kelvin to celcius
96                else if ( kelvinBoxFrom.isSelected() &&
97                   celciusBoxTo.isSelected() ) {
98                   convertTemp = temp - 273;
99                   temperature2.setText( String.valueOf( convertTemp ) );
100               }
101
102               // kelvin to fahrenheit
103               else if ( kelvinBoxFrom.isSelected() &&
104                  fahrenheitBoxTo.isSelected() ) {
105                  convertTemp = ( int ) ( 5.0f /
106                     9.0f * ( temp - 273 ) + 32  );
107                  temperature2.setText( String.valueOf( convertTemp ) );
108               }
109
110            }  // end method actionPerformed
111
112         }  // end anonymous inner class
113
114      ); // end call to addActionListener
115
116      temperature2 = new JTextField( 10 );
117      temperature2.setEditable( false );
118
119      // add components to GUI
120      Container container = getContentPane();
121      container.setLayout( new GridLayout( 8, 1 ) );
122      container.add( label1 );
123      container.add( convertFrom );
124      container.add( label3 );
125      container.add( temperature1 );
```

```
126         container.add( label2 );
127         container.add( convertTo );
128         container.add( label4 );
129         container.add( temperature2 );
130
131         setSize( 250, 225 );
132         setVisible( true );
133
134      }  // end constructor
135
136      public static void main ( String args[] )
137      {
138         Convert application = new Convert();
139         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
140      }
141
142   }  // end class Convert
```





13.14 Write an application that allows the user to draw a rectangle by dragging the mouse on the application window. The upper left coordinate should be the location where the user presses the mouse button, and the lower right coordinate should be the location where the user releases the mouse button. Also display the area of the rectangle in a `JLabel` in the `SOUTH` region of a `BorderLayout`. Use the following formula for the area:

$$area = width \times height$$

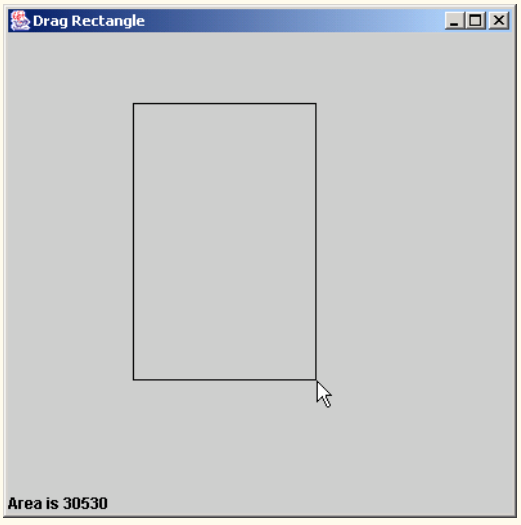
ANS:

```
1  // Exercise 13.14 Solution: DragRectangle.java
2  // Program draws a rectangle with the mouse
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DragRectangle extends JFrame {
8     private int topX, topY;
9     private int width, height;
10    private int bottomX, bottomY;
11    protected JLabel status;
12
```

```
13      // initialize values and set up the area label
14      public DragRectangle()
15      {
16         super( "Drag Rectangle" );
17         topX = 0;
18         topY = 0;
19         addMouseListener( new MouseHandler( this ) );
20
21         status = new JLabel();
22         getContentPane().add( status, BorderLayout.SOUTH );
23
24         setSize( 400, 400 );
25         setVisible( true );
26      }
27
28      // accessor methods for rectangle drawing
29      public int getTopX()
30      {
31         return topX;
32      }
33
34      public int getTopY()
35      {
36         return topY;
37      }
38
39      public int getWidth()
40      {
41         return width;
42      }
43
44      public int getHeight()
45      {
46         return height;
47      }
48
49      public int getBottomX()
50      {
51         return bottomX;
52      }
53
54      public int getBottomY()
55      {
56         return bottomY;
57      }
58
59      // mutator methods for rectangle drawing
60      public void setTopX( int x )
61      {
62         topX = x;
63      }
64
65      public void setTopY( int y )
66      {
```

```
67          topY = y;
68       }
69
70       public void setBottomX( int x )
71       {
72          bottomX = x;
73       }
74
75       public void setBottomY( int y )
76       {
77          bottomY = y;
78       }
79
80       public void setWidth( int w )
81       {
82          width = w;
83       }
84
85       public void setHeight( int h )
86       {
87          height = h;
88       }
89
90       // draw the rectangle
91       public void paint( Graphics g )
92       {
93          super.paint( g );
94
95          g.drawRect( topX, topY, width, height );
96       }
97
98       public static void main( String args[] )
99       {
100         DragRectangle app = new DragRectangle();
101         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
102      }
103
104   }  // end class DragRectangle
105
106   class MouseHandler extends MouseAdapter {
107      private DragRectangle draw;
108
109      // constructor
110      public MouseHandler( DragRectangle drag )
111      {
112         draw = drag;
113      }
114
115      // when mouse is release, draw new rectangle
116      public void mouseReleased( MouseEvent e )
117      {
118         // determine coordinates
119         draw.setBottomX( e.getX() );
120         draw.setBottomY( e.getY() );
```

```
121         draw.setWidth( Math.abs( draw.getTopX() - draw.getBottomX() ) );
122         draw.setHeight( Math.abs( draw.getTopY() - draw.getBottomY() ) );
123         draw.setTopX( Math.min( draw.getTopX(), draw.getBottomX() ) );
124         draw.setTopY( Math.min( draw.getTopY(), draw.getBottomY() ) );
125
126         // set label
127         draw.status.setText( "Area is " + ( draw.getWidth() *
128            draw.getHeight() ) );
129
130         // draw the new rectangle
131         draw.repaint();
132      }
133
134      // determine initial coordinates
135      public void mousePressed( MouseEvent e )
136      {
137         draw.setTopX( e.getX() );
138         draw.setTopY( e.getY() );
139      }
140
141   }  // end class MouseHandler
```



13.15 Modify the program of Exercise 13.14 to draw different shapes. The user should be allowed to choose from an oval, an arc, a line and a rectangle with rounded corners. Also display the mouse coordinates in the status bar.

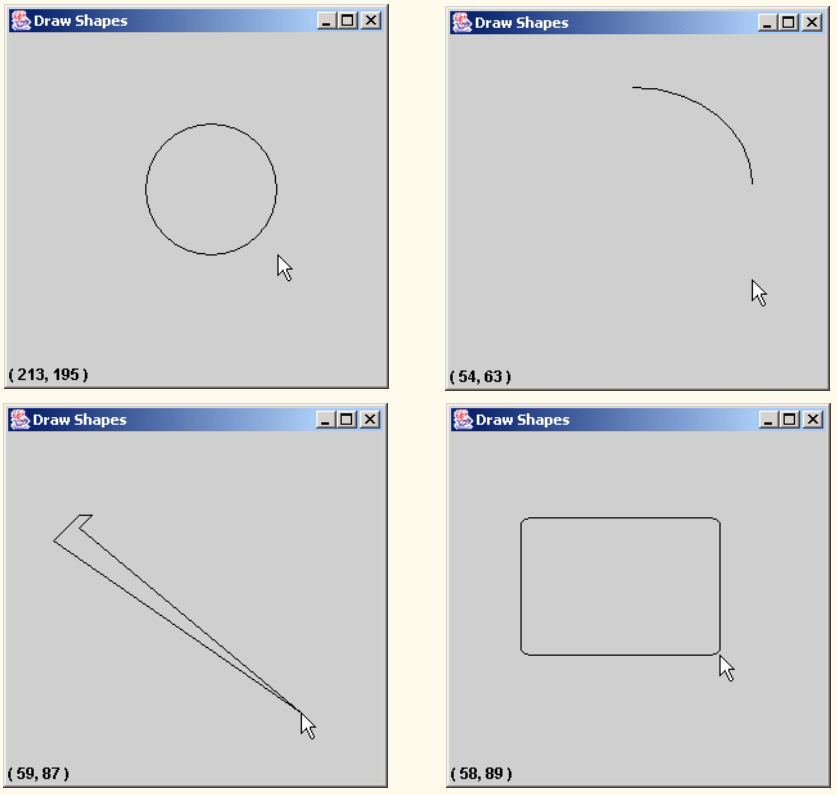
ANS:

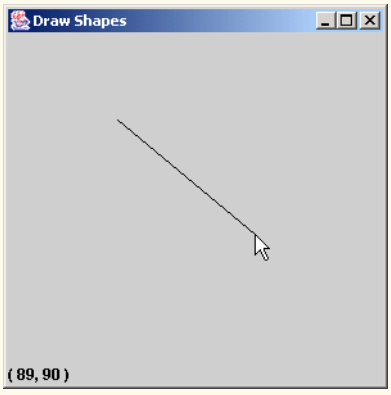
```
1  // Exercise 13.15 Solution: DrawShape.java
2  // Program draws shapes with the mouse
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
```

```
 6  import javax.swing.event.*;
 7
 8  public class DrawShape extends JFrame {
 9     private int topX, topY, width, height, shape, bottomX, bottomY;
10     private JLabel status;
11     private final int LINE = 1, OVAL = 2, ARC = 3, ROUND = 4, POLY = 5;
12     private String shapeNames[] =
13        { "Oval", "Arc", "Line", "Round Rect", "Polygon" };
14
15     // constructor
16     public DrawShape()
17     {
18        super( "Draw Shapes" );
19        topX = 0;
20        topY = 0;
21
22        status = new JLabel();
23
24        addMouseListener( new MouseHandler() );
25        addMouseMotionListener( new MouseMotionHandler() );
26        addKeyListener( new KeyHandler() );
27
28        // add label to bottom of window via call to default layout
29        Container container = getContentPane();
30        container.add( status, BorderLayout.SOUTH );
31
32        setSize( 300, 300 );
33        setVisible( true );
34        JOptionPane.showMessageDialog( null,
35           "Press a key to choose the shape to draw\n" +
36           "1 - line\n2 - oval\n3 - arc\n4 - rectangle\n5 - polygon\n" );
37
38     }  // end constructor
39
40     // draw shape
41     public void paint( Graphics g )
42     {
43        super.paint( g );
44
45        if ( shape != LINE ) {
46           topX = Math.min( topX, bottomX );
47           topY = Math.min( topY, bottomY );
48        }
49
50        switch ( shape ) {
51
52           case LINE:
53              g.drawLine( topX, topY, bottomX, bottomY );
54              break;
55
56           case OVAL:
57              g.drawOval( topX, topY, width, height );
58              break;
59
```

```
60             case ARC:
61                g.drawArc( topX, topY, width, height, 0, 90 );
62                break;
63
64             case ROUND:
65                g.drawRoundRect( topX, topY, width, height, 20, 10 );
66                break;
67
68             case POLY:
69                int xValues[] =
70                   { topX + 10, topX, bottomX, topX - 20, topX - 10, topX };
71                int yValues[] =
72                   { topY, topY + 10, bottomY, topY + 20, topY + 10, topY };
73                g.drawPolygon( xValues, yValues, 6 );
74                break;
75          }
76
77       }  // end method paint
78
79       public static void main( String args[] )
80       {
81          DrawShape application = new DrawShape();
82          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
83       }
84
85       // inner class to handle mouse motion events
86       private class MouseMotionHandler extends MouseMotionAdapter {
87
88          public void mouseMoved( MouseEvent event )
89          {
90             status.setText(
91                "( " + event.getX() + ", " + event.getY() + " )" );
92          }
93       }
94
95       // inner class to handle mouse events
96       private class MouseHandler extends MouseAdapter {
97
98          public void mousePressed( MouseEvent event )
99          {
100            topX = event.getX();
101            topY = event.getY();
102         }
103
104         public void mouseReleased( MouseEvent event )
105         {
106            bottomX = event.getX();
107            bottomY = event.getY();
108            width = Math.abs( topX - bottomX );
109            height = Math.abs( topY - bottomY );
110
111            repaint();
112         }
113
114      }  // end private inner class MouseHandler
```

```
115
116    // inner class to handle key events
117    private class KeyHandler extends KeyAdapter {
118
119       public void keyPressed( KeyEvent event )
120       {
121          shape = Integer.parseInt( String.valueOf( event.getKeyChar() ) );
122       }
123
124    }  // end inner class KeyHandler
125
126 }  // end class DrawShape
```





13.16 Write a program that will allow the user to draw a shape with the mouse. The shape to draw should be determined by a `KeyEvent`, using the following keys: *c* draws a circle, *o* draws an oval, *r* draws a rectangle and *l* draws a line. The size and placement of the shape should be determined by the `mousePressed` and `mouseReleased` events. Display the name of the current shape in a `JLabel` in the `SOUTH` region of a `BorderLayout`. The initial shape should default to a circle.

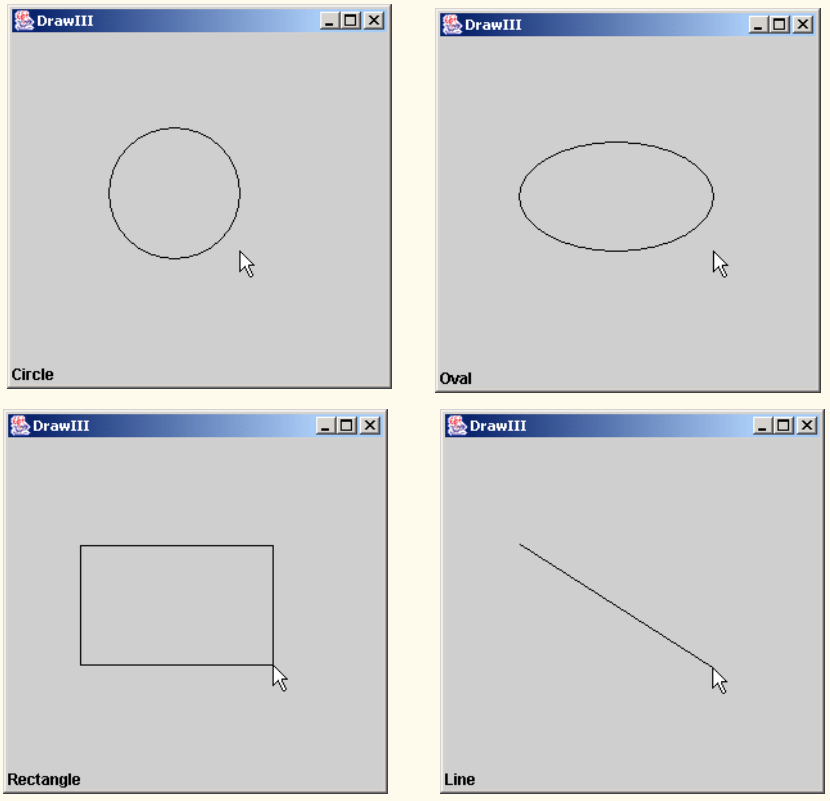
ANS:

```
1  // Exercise 13.16 Solution: DrawIII.java
2  // Program draws different shapes with the mouse
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class DrawIII extends JFrame {
8     private int topX, topY;
9     private int width, height, shape;
10    private int bottomX, bottomY;
11    protected JLabel status;
12    public static final int CIRCLE = 0, OVAL = 1, LINE = 2, RECT = 3;
13
14    // constructor initializes values
15    public DrawIII()
16    {
17       super( "DrawIII" );
18       topX = 0;
19       topY = 0;
20
21       status = new JLabel();
22       status.setText( "Circle" );
23       getContentPane().add( status, BorderLayout.SOUTH );
24
25       addMouseListener( new MouseHandler() );
26       addKeyListener( new KeyHandler() );
27
28       setSize( 300, 300 );
29       setVisible( true );
30    }
```

```
31
32    // accessor methods
33    public int getTopX()
34    {
35       return topX;
36    }
37
38    public int getTopY()
39    {
40       return topY;
41    }
42
43    public int getBottomX()
44    {
45       return bottomX;
46    }
47
48    public int getBottomY()
49    {
50       return bottomY;
51    }
52
53    // mutator methods
54    public void setTopX( int x )
55    {
56       topX = x;
57    }
58
59    public void setTopY( int y )
60    {
61       topY = y;
62    }
63
64    public void setBottomX( int x )
65    {
66       bottomX = x;
67    }
68
69    public void setBottomY( int y )
70    {
71       bottomY = y;
72    }
73
74    public void setWidth( int w )
75    {
76       width = w;
77    }
78
79    public void setHeight( int h )
80    {
81       height = h;
82    }
83
```

```
84       // draw the chosen shape
85       public void paint( Graphics g )
86       {
87          super.paint( g );
88
89          if ( shape != LINE ) {
90             topX = Math.min( topX, bottomX );
91             topY = Math.min( topY, bottomY );
92          }
93
94          switch ( shape ) {
95
96             case CIRCLE:
97                g.drawOval( topX, topY, width, width );
98                break;
99
100            case OVAL:
101               g.drawOval( topX, topY, width, height );
102               break;
103
104            case LINE:
105               g.drawLine( topX, topY, bottomX, bottomY );
106               break;
107
108            case RECT:
109               g.drawRect( topX, topY, width, height );
110               break;
111         }
112      }
113
114      // indicate which object should be drawn
115      public void setShape( int preference )
116      {
117         shape = preference;
118      }
119
120      public static void main( String args[] )
121      {
122         DrawIII app = new DrawIII();
123         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
124      }
125
126      // class tells DrawIII where to draw the shape
127      private class MouseHandler extends MouseAdapter {
128
129         // get shape's final values
130         public void mouseReleased( MouseEvent e )
131         {
132            setBottomX( e.getX() );
133            setBottomY( e.getY() );
134            setWidth( Math.abs( getTopX() - getBottomX() ) );
135            setHeight( Math.abs( getTopY() - getBottomY() ) );
136            repaint();
137         }
```

```
138
139       // get initial location values
140       public void mousePressed( MouseEvent e )
141       {
142          setTopX( e.getX() );
143          setTopY( e.getY() );
144       }
145
146    }  // end inner class MouseHandler
147
148    // class determines which shape to draw
149    private class KeyHandler extends KeyAdapter {
150
151       public void keyTyped( KeyEvent e )
152       {
153          int shape = DrawIII.CIRCLE;
154
155          switch ( e.getKeyChar() ) {
156
157             case 'c': case 'C':
158                shape = DrawIII.CIRCLE;
159                status.setText( "Circle" );
160                break;
161
162             case 'o': case 'O':
163                shape = DrawIII.OVAL;
164                status.setText( "Oval" );
165                break;
166
167             case 'r': case 'R':
168                shape = DrawIII.RECT;
169                status.setText( "Rectangle" );
170                break;
171
172             case 'l': case 'L':
173                shape = DrawIII.LINE;
174                status.setText( "Line" );
175                break;
176
177             default:
178                status.setText( "Circle" );
179                break;
180          }
181
182          setShape( shape );
183       }
184
185    }  // end inner class KeyHandler
186
187 }  // end class DrawIII
```



13.17 Create an application that enables the user to paint a picture. The user should be able to choose the shape to draw, the color in which the shape should appear and whether the shape should be filled with color. Use the graphical user interface components we discussed in this chapter, such as `JComboBox`es, `JRadioButton`s and `JCheckBox`es, to allow the user to select various options. The program should provide a `JButton` object that allows the user to erase the contents of the window.

ANS:

```
1  // Exercise 13.17 Solution: Painter.java
2  // Paint program with different shapes, colors, fills
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Painter extends JFrame {
9     private int topX, topY, height, width, bottomY, bottomX, shape;
10    private boolean clear, filled;
11    private Color drawingColor;
12
13    private JPanel panel1, panel2, panel3, panel4;
14    private JRadioButton ovalBox, rectBox, lineBox;
15    private ButtonGroup shapeGroup;
```

```
16    private JCheckBox fillBox;
17    private JList colorList;
18    private JButton clearButton;
19
20    private String colorNames[] = { "Black", "Green", "Blue",
21       "Red", "Cyan", "Yellow" };
22    private Color colors[] = { Color.black, Color.green,
23       Color.blue, Color.red, Color.cyan, Color.yellow };
24    private final int OVAL = 1, LINE = 2, RECT = 3;
25
26    // constructor
27    public Painter()
28    {
29       super( "Painter" );
30
31       // set default to black oval
32       setShape( OVAL );
33       setDrawingColor( Color.black );
34
35       // set this to make the window white
36       clear = true;
37
38       addMouseListener( new MouseHandler() );
39
40       // sets up GUI
41       createToolWindow();
42
43       Container container = getContentPane();
44       container.add( panel4, BorderLayout.SOUTH );
45
46       setSize( 500, 500 );
47       setVisible( true );
48    }
49
50    // accessor methods
51    public int getTopX()
52    {
53       return topX;
54    }
55
56    public int getTopY()
57    {
58       return topY;
59    }
60
61    public int getWidth()
62    {
63       return width;
64    }
65
66    public int getHeight()
67    {
68       return height;
69    }
```

```
70
71    public int getBottomX()
72    {
73       return bottomX;
74    }
75
76    public int getBottomY()
77    {
78       return bottomY;
79    }
80
81    public int getShape()
82    {
83       return shape;
84    }
85
86    public Color getDrawingColor()
87    {
88       return drawingColor;
89    }
90
91    public boolean getFill()
92    {
93       return filled;
94    }
95
96    // mutator methods
97    public void setTopX( int x )
98    {
99       topX = x;
100   }
101
102   public void setTopY( int y )
103   {
104      topY = y;
105   }
106
107   public void setBottomX( int x )
108   {
109      bottomX = x;
110   }
111
112   public void setBottomY( int y )
113   {
114      bottomY = y;
115   }
116
117   public void setWidth( int wide )
118   {
119      width = wide;
120   }
121
122   public void setHeight( int high )
123   {
```

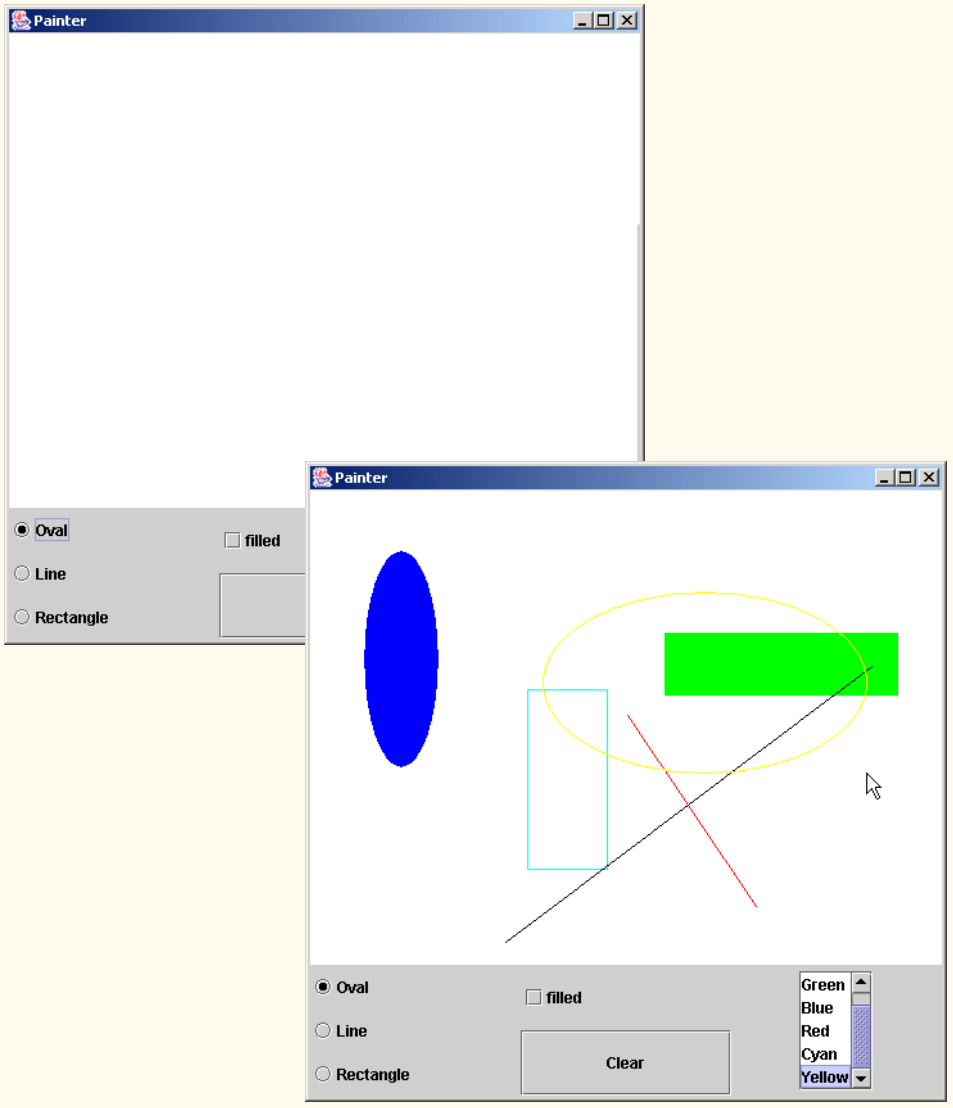
```
124         height = high;
125      }
126
127      public void setShape( int preference )
128      {
129         if ( preference >= OVAL && preference <= RECT )
130            shape = preference;
131      }
132
133      public void setDrawingColor( Color color )
134      {
135         for ( int i = 0; i < colors.length; i++ )
136
137            if ( color == colors[ i ] ) {
138               drawingColor = color;
139               return;
140            }
141      }
142
143      public void setClear()
144      {
145         clear = true;
146      }
147
148      public void setFill()
149      {
150         filled = true;
151      }
152
153      public void clearFill()
154      {
155         filled = false;
156      }
157
158      // draw new shape with specified color, fill, position
159      public void paint( Graphics g )
160      {
161         g.setColor( drawingColor );
162         width = Math.abs( topX - bottomX );
163         height = Math.abs( topY - bottomY );
164
165         if ( shape != LINE ) {
166            topX = Math.min( topX, bottomX );
167            topY = Math.min( topY, bottomY );
168
169            // draw filled shapes
170            if ( filled && shape != LINE )
171               switch ( shape ) {
172
173                  case OVAL:
174                     g.fillOval( topX, topY,  width, height );
175                     break;
176
```

```
177                case RECT:
178                   g.fillRect( topX, topY, width, height );
179                   break;
180             }
181
182          // draw unfilled shapes
183          else
184             switch ( shape ) {
185
186                case OVAL:
187                   g.drawOval( topX, topY,  width, height );
188                   break;
189
190                case RECT:
191                   g.drawRect( topX, topY, width, height );
192                   break;
193             }
194       }
195
196       // draw line
197       else if ( shape == LINE )
198          g.drawLine( topX, topY, bottomX, bottomY );
199
200       // erase entire canvase
201       if ( clear == true ) {
202          g.setColor( Color.white );
203          g.fillRect( 0, 0, 500, 500 );
204          clear = false;
205       }
206
207       // ensure that the control panel is still visible
208       panel4.repaint();
209
210    }  // end method paint
211
212    // set up the buttons and controls
213    public void createToolWindow()
214    {
215       // set up to handle clear button
216       clearButton = new JButton( "Clear" );
217       clearButton.addActionListener( new ClearButtonHandler() );
218
219       // set up to choose shapes
220       ovalBox = new JRadioButton( "Oval", true );
221       lineBox = new JRadioButton( "Line", false );
222       rectBox = new JRadioButton( "Rectangle", false );
223       RadioButtonHandler handler = new RadioButtonHandler();
224       ovalBox.addItemListener( handler );
225       lineBox.addItemListener( handler );
226       rectBox.addItemListener( handler );
227
228       // group the shapes
229       shapeGroup = new ButtonGroup();
230       shapeGroup.add( ovalBox );
```

```
231         shapeGroup.add( lineBox );
232         shapeGroup.add( rectBox );
233
234         // set up to choose if filled
235         fillBox  = new JCheckBox( "filled" );
236         FillBoxHandler fillHandler = new FillBoxHandler();
237         fillBox.addItemListener( fillHandler );
238
239         // set up to choose color
240         colorList = new JList( colorNames );
241         colorList.setVisibleRowCount( 5 );
242         colorList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
243         colorList.setSelectedValue( "Black", true );
244
245         // set up event handler
246         colorList.addListSelectionListener(
247
248            new ListSelectionListener() {
249
250               public void valueChanged( ListSelectionEvent e )
251               {
252                  drawingColor = colors[ colorList.getSelectedIndex() ];
253               }
254            }
255         );
256
257         // create panels and layout
258         panel1 = new JPanel();
259         panel2 = new JPanel();
260         panel3 = new JPanel();
261         panel4 = new JPanel();
262
263         panel1.setLayout( new GridLayout( 3, 1 ) );
264         panel2.setLayout( new GridLayout( 2, 1 ) );
265         panel4.setLayout( new GridLayout( 1, 3 ) );
266
267         panel1.add( ovalBox );
268         panel1.add( lineBox );
269         panel1.add( rectBox );
270         panel2.add( fillBox );
271         panel2.add( clearButton );
272         panel3.add( new JScrollPane( colorList ) );
273
274         panel4.add( panel1 );
275         panel4.add( panel2 );
276         panel4.add( panel3 );
277      }
278
279      // class determines location and size of shapes
280      private class MouseHandler extends MouseAdapter {
281
282         // initial coordinates
283         public void mousePressed( MouseEvent e )
284         {
```

```
285          setTopX( e.getX() );
286          setTopY( e.getY() );
287       }
288
289       // end coordinates
290       public void mouseReleased( MouseEvent e )
291       {
292          setBottomX( e.getX() );
293          setBottomY( e.getY() );
294          repaint();
295       }
296
297    }  // end inner class MouseHandler
298
299    // class responds to desire to erase canvas
300    private class ClearButtonHandler implements ActionListener {
301
302       public void actionPerformed( ActionEvent e )
303       {
304          setClear();
305          repaint();
306       }
307
308    }  // end inner class ClearButtonHandler
309
310    // class determines which shape to draw
311    private class RadioButtonHandler implements ItemListener {
312
313       public void itemStateChanged( ItemEvent e )
314       {
315          if ( e.getSource() == ovalBox )
316             setShape( OVAL );
317
318          else if ( e.getSource() == lineBox )
319             setShape( LINE );
320
321          else if (  e.getSource() == rectBox )
322             setShape( RECT );
323       }
324
325    }  // end inner class RadioButtonHandler
326
327    // class determines whether shape is to be filled
328    private class FillBoxHandler implements ItemListener {
329
330       public void itemStateChanged( ItemEvent e )
331       {
332          if ( e.getSource() == fillBox )
333             if ( e.getStateChange() == ItemEvent.SELECTED )
334                setFill();
335
336             else
337                clearFill();
338       }
```

```
339
340     }  // end inner class FillBoxHandler
341
342     public static void main( String args[] )
343     {
344        Painter app = new Painter();
345        app.setDefaultCloseOperation( EXIT_ON_CLOSE );
346     }
347
348  }  // end class Painter
```



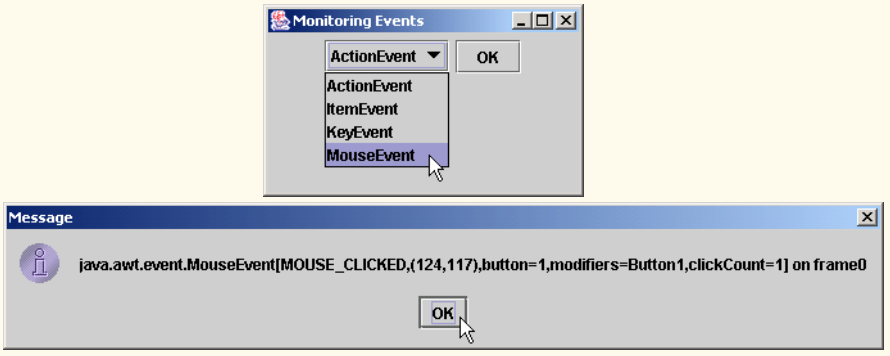
13.18 Write a program that uses `System.out.println` statements to print out events as they occur. Provide a `JComboBox` with a minimum of four items. The user should be able to choose an event to monitor from the `JComboBox`. When that particular event occurs, display information about the event in a message dialog box. Use method `toString` on the event object to convert it to a string representation.

ANS:

```
1  // Exercises 13.18 Solution: EventMonitor.java
2  // Program monitors events occurred.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class EventMonitor extends JFrame {
9     private JComboBox eventsComboBox;
10    private JButton okButton;
11    private String eventSelected = "ActionEvent";
12    private String events[] =
13       { "ActionEvent", "ItemEvent", "KeyEvent", "MouseEvent" };
14
15    // set up GUI
16    public EventMonitor()
17    {
18       super( "Monitoring Events" );
19
20       // get content pane and set its layout
21       Container container = getContentPane();
22       container.setLayout( new FlowLayout() );
23
24       // set up JComboBox and register its event handler
25       eventsComboBox = new JComboBox( events );
26       eventsComboBox.setMaximumRowCount( events.length );
27       eventsComboBox.addItemListener( new ComboBoxHandler() );
28       eventsComboBox.addKeyListener( new KeyHandler() );
29
30       // set up OK button and register its event handler
31       okButton = new JButton( "OK" );
32       okButton.addActionListener( new ActionHandler() );
33       okButton.addKeyListener( new KeyHandler() );
34
35       // register mouse, key and window event handlers
36       addMouseListener( new MouseHandler() );
37
38       container.add( eventsComboBox );
39       container.add( okButton );
40       setSize( 250, 150);
41       setVisible( true );
42
43    } // end EventMonitor constructor
44
45    // class handles combo box event
46    private class ComboBoxHandler implements ItemListener {
47
```

```
48          public void itemStateChanged( ItemEvent event )
49          {
50             if ( event.getStateChange() == ItemEvent.SELECTED ) {
51                eventSelected = events[ eventsComboBox.getSelectedIndex() ];
52                System.out.println( "ItemEvent occurred: " +
53                   "Item selected " + eventSelected + "." );
54
55                if ( eventSelected.equals( "ItemEvent" ) )
56                   JOptionPane.showMessageDialog( null, event.toString() );
57             }
58          }
59
60       } // end inner class ComboBoxHandler
61
62       // class handles mouse event
63       private class MouseHandler extends MouseAdapter {
64
65          public void mouseClicked( MouseEvent event )
66          {
67             System.out.println( "MouseEvent occurred: Clicked at [" +
68                event.getX() + ", " + event.getY() + "]." );
69
70             if ( eventSelected.equals( "MouseEvent" ) )
71                JOptionPane.showMessageDialog( null, event.toString() );
72          }
73
74       } // end inner class MouseHandler
75
76       // class handles action event
77       private class ActionHandler implements ActionListener {
78
79          public void actionPerformed( ActionEvent event )
80          {
81             System.out.println(
82                "ActionEvent occurred: Clicked OK button." );
83
84             if ( eventSelected.equals( "ActionEvent" ) )
85                JOptionPane.showMessageDialog( null, event.toString() );
86          }
87
88       } // end inner class ActionHandler
89
90       // class handles key event
91       private class KeyHandler extends KeyAdapter {
92
93          public void keyTyped( KeyEvent event )
94          {
95             System.out.println( "KeyEvent occurred: Typed " +
96                event.getKeyChar() );
97
98             if ( eventSelected.equals( "KeyEvent" ) )
99                JOptionPane.showMessageDialog( null, event.toString() );
100         }
101
102      } // end inner class KeyHandler
```

```
103
104    public static void main( String[] args )
105    {
106       EventMonitor application = new EventMonitor();
107       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
108    }
109
110 } // end class EventMonitor
```

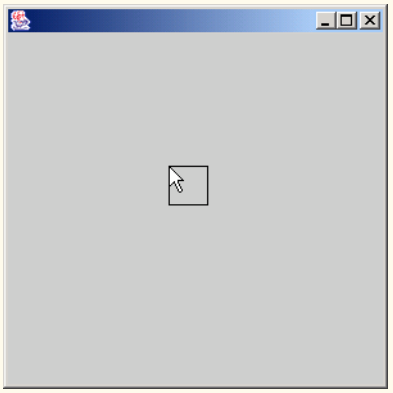


13.19 Write a program that draws a square. As the mouse moves over the drawing area, repaint the square, with the upper left corner of the square following the exact path of the mouse cursor.

ANS:

```
1  // Exercise 13.19 Solution: Follow.java
2  // Program draws a rectangle that follows the mouse pointer.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Follow extends JFrame {
9     private int xMouse, yMouse;
10
11    // constructor creates mouse listener
12    public Follow()
13    {
14       addMouseMotionListener(
15
16          new MouseMotionAdapter() {
17
18             public void mouseMoved( MouseEvent e )
19             {
20                xMouse = e.getX();
21                yMouse = e.getY();
22                repaint();
23             }
24          }
25       );
```

```
26
27          setSize( 300, 300 );
28          setVisible( true );
29       }
30
31       // draws the rectangle
32       public void paint( Graphics g )
33       {
34          super.paint( g );
35          g.drawRect( xMouse, yMouse, 30, 30 );
36       }
37
38       public static void main( String args[] )
39       {
40          Follow app = new Follow();
41          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
42       }
43    }
```



13.20 Modify the program of Fig. 13.19 to incorporate colors. Allow the user to select the color with `JRadioButton` objects that represent the following six colors: red, black, magenta, blue, green and yellow. When a new color is selected, drawing should occur in the new color.

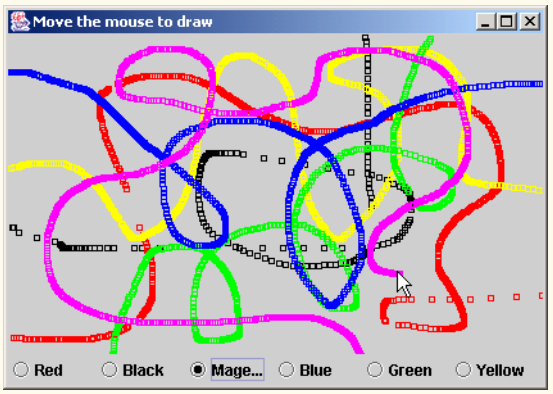
ANS:]

```
1  // Exercise 13.20 Solution: Drag.java
2  // Program allows the user to specify the color of a dragged object.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Drag extends JFrame {
9     private int xValue = -10, yValue = -10;
10    private Color color;
11    private String colorNames[] = { "Red", "Black",
12       "Magenta", "Blue", "Green", "Yellow" };
13    private Color colors[] = { Color.red, Color.black,
```

```
14          Color.magenta, Color.blue, Color.green, Color.yellow };
15       private JPanel colorPanel;
16       private JRadioButton buttons[];
17       private ButtonGroup radio;
18
19       // constructor sets up GUI
20       public Drag()
21       {
22          super( "Move the mouse to draw" );
23
24          // create buttons
25          buttons = new JRadioButton[ 6 ];
26          ButtonHandler bHandle = new ButtonHandler();
27          radio = new ButtonGroup();
28          colorPanel = new JPanel();
29          colorPanel.setLayout( new GridLayout( 1, 6, 0, 0 ) );
30
31          // initialize buttons
32          for ( int j = 0; j < buttons.length; j++ ) {
33             buttons[ j ] = new JRadioButton( colorNames[ j ] );
34             colorPanel.add( buttons[ j ] );
35             buttons[ j ].addActionListener( bHandle );
36             radio.add( buttons[ j ] );
37          }
38
39          // set initial color
40          buttons[ 1 ].setSelected( true );
41          color = colors[ 1 ];
42          addMouseMotionListener( new MotionHandler( ) );
43
44          Container container = getContentPane();
45          container.add( colorPanel, BorderLayout.SOUTH );
46
47          setSize( 425, 300 );
48          setVisible( true );
49       }
50
51       // draw rectangle
52       public void paint( Graphics g )
53       {
54          g.setColor( color );
55          g.drawRect( xValue, yValue, 4, 4 );
56
57          // make sure the bottom panel is visible
58          colorPanel.repaint();
59       }
60
61       // set the current drawing color
62       public void setColor( Color choice )
63       {
64          color = choice;
65       }
66
```

```
67     // set the coordinates of the rectangle
68     public void setCoordinates( int x, int y )
69     {
70        xValue = x;
71        yValue = y;
72        repaint();
73     }
74
75     public static void main( String args[] )
76     {
77        Drag app = new Drag();
78        app.setDefaultCloseOperation( EXIT_ON_CLOSE );
79     }
80
81     // inner class determines where the rectangle is drawn
82     private class MotionHandler implements MouseMotionListener {
83
84        public void mouseDragged( MouseEvent e )
85        {
86           setCoordinates( e.getX(), e.getY() );
87        }
88
89        public void mouseMoved( MouseEvent e )
90        {
91           setCoordinates( e.getX(), e.getY() );
92        }
93
94     }  // end inner class MotionHandler
95
96     // inner class determines the color with which to draw
97     private class ButtonHandler implements ActionListener {
98
99        public void actionPerformed( ActionEvent e )
100       {
101          if ( e.getActionCommand().equals( "Red" ) )
102             setColor( Color.red );
103
104          else if ( e.getActionCommand().equals( "Green" ) )
105             setColor( Color.green );
106
107          else if ( e.getActionCommand().equals( "Magenta" ) )
108             setColor( Color.magenta );
109
110          else if ( e.getActionCommand().equals( "Blue" ) )
111             setColor( Color.blue );
112
113          else if ( e.getActionCommand().equals( "Black" ) )
114             setColor( Color.black );
115
116          else if ( e.getActionCommand().equals( "Yellow" ) )
117             setColor( Color.yellow );
118       }
119
120    }  // end inner class ButtonHandler
```

```
121
122 } // end class Drag
```



13.21 Write a program that plays "guess the number" as follows: Your program chooses the number to be guessed by selecting an integer at random in the range 1–1000. The program then displays the following in a label:

```
I have a number between 1 and 1000 can you guess my number?
Please enter your first guess.
```

A `JTextField` should be used to input the guess. As each guess is input, the background color should change to either red or blue. Red indicates that the user is getting "warmer," and blue indicates that the user is getting "colder." A `JLabel` should display either "`Too High`" or "`Too Low`" to help the user zero in on the correct answer. When the user gets the correct answer, "`Correct!`" should be displayed, and the `JTextField` used for input should be changed to be uneditable. A `JButton` should be provided to allow the user to play the game again. When the `JButton` is clicked, a new random number should be generated and the input `JTextField` changed to be editable.

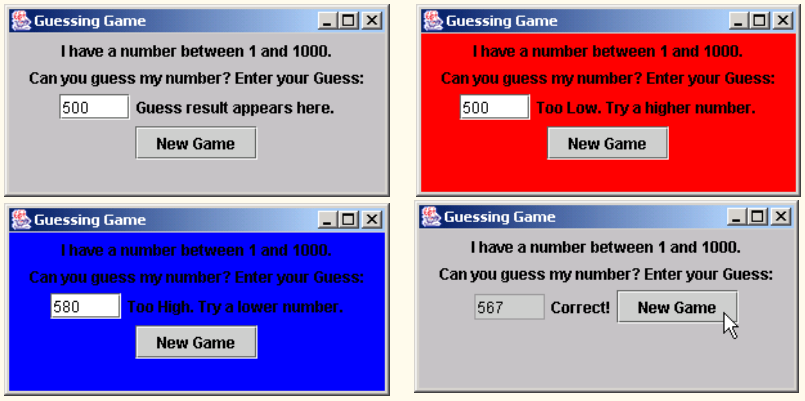
ANS:

```
1  // Exercise 13.21 Solution: GuessGame.java
2  // Guess the number
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class GuessGame extends JFrame {
8     private int number, guessCount;
9     private int lastDistance;
10    private JTextField guessInput;
11    private JLabel prompt1, prompt2, message;
12    private JButton newGame;
13    private Color background;
14    Container container;
15
16    // set up GUI and initialize values
17    public GuessGame()
18    {
```

```
19         super( "Guessing Game" );
20
21         guessCount = 0;
22         background = Color.lightGray;
23
24         // create GUI components
25         prompt1 = new JLabel( "I have a number between 1 and 1000." );
26      prompt2 = new JLabel(
27            "Can you guess my number? Enter your Guess:" );
28
29         guessInput = new JTextField( 5 );
30         guessInput.addActionListener( new GuessHandler( ) );
31         message = new JLabel( "Guess result appears here." );
32
33         // button starts a new game
34         newGame = new JButton( "New Game" );
35         newGame.addActionListener(
36
37            new ActionListener() {
38
39               public void actionPerformed( ActionEvent e )
40               {
41                  message.setText( "Guess Result" );
42                  guessInput.setText( "" );
43                  guessInput.setEditable( true );
44                  background = Color.lightGray;
45                  theGame();
46                  repaint();
47               }
48            }
49         );
50
51         // add components to JFrame
52         container = getContentPane();
53         container.setLayout( new FlowLayout() );
54         container.add( prompt1 );
55         container.add( prompt2 );
56         container.add( guessInput );
57         container.add( message );
58         container.add( newGame );
59
60         setSize( 300, 150 );
61         setVisible( true );
62
63         theGame();
64      }
65
66      // choose a new random number
67      public void theGame()
68      {
69         number = ( int ) ( Math.random() * 1000 + 1 );
70      }
71
```

```
72     // change background color
73     public void paint( Graphics g )
74     {
75        super.paint( g );
76        container.setBackground( background );
77     }
78
79     // react to new guess
80     public void react( int guess ) {
81        guessCount++;
82        int currentDistance = 1000;
83
84        // first guess
85        if ( guessCount == 1 ) {
86           lastDistance = Math.abs( guess - number );
87
88           if ( guess > number )
89              message.setText( "Too High. Try a lower number." );
90           else
91              message.setText( "Too Low. Try a higher number." );
92        }
93        else {
94           currentDistance = Math.abs( guess - number );
95
96          // guess is too high
97           if ( guess > number ) {
98              message.setText( "Too High. Try a lower number." );
99
100             background = ( currentDistance <= lastDistance ) ?
101               Color.red : Color.blue;
102
103             lastDistance = currentDistance;
104          }
105
106         // guess is too low
107         else if ( guess < number ) {
108             message.setText( "Too Low. Try a higher number." );
109
110             background = ( currentDistance <= lastDistance ) ?
111               Color.red : Color.blue;
112
113             lastDistance = currentDistance;
114          }
115
116          // guess is correct
117          else {
118             message.setText( "Correct!" );
119             background = Color.lightGray;
120             guessInput.setEditable( false );
121             guessCount = 0;
122          }
123
124          repaint();
125       }
```

```
126
127    } // end method react
128
129    public static void main( String args[] )
130    {
131       GuessGame app = new GuessGame();
132       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
133    }
134
135    // inner class acts on user input
136    class GuessHandler implements ActionListener
137    {
138       public void actionPerformed( ActionEvent e )
139       {
140          int guess = Integer.parseInt( guessInput.getText() );
141          react( guess );
142       }
143
144    }  // end inner class GuessHandler
145
146 }  // end class GuessGame
```



13.22 It is often useful to display the events that occur during the execution of a program. This can help you understand when the events occur and how they are generated. Write a program that enables the user to generate and process every event discussed in this chapter. The program should provide methods from the `ActionListener`, `ItemListener`, `ListSelectionListener`, `MouseListener`, `MouseMotionListener` and `KeyListener` interfaces to display messages when the events occur. Use method `toString` to convert the event objects received in each event handler into a `String` that can be displayed. Method `toString` creates a `String` containing all the information in the event object.

ANS:

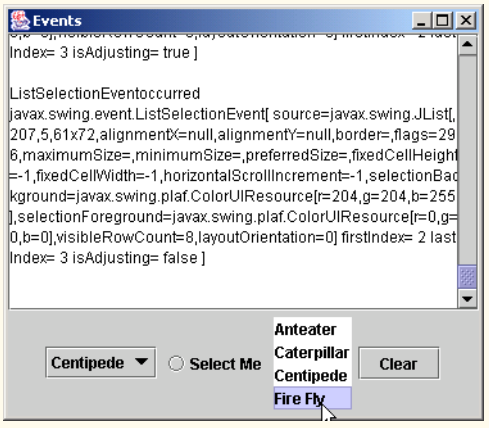
```
1  // Exercise 13.22 Solution: Events.java
2  // Program displays events that occur during execution.
3  import java.awt.*;
4  import java.awt.event.*;
```

```
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Events extends JFrame implements ActionListener,
9     ItemListener, MouseListener, MouseMotionListener,
10    KeyListener, ListSelectionListener {
11    private JPanel panel1;
12    private JScrollPane scrollPane;
13    private JTextArea output;
14    private JComboBox comboBox;
15    private JRadioButton radioButton;
16    private JList list;
17    private JButton clearButton;
18    Container container;
19
20    private String names[] = {
21       "Anteater", "Caterpillar", "Centipede", "Fire Fly" };
22
23    // set up GUI and register event handlers
24    public Events()
25    {
26       super( "Events" );
27
28       // create GUI components
29       output = new JTextArea( 10, 30 );
30       output.setLineWrap( true );
31       output.setEditable( false );
32       output.setBackground( Color.white );
33       output.setForeground( Color.black );
34
35       // add the output area to a scroll pane
36       // so the user can scroll the output
37       scrollPane = new JScrollPane( output );
38
39       // comboBox listens for item and key events
40       comboBox = new JComboBox( names );
41       comboBox.addItemListener( this );
42       comboBox.addKeyListener( this );
43
44       // radioButton listens for action events
45       radioButton = new JRadioButton( "Select Me", false );
46       radioButton.addActionListener( this );
47
48       // list listens for list selection events
49       list = new JList( names );
50       list.addListSelectionListener( this );
51
52       // clear button for clearing the output area
53       clearButton = new JButton( "Clear" );
54       clearButton.addActionListener(
55
56          // anonymous inner class for clearing the output area
57          new ActionListener() {
58
```

```
59              public void actionPerformed( ActionEvent event )
60              {
61                 output.setText( "" );
62              }
63
64           } // end anonymous inner class
65
66        ); // end call to addActionListener
67
68        // application listens to its own key and mouse events
69        addMouseMotionListener( this );
70        addMouseListener( this );
71
72        panel1 = new JPanel();
73        panel1.add( comboBox );
74        panel1.add( radioButton );
75        panel1.add( list );
76        panel1.add( clearButton );
77
78        // add components to container
79        container = getContentPane();
80        container.setLayout( new BorderLayout() );
81        container.add( scrollPane, BorderLayout.CENTER );
82        container.add( panel1, BorderLayout.SOUTH );
83
84        setSize( 375, 325 );
85        setVisible( true );
86
87     }  // end constructor Events
88
89     // ActionListener event handlers
90     public void actionPerformed( ActionEvent event )
91     {
92        display( "ActionEvent", event );
93     }
94
95     // ItemListener event handlers
96     public void itemStateChanged( ItemEvent event )
97     {
98        display( "ItemEvent", event );
99     }
100
101    // MouseListener event handlers
102    public void mouseClicked( MouseEvent event )
103    {
104       display( "MouseEvent", event );
105    }
106
107    public void mouseEntered( MouseEvent event )
108    {
109       display( "MouseEvent", event );
110    }
111
112    public void mouseExited( MouseEvent event )
113    {
```

```
114        display( "MouseEvent", event );
115     }
116
117     public void mousePressed( MouseEvent event )
118     {
119        display( "MouseEvent", event );
120     }
121
122     public void mouseReleased( MouseEvent event )
123     {
124        display( "MouseEvent", event );
125     }
126
127     // MouseMotionListener event handlers
128     public void mouseDragged( MouseEvent event )
129     {
130        display( "MouseEvent", event );
131     }
132
133     public void mouseMoved( MouseEvent event )
134     {
135        display( "MouseEvent", event );
136     }
137
138     // KeyListener event handlers
139     public void keyPressed( KeyEvent event )
140     {
141        display( "KeyEvent", event );
142     }
143
144     public void keyReleased( KeyEvent event )
145     {
146        display( "KeyEvent", event );
147     }
148
149     public void keyTyped( KeyEvent event )
150     {
151        display( "KeyEvent", event );
152     }
153
154     // ListSelectionListener event handlers
155     public void valueChanged( ListSelectionEvent event )
156     {
157        display( "ListSelectionEvent", event );
158     }
159
160     // display event occurred to output
161     public void display( String eventName, Object event )
162     {
163        output.append(
164           eventName + "occurred\n" + event.toString() + "\n\n" );
165     }
166
167     public static void main( String args[] )
168     {
```

```
169         Events app = new Events();
170         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
171      }
172
173   }  // end class Events
```



13.23 Modify your solution to Exercise 13.17 to enable the user to select a font and a font size and type text into a `JTextField`. When the user presses *Enter*, the text should be displayed in the chosen font and size. Modify the program further to allow the user to specify the exact position at which the text should be displayed.

ANS:

```
1  // Exercise 13.23 Solution: Painter2.java
2  // Program paints shapes and text of different fonts and colors.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Painter2 extends JFrame {
9     private int topX, topY, width, fontSize,
10       height, bottomX, bottomY, shape;
11    private boolean clear, textOn, filled;
12    private Color drawingColor;
13    private String font;
14    private JTextField text;
15    private JPanel panel1, panel2, panel3;
16    private JRadioButton ovalBox, rectBox, lineBox;
17    private ButtonGroup shapeGroup;
18    private JCheckBox fillBox;
19    private JComboBox colorList, fontList, sizeList;
20    private JButton clearButton;
21    private String colorNames[] = {"Black", "Green", "Blue",
22       "Red", "Cyan" };
23    private Color colors[] = { Color.black, Color.green, Color.blue,
24       Color.red, Color.cyan };
```

```
25    private String fontNames[] = { "Serif", "SansSerif", "Monospaced" };
26    private String sizeNames[] =  {"9", "10", "22", "72" };
27    private int sizes[] = { 9, 10, 22, 72 };
28    private final int OVAL = 1, LINE = 2, RECT = 3;
29
30    private ToolWindow tools;
31
32    // Painter2 constructor
33    public Painter2()
34    {
35       addMouseListener( new MouseHandler() );
36
37       // set defaults for painting
38       drawingColor = Color.black;
39       shape = OVAL;
40       font = "Serif";
41       fontSize = 9;
42
43       setSize( 300, 300 );
44       setVisible( true );
45
46       // create new ToolWindow
47       tools = new ToolWindow();
48    }
49
50    // paint the new window.  super is not called so
51    // that the previous images will not be erased.
52    public void paint( Graphics g )
53    {
54       g.setColor( drawingColor );
55
56       // draw text
57       if ( textOn ) {
58          g.setFont( new Font( font, Font.PLAIN, fontSize ) );
59          g.drawString( text.getText(), topX, topY );
60          textOn = false;
61          return;
62       }
63
64       // set shape's top left coordinates
65       if ( shape != LINE ) {
66          topX = Math.min( topX, bottomX );
67          topY = Math.min( topY, bottomY );
68       }
69
70       // draw filled shape
71       if ( filled && shape != LINE )
72
73          switch ( shape ) {
74
75             case OVAL:
76                g.fillOval( topX, topY,  width, height );
77                break;
78
```

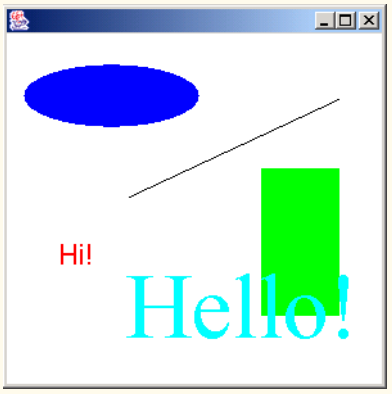
```
79                case RECT:
80                   g.fillRect( topX, topY, width, height );
81                   break;
82             }
83
84          // draw unfilled shapes
85          else
86
87             switch ( shape ) {
88
89                case OVAL:
90                   g.drawOval( topX, topY,  width, height );
91                   break;
92
93                case LINE:
94                   g.drawLine( topX, topY, bottomX, bottomY );
95                   break;
96
97                case RECT:
98                   g.drawRect( topX, topY, width, height );
99                   break;
100            }
101
102         // clear background
103         if ( clear == true ) {
104            g.setColor( Color.white );
105            g.fillRect( 0, 0, getSize().width, getSize().height );
106            clear = false;
107         }
108
109      }  // end method paint
110
111      // inner class for window containing GUI
112      private class ToolWindow extends JFrame {
113
114         // ToolWindow constructor
115         public ToolWindow()
116         {
117            // set up to edit text
118            text = new JTextField( "Text", 20 );
119            text.addActionListener(
120
121               // anonymous inner class to handle text drawing
122               new ActionListener () {
123
124                  public void actionPerformed( ActionEvent event )
125                  {
126                     textOn = true;
127                     repaint();
128                  }
129
130               }  // end anonymous inner class
131
132            ); // end call to addActionListener
```

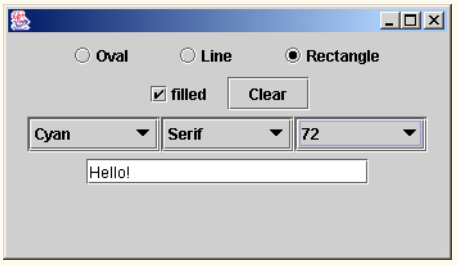
```
133
134          // set up to choose font
135          fontList = new JComboBox( fontNames );
136          fontList.setMaximumRowCount( 3 );
137          fontList.addItemListener(
138
139             new ItemListener() {  // anonymous inner class to select font
140
141                // change font
142                public void itemStateChanged( ItemEvent event )
143                {
144                   font = fontNames[ fontList.getSelectedIndex() ];
145                }
146
147             }  // end anonymous inner class
148
149          ); // end call to addItemListener
150
151          // set up to choose font size
152          sizeList = new JComboBox( sizeNames );
153          sizeList.setMaximumRowCount( 3 );
154          sizeList.addItemListener(
155
156             // anonymous inner class to select font size
157             new ItemListener() {
158
159                // change font size
160                public void itemStateChanged( ItemEvent event )
161                {
162                   fontSize = sizes[ sizeList.getSelectedIndex() ];
163                }
164
165             }  // end anonymous inner class
166
167          ); // end call to addItemListener
168
169          // set up to choose color
170          colorList = new JComboBox( colorNames );
171          colorList.setMaximumRowCount( 3 );
172          colorList.addItemListener(
173
174             new ItemListener() {  // anonymous inner class to select color
175
176                // change color
177                public void itemStateChanged( ItemEvent event )
178                {
179                   drawingColor = colors[ colorList.getSelectedIndex() ];
180                }
181
182             }  // end anonymous inner class
183
184          ); // end call to addItemListener
185
186          // set up clear button
187          clearButton = new JButton( "Clear" );
```

```
188          clearButton.addActionListener( new ClearButtonHandler() );
189
190          // set up to choose filled
191          fillBox  = new JCheckBox( "filled" );
192          FillBoxHandler fillHandle = new FillBoxHandler();
193          fillBox.addItemListener( fillHandle );
194
195          // set up to choose shapes
196          ovalBox = new JRadioButton( "Oval", true );
197          lineBox = new JRadioButton( "Line", false );
198          rectBox = new JRadioButton( "Rectangle", false );
199          RadioButtonHandler handler = new RadioButtonHandler();
200          ovalBox.addItemListener( handler );
201          lineBox.addItemListener( handler );
202          rectBox.addItemListener( handler );
203          shapeGroup = new ButtonGroup();
204          shapeGroup.add( ovalBox );
205          shapeGroup.add( lineBox );
206          shapeGroup.add( rectBox );
207
208          // set up GUI layout
209          panel1 = new JPanel();
210          panel2 = new JPanel();
211          panel3 = new JPanel();
212
213          panel1.setLayout( new GridLayout( 1, 3 ) );
214          panel2.setLayout( new GridLayout( 1, 2 ) );
215          panel3.setLayout( new GridLayout( 1, 3 ) );
216
217          panel1.add( ovalBox );
218          panel1.add( lineBox );
219          panel1.add( rectBox );
220          panel2.add( fillBox );
221          panel2.add( clearButton );
222          panel3.add( new JScrollPane ( colorList ) );
223          panel3.add( new JScrollPane ( fontList ) );
224          panel3.add( new JScrollPane ( sizeList ) );
225
226          Container container = getContentPane();
227          container.setLayout( new FlowLayout() );
228          container.add( panel1 );
229          container.add( panel2 );
230          container.add( panel3 );
231          container.add( text );
232
233          setSize( 350, 200 );
234          setLocation( 300, 0 );
235          setVisible( true );
236
237       }  // end ToolWindow constructor
238
239    }  // end inner class ToolWindow
240
```

```
241    // set coordinate and dimension values
242    private class MouseHandler extends MouseAdapter {
243
244       public void mousePressed( MouseEvent event )
245       {
246          topX = event.getX();
247          topY = event.getY();
248       }
249
250       public void mouseReleased( MouseEvent event )
251       {
252          bottomX = event.getX();
253          bottomY = event.getY();
254          width = Math.abs( topX - bottomX );
255          height = Math.abs( topY - bottomY );
256
257          repaint();
258       }
259
260    }  // end inner class MouseHandler
261
262    // clear background
263    private class ClearButtonHandler implements ActionListener {
264
265       public void actionPerformed( ActionEvent event )
266       {
267          clear = true;
268          repaint();
269       }
270
271    }  // end inner class ClearButtonHandler
272
273    // determine which type of shape to draw
274    private class RadioButtonHandler implements ItemListener {
275
276       public void itemStateChanged( ItemEvent event )
277       {
278          if ( event.getSource() == ovalBox )
279             shape = OVAL;
280
281          else if ( event.getSource() == lineBox )
282             shape = LINE;
283
284          else if ( event.getSource() == rectBox )
285             shape = RECT;
286       }
287
288    }  // end inner class RadioButtonHandler
289
290    // determine if shape should be filled
291    private class FillBoxHandler implements ItemListener {
292
293       public void itemStateChanged( ItemEvent event )
294       {
```

```
295          if ( event.getStateChange() == ItemEvent.SELECTED )
296             filled = true;
297
298          else
299             filled = false;
300       }
301
302    }  // end inner class FillBoxHandler
303
304    public static void main( String args[] )
305    {
306       Painter2 application = new Painter2();
307       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
308    }
309
310 }  // end class Painter2
```





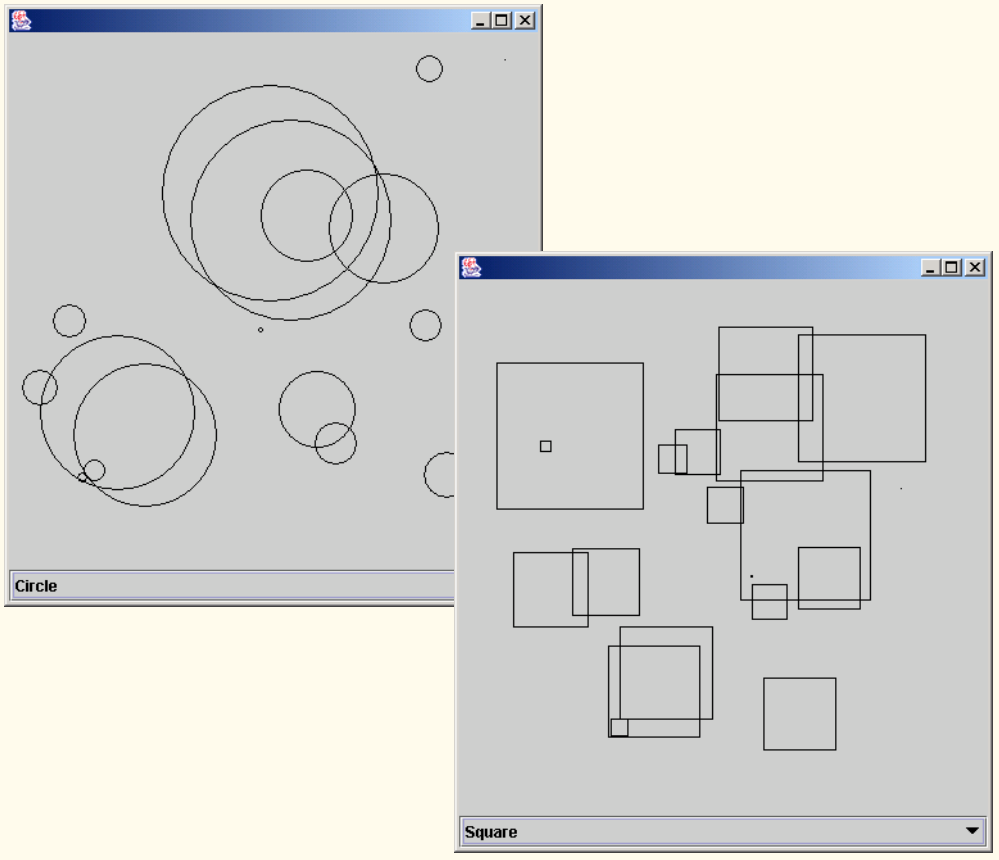
13.24 Write a program that allows the user to select a shape from a `JComboBox` and draws that shape 20 times with random locations and dimensions in method `paint`. The first item in the `JComboBox` should be the default shape that is displayed the first time `paint` is called.

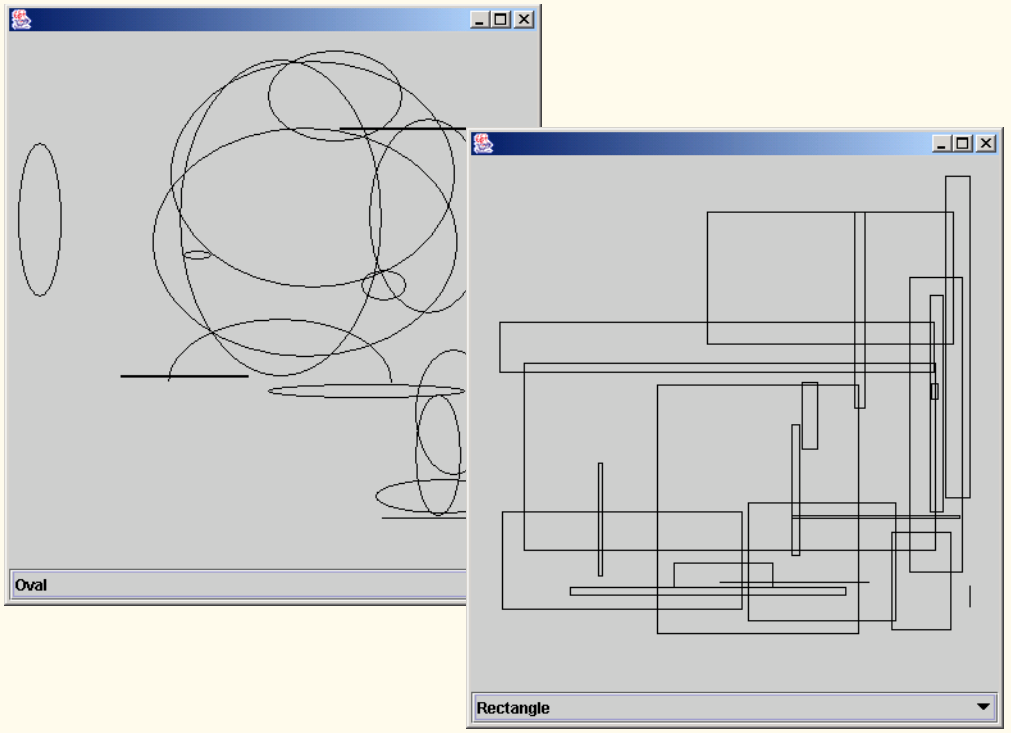
ANS:

```
1  // Exercise 13.24 Solution: SelectShape.java
2  // Draw a shape 20 times in random positions
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class SelectShape extends JFrame {
8     private final int CIRCLE = 0, SQUARE = 1, OVAL = 2,
9        RECTANGLE = 3, SIZE = 400;
10    private JComboBox choice;
11    private String[] shapes = { "Circle", "Square", "Oval", "Rectangle" };
12    private int shape;
13
14    public SelectShape()
15    {
```

```
16          choice = new JComboBox( shapes );
17          choice.addItemListener(
18
19             new ItemListener() {
20
21                public void itemStateChanged( ItemEvent e )
22                {
23                   setShape( choice.getSelectedIndex() );
24                   repaint();
25                }
26             }
27          );
28
29          getContentPane().add( choice, BorderLayout.SOUTH );
30
31          // add some to compensate for edges, combo box
32          setSize( SIZE + 20, SIZE + 70 );
33          setVisible( true );
34       }
35
36       // draw the new shape in random locations 20 times
37       public void paint( Graphics g )
38       {
39          super.paint( g );
40
41          for ( int count = 1; count <= 20; count++ ) {
42
43             // add 10 and 25 to prevent drawing over edge
44             int x = ( int ) ( Math.random() * SIZE ) + 10;
45             int y = ( int ) ( Math.random() * SIZE ) + 25;
46             int width = ( int ) ( Math.random() * ( SIZE - x ) );
47             int height = ( int ) ( Math.random() * ( SIZE - y ) );
48
49             // used for circle and square, to prevent drawing off the window
50             int diameter = width;
51             if ( width > height )
52                diameter = height;
53
54             // draw the appropriate shape
55             switch ( shape ) {
56
57                case CIRCLE:
58                   g.drawOval( x, y, diameter, diameter );
59                   break;
60
61                case SQUARE:
62                   g.drawRect( x, y, diameter, diameter );
63                   break;
64
65                case OVAL:
66                   g.drawOval( x, y, width, height );
67                   break;
68
69                case RECTANGLE:
70                   g.drawRect( x, y, width, height );
```

```
71                   break;
72
73               }
74            }
75
76      } // end method paint
77
78      // set new shape
79      public void setShape( int preference )
80      {
81         shape = preference;
82      }
83
84      public static void main( String args[] )
85      {
86         SelectShape app = new SelectShape();
87         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
88      }
89
90   }  // end class SelectShape
```





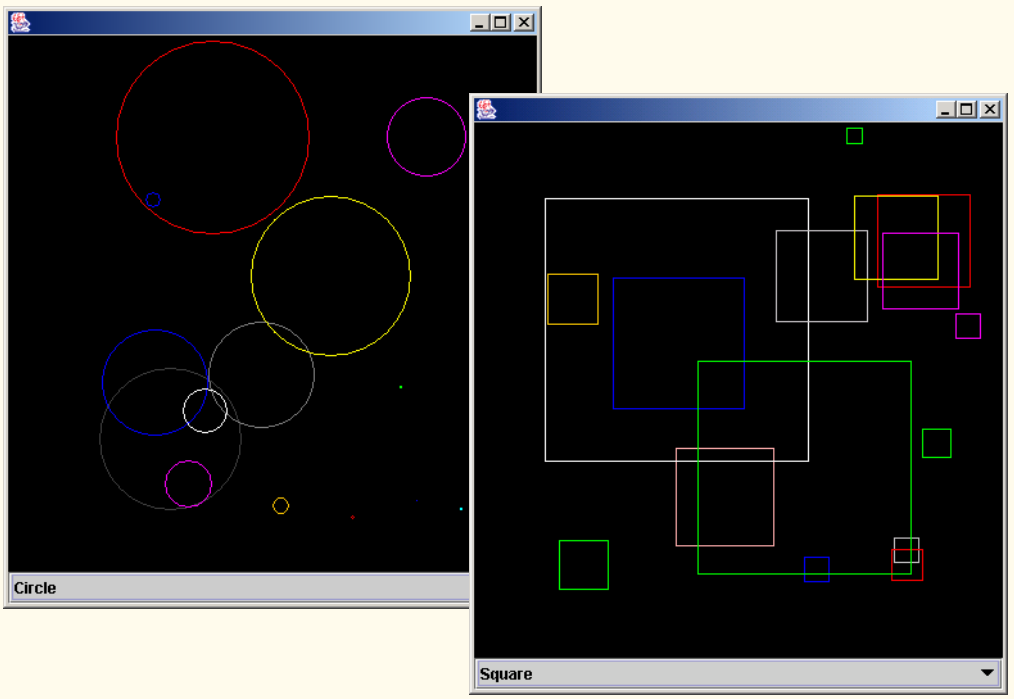
13.25 Modify Exercise 13.24 to draw each of the 20 randomly sized shapes in a randomly selected color. Use all 13 predefined `Color` objects in an array of `Colors`.

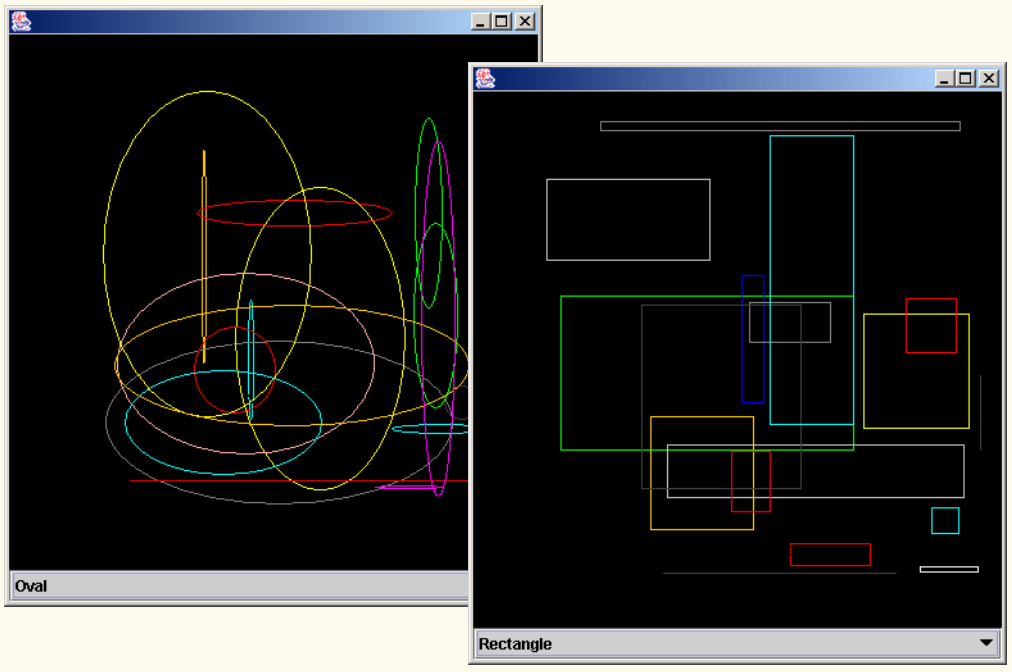
ANS:

```
1  // Exercise 13.25 Solution: SelectShape2.java
2  // Draw a shape in 20 random positions, shapes, and colors
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class SelectShape2 extends JFrame {
8     private final int CIRCLE = 0, SQUARE = 1, OVAL = 2,
9        RECTANGLE = 3, SIZE = 400;
10    private JComboBox choice;
11    private String[] shapes = { "Circle", "Square", "Oval", "Rectangle" };
12    private int shape;
13
14    Color colors[] = { Color.green, Color.pink, Color.black, Color.orange,
15       Color.magenta, Color.white, Color.cyan, Color.gray, Color.darkGray,
16       Color.blue, Color.yellow, Color.red, Color.lightGray };
17
18    public SelectShape2()
19    {
20       choice = new JComboBox( shapes );
21       choice.addItemListener(
```

```
22
23          new ItemListener() {
24
25             public void itemStateChanged( ItemEvent e )
26             {
27                setShape( choice.getSelectedIndex() );
28                repaint();
29             }
30          }
31       );
32
33       getContentPane().add( choice, BorderLayout.SOUTH );
34
35       // make the graphics easier to see
36       getContentPane().setBackground( Color.black );
37
38       // add some to compensate for edges, combo box
39       setSize( SIZE + 20, SIZE + 70 );
40       setVisible( true );
41    }
42
43    // draw the new shape in random locations 20 times
44    public void paint( Graphics g )
45    {
46       super.paint( g );
47
48       for ( int count = 1; count <= 20; count++ ) {
49
50          // add 10 and 25 to prevent drawing over edge
51          int x = ( int ) ( Math.random() * SIZE ) + 10;
52          int y = ( int ) ( Math.random() * SIZE ) + 25;
53          int width = ( int ) ( Math.random() * ( SIZE - x ) );
54          int height = ( int ) ( Math.random() * ( SIZE - y ) );
55
56          // used for circle and square, to prevent drawing off the window
57          int diameter = width;
58          if ( width > height )
59             diameter = height;
60
61          // set random color
62          int color = ( int ) ( Math.random() * colors.length );
63          g.setColor( colors[ color ] );
64
65          // draw the appropriate shape
66          switch ( shape ) {
67
68             case CIRCLE:
69                g.drawOval( x, y, diameter, diameter );
70                break;
71
72             case SQUARE:
73                g.drawRect( x, y, diameter, diameter );
74                break;
75
```

```
76                case OVAL:
77                   g.drawOval( x, y, width, height );
78                   break;
79
80                case RECTANGLE:
81                   g.drawRect( x, y, width, height );
82                   break;
83             }
84          }
85
86       } // end method paint
87
88       // set new shape
89       public void setShape( int preference )
90       {
91          shape = preference;
92       }
93
94       public static void main( String args[] )
95       {
96          SelectShape2 app = new SelectShape2();
97          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
98       }
99
100   }  // end class SelectShape2.
```





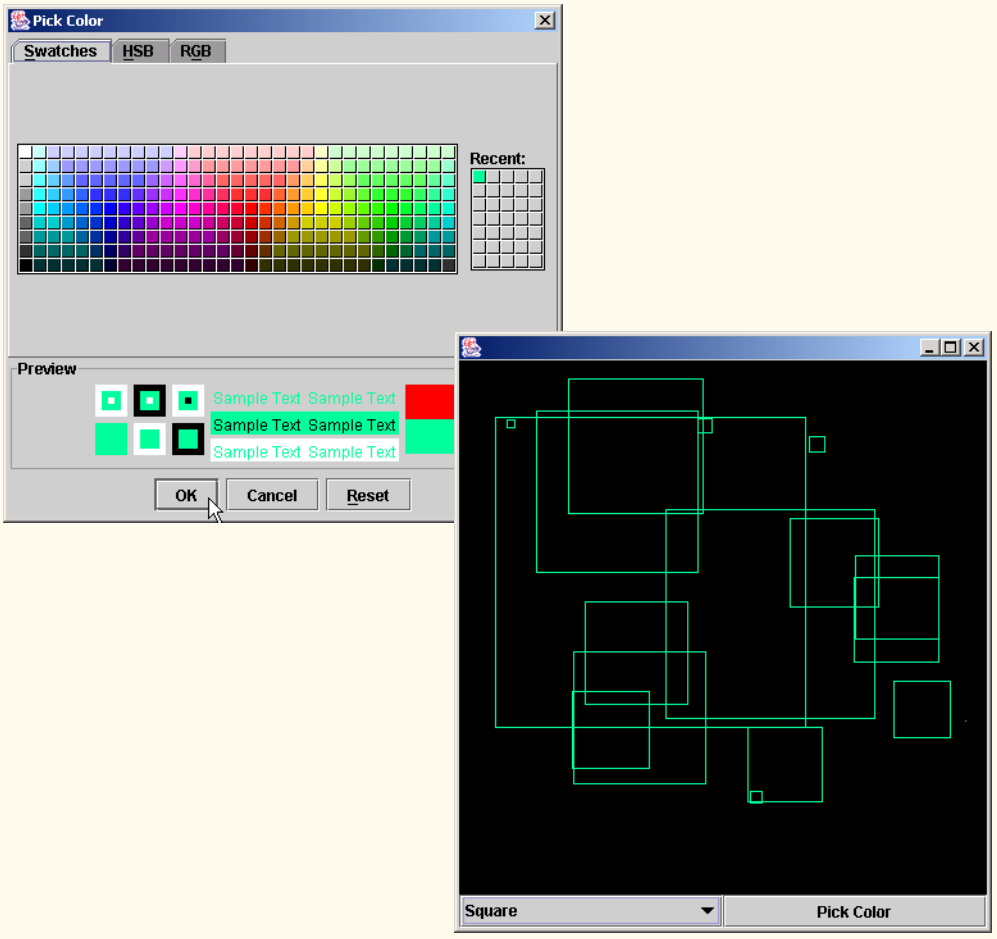
13.26 Modify Exercise 13.25 to allow the user to select the color in which shapes should be drawn from a `JColorChooser` dialog.

ANS:

```
1  // Exercise 13.26 Solution: SelectShape3.java
2  // Draw a shape in 20 random positions, shapes
3  // and allow the user to choose the color
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class SelectShape3 extends JFrame {
9     private final int CIRCLE = 0, SQUARE = 1, OVAL = 2,
10       RECTANGLE = 3, SIZE = 400;
11    private JComboBox choice;
12    private String[] shapes = { "Circle", "Square", "Oval", "Rectangle" };
13    private JButton chooseColor;
14    private JPanel panel;
15    private int shape;
16    private Color color;
17
18    public SelectShape3()
19    {
20       choice = new JComboBox( shapes );
21       choice.addItemListener(
22
23          new ItemListener() {
```

```
24
25             public void itemStateChanged( ItemEvent e )
26             {
27                setShape( choice.getSelectedIndex() );
28                repaint();
29             }
30          }
31       );
32
33       // add button that pops up a color dialog
34       chooseColor = new JButton( "Pick Color" );
35       chooseColor.addActionListener(
36
37          new ActionListener() {
38
39             public void actionPerformed( ActionEvent e )
40             {
41                color = JColorChooser.showDialog( null,
42                   "Pick Color", Color.red );
43                repaint();
44             }
45          }
46       );
47
48       // add components to GUI
49       panel = new JPanel();
50       panel.setLayout( new GridLayout( 1, 2 ) );
51       panel.add( choice );
52       panel.add( chooseColor );
53       Container container = getContentPane();
54       container.add( panel, BorderLayout.SOUTH );
55
56       // make the graphics easier to see
57       container.setBackground( Color.black );
58
59       // add some to compensate for edges, combo box
60       setSize( SIZE + 20, SIZE + 70 );
61       setVisible( true );
62    }
63
64    // draw the new shape in random locations 20 times
65    public void paint( Graphics g )
66    {
67       super.paint( g );
68
69       for ( int count = 1; count <= 20; count++ ) {
70
71          // add 10 and 25 to prevent drawing over edge
72          int x = ( int ) ( Math.random() * SIZE ) + 10;
73          int y = ( int ) ( Math.random() * SIZE ) + 25;
74          int width = ( int ) ( Math.random() * ( SIZE - x ) );
75          int height = ( int ) ( Math.random() * ( SIZE - y ) );
76
77          // used for circle and square, to prevent drawing off the window
78          int diameter = width;
```

```
79            if ( width > height )
80               diameter = height;
81
82            // set color chosen by user
83            g.setColor( color );
84
85            // draw the appropriate shape
86            switch ( shape ) {
87
88               case CIRCLE:
89                  g.drawOval( x, y, diameter, diameter );
90                  break;
91
92               case SQUARE:
93                  g.drawRect( x, y, diameter, diameter );
94                  break;
95
96               case OVAL:
97                  g.drawOval( x, y, width, height );
98                  break;
99
100              case RECTANGLE:
101                 g.drawRect( x, y, width, height );
102                 break;
103           }
104        }
105
106     } // end method paint
107
108     // set new shape
109     public void setShape( int preference )
110     {
111        shape = preference;
112     }
113
114     public static void main( String args[] )
115     {
116        SelectShape3 app = new SelectShape3();
117        app.setDefaultCloseOperation( EXIT_ON_CLOSE );
118     }
119
120  }  // end class SelectShape3
```

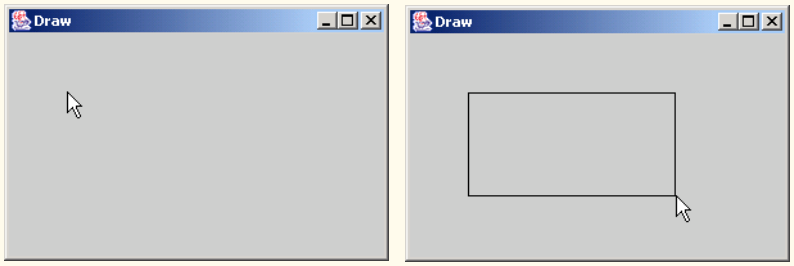


13.27 Write a program using methods from interface `MouseListener`, that allows the user to press the mouse button, drag the mouse and release the mouse button. When the mouse is released, draw a rectangle with the appropriate upper-left corner, width and height. [*Hint*: The `mousePressed` method should capture the set of coordinates at which the user presses and holds the mouse button initially, and the `mouseReleased` method should capture the set of coordinates at which the user releases the mouse button. Both methods should store the appropriate coordinate values. All calculations of the width, height and upper-left corner should be performed by the `paint` method before the shape is drawn.]

ANS:)

```
1  // Exercise 13.27 Solution: DrawRectangle.java
2  // Program draws a rectangle with the mouse.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
```

```
 7  public class DrawRectangle extends JFrame {
 8     private int topX, topY, width, height, bottomX, bottomY;
 9
10     // constructor
11     public DrawRectangle()
12     {
13        super( "Draw" );
14        addMouseListener( new MouseHandler() );
15        setSize( 300, 200 );
16        setVisible( true );
17     }
18
19     // draw rectangle
20     public void paint( Graphics g )
21     {
22        super.paint( g );
23
24        width = Math.abs( topX - bottomX );
25        height = Math.abs( topY - bottomY );
26        int upperX = Math.min( topX, bottomX );
27        int upperY = Math.min( topY, bottomY );
28
29        g.drawRect( upperX, upperY, width, height );
30     }
31
32     public static void main( String args[] )
33     {
34        DrawRectangle application = new DrawRectangle();
35        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
36     }
37
38     // set coordinate values
39     private class MouseHandler extends MouseAdapter {
40
41        public void mouseReleased( MouseEvent event )
42        {
43           bottomX = event.getX();
44           bottomY = event.getY();
45           repaint();
46        }
47
48        public void mousePressed( MouseEvent event )
49        {
50           topX = event.getX();
51           topY = event.getY();
52        }
53
54     }  // end inner class MouseHandler
55
56  }  // end class DrawRectangle
```



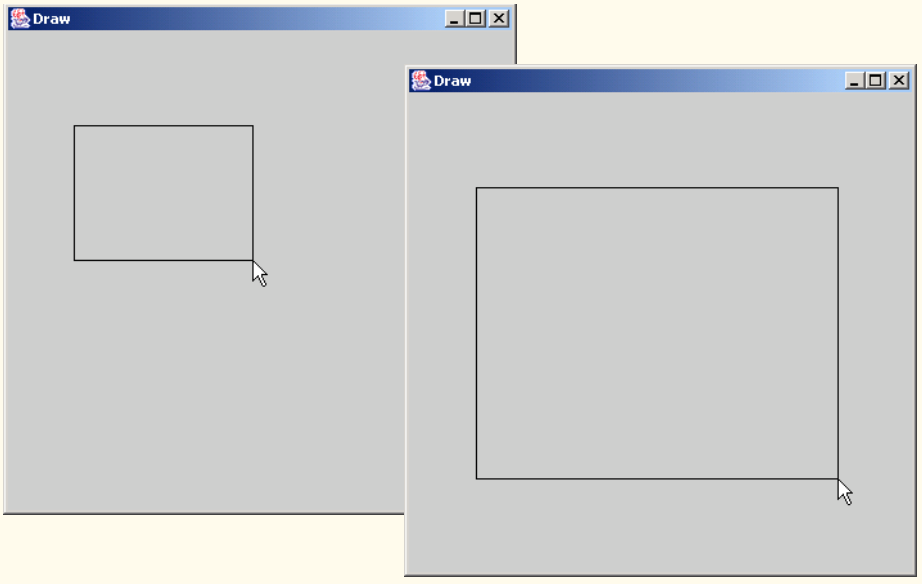
13.28 Modify Exercise 13.27 to provided a "rubberbanding" effect. As the user drags the mouse, the user should be able to see the current size of the rectangle to know exactly what the rectangle will look like when the mouse button is released. [*Hint*: Method `mouseDragged` should perform the same tasks as `mouseReleased`.]

ANS:)

```
1  // Exercise 13.28 Solution: Draw2.java
2  // Program draws a shape chosen by the user with the mouse with a
3  // rubber-banding effect
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Draw2 extends JFrame {
9     private int topX, topY, width, height, upperX, upperY,
10       bottomX, bottomY;
11
12    // constructor
13    public Draw2()
14    {
15       super( "Draw" );
16       addMouseListener( new MouseHandler() );
17       addMouseMotionListener( new MouseMotionHandler() );
18
19       setSize( 400, 400 );
20       setVisible( true );
21    }
22
23    // mutator methods
24    public void setTopX( int x )
25    {
26       topX = x;
27    }
28
29    public void setTopY( int y )
30    {
31       topY = y;
32    }
33
34    public void setBottomX( int x )
35    {
```

```
36          bottomX = x;
37       }
38
39       public void setBottomY( int y )
40       {
41          bottomY = y;
42       }
43
44       // draw new rectangle
45       public void paint( Graphics g )
46       {
47          super.paint( g );
48
49          width = Math.abs( topX - bottomX );
50          height = Math.abs( topY - bottomY );
51          upperX = Math.min( topX, bottomX );
52          upperY = Math.min( topY, bottomY );
53
54          g.drawRect( upperX, upperY, width, height );
55       }
56
57       public static void main( String args[] )
58       {
59          Draw2 app = new Draw2();
60          app.setDefaultCloseOperation( EXIT_ON_CLOSE );
61       }
62
63       // inner class finds coordinates of next rectangle
64       private class MouseHandler extends MouseAdapter {
65
66          // find initial coordinates
67          public void mouseReleased( MouseEvent e )
68          {
69             setBottomX( e.getX() );
70             setBottomY( e.getY() );
71             repaint();
72          }
73
74          // find final coordinates
75          public void mousePressed( MouseEvent e )
76          {
77             setTopX( e.getX() );
78             setTopY( e.getY() );
79          }
80
81       }  // end inner class MouseHandler
82
83       // inner class monitors coordinates of rectangle
84       private class MouseMotionHandler extends MouseMotionAdapter {
85
86          // find current coordinates
87          public void mouseDragged( MouseEvent e )
88          {
89             setBottomX( e.getX() );
```

```
90             setBottomY( e.getY() );
91             repaint();
92          }
93
94       }  // end inner class MouseMotionHandler
95
96    }  // end class Draw2
```



13.29 Modify Exercise 13.28 to allow the user to select which shape to draw. A `JComboBox` should provide options including at least rectangle, oval, line and rounded rectangle.

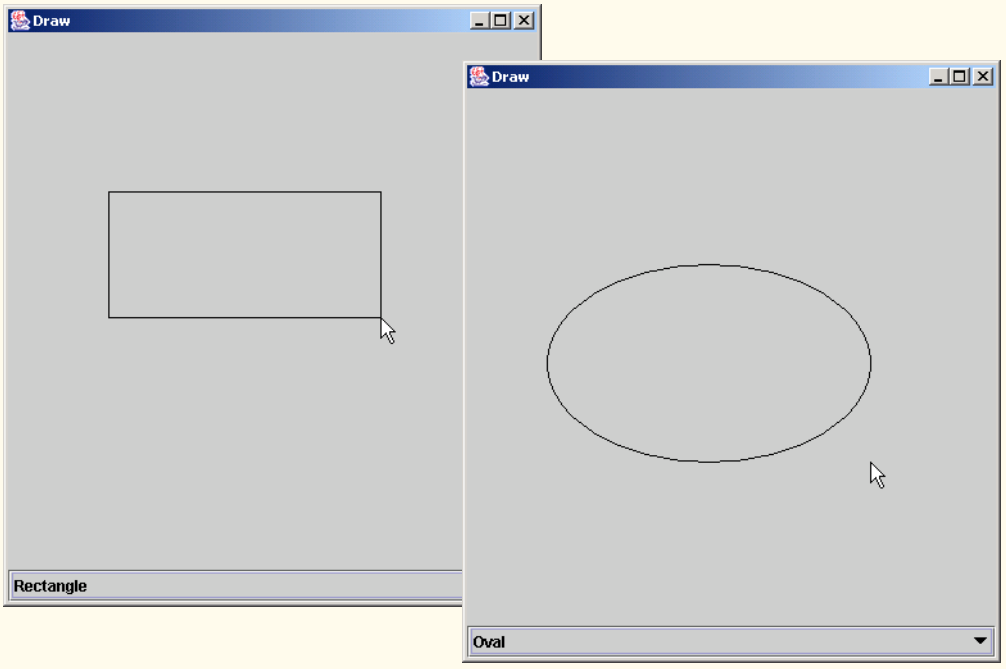
ANS:

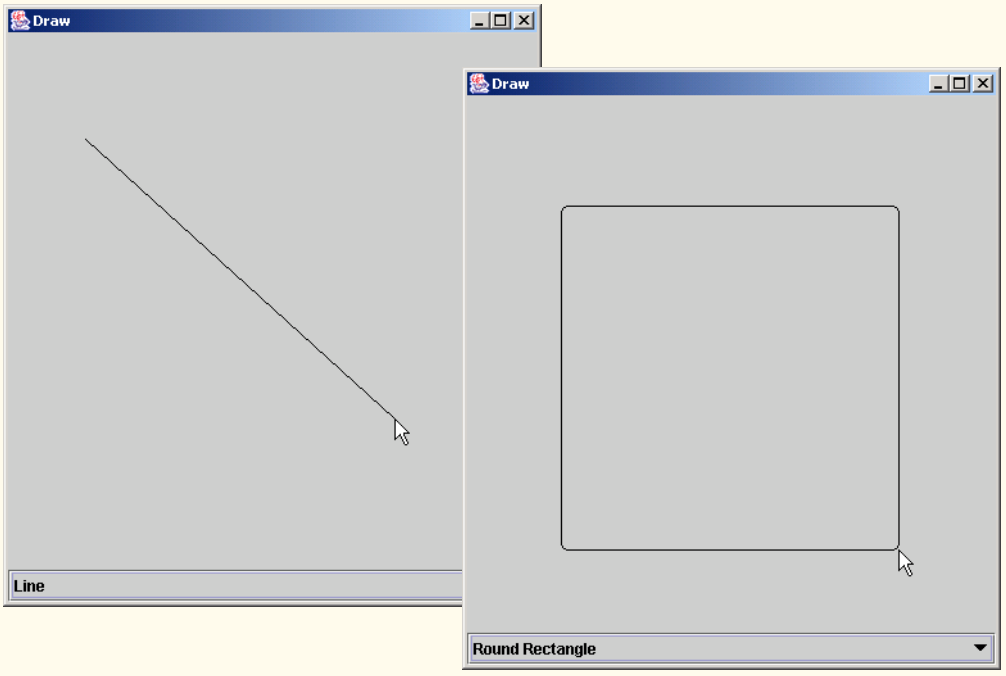
```
1  // Exercise 13.29 Solution: Draw3.java
2  // Program draws a shape chosen by the user with the mouse
3  // with a rubber-banding effect
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Draw3 extends JFrame {
9     private int topX, topY, width, height, upperX, upperY,
10       bottomX, bottomY, shape;
11    private final int RECTANGLE = 0, OVAL = 1, LINE = 2,
12       ROUNDEDRECTANGLE = 3, SIZE = 400;
13    private JComboBox choice;
14    private String[] shapes = {
15       "Rectangle", "Oval", "Line", "Round Rectangle" };
16
17    // constructor
18    public Draw3()
19    {
```

```
20          super( "Draw" );
21          addMouseListener( new MouseHandler() );
22          addMouseMotionListener( new MouseMotionHandler() );
23
24          // set up GUI
25          choice = new JComboBox( shapes );
26          choice.addItemListener(
27
28             new ItemListener() {
29
30                public void itemStateChanged( ItemEvent e )
31                {
32                   setShape( choice.getSelectedIndex() );
33                   repaint();
34                }
35             }
36          );
37
38          // set default shape to Rectangle
39          shape = RECTANGLE;
40
41          Container container = getContentPane();
42          container.setLayout( new BorderLayout() );
43          container.add( choice, BorderLayout.SOUTH );
44
45          // add some to compensate for edges, combo box
46          setSize( SIZE + 20, SIZE + 70 );
47          setVisible( true );
48       }
49
50       // mutator methods
51       public void setTopX( int x )
52       {
53          topX = x;
54       }
55
56       public void setTopY( int y )
57       {
58          topY = y;
59       }
60
61       public void setBottomX( int x )
62       {
63          bottomX = x;
64       }
65
66       public void setBottomY( int y )
67       {
68          bottomY = y;
69       }
70
71       public void setShape( int preference )
72       {
73          shape = preference;
74       }
```

```
75
76
77       // draw the chosen shape
78       public void paint( Graphics g )
79       {
80          super.paint( g );
81
82          width = Math.abs( topX - bottomX );
83          height = Math.abs( topY - bottomY );
84          upperX = Math.min( topX, bottomX );
85          upperY = Math.min( topY, bottomY );
86
87          // for drawing circles and squares
88          int diameter = width;
89          if ( width > height )
90             diameter = height;
91
92          // draw the appropriate shape
93          switch ( shape ) {
94
95             case RECTANGLE:
96                g.drawRect( upperX, upperY, width, height );
97                break;
98
99             case OVAL:
100               g.drawOval( upperX, upperY, width, height );
101               break;
102
103            case LINE:
104               g.drawLine( topX, topY, bottomX, bottomY );
105               break;
106
107            case ROUNDRECTANGLE:
108               g.drawRoundRect( upperX, upperY, width, height, 10, 10 );
109               break;
110         }
111      }
112
113      public static void main( String args[] )
114      {
115         Draw3 app = new Draw3();
116         app.setDefaultCloseOperation( EXIT_ON_CLOSE );
117      }
118
119      // inner class finds coordinates of next rectangle
120      private class MouseHandler extends MouseAdapter {
121
122         // find initial coordinates
123         public void mouseReleased( MouseEvent e )
124         {
125            setBottomX( e.getX() );
126            setBottomY( e.getY() );
127            repaint();
128         }
```

```
129
130       // find final coordinates
131       public void mousePressed( MouseEvent e )
132       {
133          setTopX( e.getX() );
134          setTopY( e.getY() );
135       }
136
137    }  // end inner class MouseHandler
138
139    // inner class monitors coordinates of rectangle
140    private class MouseMotionHandler extends MouseMotionAdapter {
141
142       // find current coordinates
143       public void mouseDragged( MouseEvent e )
144       {
145          setBottomX( e.getX() );
146          setBottomY( e.getY() );
147          repaint();
148       }
149
150    }  // end inner class MouseMotionHandler
151
152 }  // end class Draw3
```





13.30 Modify Exercise 13.29 to allow the user to select the drawing color from a `JColorChooser` dialog box.

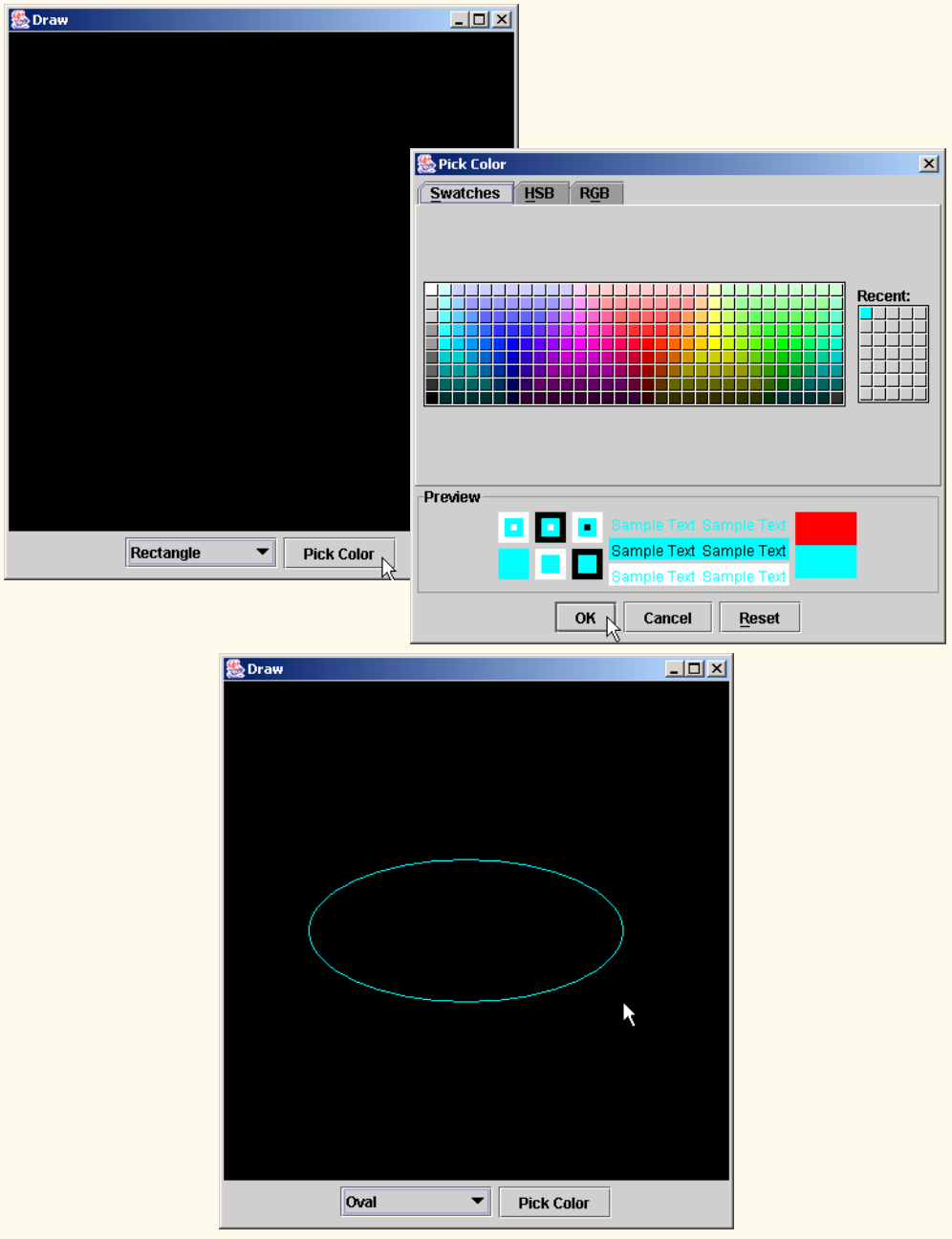
ANS:

```
1  // Exercise 13.30 Solution: Draw4.java
2  // Program draws a shape chosen by the user with the mouse
3  // with a rubber-banding effect, in a color chosen by the user
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Draw4 extends JFrame {
9     private int topX, topY, width, height, upperX, upperY,
10       bottomX, bottomY, shape;
11    private final int RECTANGLE = 0, OVAL = 1, LINE = 2,
12       ROUNDRECTANGLE = 3, SIZE = 400;
13    private JPanel panel;
14    private JComboBox choice;
15    private String[] shapes = {
16       "Rectangle", "Oval", "Line", "Round Rectangle" };
17    private JButton chooseColor;
18    private Color color;
19
20    // constructor
21    public Draw4()
22    {
```

```
23         super( "Draw" );
24         addMouseListener( new MouseHandler() );
25         addMouseMotionListener( new MouseMotionHandler() );
26
27         // set up GUI
28         choice = new JComboBox( shapes );
29         choice.addItemListener(
30
31            new ItemListener() {
32
33               public void itemStateChanged( ItemEvent e )
34               {
35                  setShape( choice.getSelectedIndex() );
36                  repaint();
37               }
38            }
39         );
40
41         // set default shape to Rectangle and color to Red
42         shape = RECTANGLE;
43         color = Color.red;
44
45         // allow user to choose a color
46         chooseColor = new JButton( "Pick Color" );
47         chooseColor.addActionListener(
48
49            new ActionListener() {
50
51               public void actionPerformed( ActionEvent e )
52               {
53                  color = JColorChooser.showDialog( null,
54                     "Pick Color", Color.red );
55               }
56            }
57         );
58
59         // add components to GUI
60         panel = new JPanel();
61         panel.add( choice );
62         panel.add( chooseColor );
63
64         Container container = getContentPane();
65         container.setLayout( new BorderLayout() );
66         container.add( panel, BorderLayout.SOUTH );
67
68         // change background to make shape more visible
69         container.setBackground( Color.black );
70
71         // add some to compensate for edges, combo box
72         setSize( SIZE + 20, SIZE + 70 );
73         setVisible( true );
74      }
75
```

```
76     // mutator methods
77     public void setTopX( int x )
78     {
79        topX = x;
80     }
81
82     public void setTopY( int y )
83     {
84        topY = y;
85     }
86
87     public void setBottomX( int x )
88     {
89        bottomX = x;
90     }
91
92     public void setBottomY( int y )
93     {
94        bottomY = y;
95     }
96
97     public void setShape( int preference )
98     {
99        shape = preference;
100    }
101
102
103    // draw the chosen shape
104    public void paint( Graphics g )
105    {
106       super.paint( g );
107
108       // set the color of the shape
109       g.setColor( color );
110
111       width = Math.abs( topX - bottomX );
112       height = Math.abs( topY - bottomY );
113       upperX = Math.min( topX, bottomX );
114       upperY = Math.min( topY, bottomY );
115
116       // for drawing circles and squares
117       int diameter = width;
118       if ( width > height )
119          diameter = height;
120
121       // draw the appropriate shape
122       switch ( shape ) {
123
124          case RECTANGLE:
125             g.drawRect( upperX, upperY, width, height );
126             break;
127
128          case OVAL:
129             g.drawOval( upperX, upperY, width, height );
```

```
130             break;
131
132          case LINE:
133             g.drawLine( topX, topY, bottomX, bottomY );
134             break;
135
136          case ROUNDRECTANGLE:
137             g.drawRoundRect( upperX, upperY, width, height, 10, 10 );
138             break;
139       }
140    }
141
142    public static void main( String args[] )
143    {
144       Draw4 app = new Draw4();
145       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
146    }
147
148    // inner class finds coordinates of next rectangle
149    private class MouseHandler extends MouseAdapter {
150
151       // find initial coordinates
152       public void mouseReleased( MouseEvent e )
153       {
154          setBottomX( e.getX() );
155          setBottomY( e.getY() );
156          repaint();
157       }
158
159       // find final coordinates
160       public void mousePressed( MouseEvent e )
161       {
162          setTopX( e.getX() );
163          setTopY( e.getY() );
164       }
165
166    }  // end inner class MouseHandler
167
168    // inner class monitors coordinates of rectangle
169    private class MouseMotionHandler extends MouseMotionAdapter {
170
171       // find current coordinates
172       public void mouseDragged( MouseEvent e )
173       {
174          setBottomX( e.getX() );
175          setBottomY( e.getY() );
176          repaint();
177       }
178
179    }  // end inner class MouseMotionHandler
180
181 }  // end class Draw4
```



13.31 Modify Exercise 13.30 to allow the user to specify whether a shape should be filled or empty when it is drawn. The user should click a `JCheckBox` to indicate filled or empty.

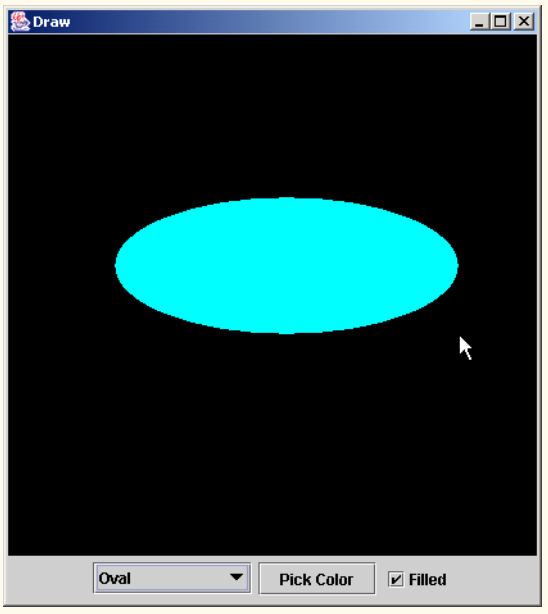
ANS:

```
1  // Exercise 13.31 Solution: Draw5.java
2  // Program draws a shape chosen by the user with the mouse
3  // with a rubber-banding effect, in a color chosen by the
4  // user, and possibly filled.
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class Draw5 extends JFrame {
10    private int topX, topY, width, height, upperX, upperY,
11       bottomX, bottomY, shape;
12    private final int RECTANGLE = 0, OVAL = 1, LINE = 2,
13       ROUNDRECTANGLE = 3, SIZE = 400;
14    private JPanel panel;
15    private JComboBox choice;
16    private String[] shapes = {
17       "Rectangle", "Oval", "Line", "Round Rectangle" };
18    private JButton chooseColor;
19    private Color color;
20    private JCheckBox filled;
21
22    // constructor
23    public Draw5()
24    {
25       super( "Draw" );
26       addMouseListener( new MouseHandler() );
27       addMouseMotionListener( new MouseMotionHandler() );
28
29       // set up GUI
30       choice = new JComboBox( shapes );
31       choice.addItemListener(
32
33          new ItemListener() {
34
35             public void itemStateChanged( ItemEvent e )
36             {
37                setShape( choice.getSelectedIndex() );
38                repaint();
39             }
40          }
41       );
42
43       // set default shape to Rectangle and color to Red
44       shape = RECTANGLE;
45       color = Color.red;
46
47       // allow user to choose a color
48       chooseColor = new JButton( "Pick Color" );
49       chooseColor.addActionListener(
50
51          new ActionListener() {
52
```

```
53              public void actionPerformed( ActionEvent e )
54              {
55                 color = JColorChooser.showDialog( null,
56                    "Pick Color", Color.red );
57              }
58           }
59        );
60
61        // allow user to specify if filled
62        filled = new JCheckBox( "Filled" );
63
64        // add components to GUI
65        panel = new JPanel();
66        panel.add( choice );
67        panel.add( chooseColor );
68        panel.add( filled );
69
70        Container container = getContentPane();
71        container.setLayout( new BorderLayout() );
72        container.add( panel, BorderLayout.SOUTH );
73
74        // change background to make shape more visible
75        container.setBackground( Color.black );
76
77        // add some to compensate for edges, combo box
78        setSize( SIZE + 20, SIZE + 70 );
79        setVisible( true );
80     }
81
82     // mutator methods
83     public void setTopX( int x )
84     {
85        topX = x;
86     }
87
88     public void setTopY( int y )
89     {
90        topY = y;
91     }
92
93     public void setBottomX( int x )
94     {
95        bottomX = x;
96     }
97
98     public void setBottomY( int y )
99     {
100       bottomY = y;
101    }
102
103    public void setShape( int preference )
104    {
105       shape = preference;
106    }
```

```
107
108
109    // draw the chosen shape
110    public void paint( Graphics g )
111    {
112       super.paint( g );
113
114       // set the color of the shape
115       g.setColor( color );
116
117       width = Math.abs( topX - bottomX );
118       height = Math.abs( topY - bottomY );
119       upperX = Math.min( topX, bottomX );
120       upperY = Math.min( topY, bottomY );
121
122       // for drawing circles and squares
123       int diameter = width;
124       if ( width > height )
125          diameter = height;
126
127       // draw the appropriate shape
128       if ( filled.isSelected() )
129          switch ( shape ) {
130
131             case RECTANGLE:
132                g.fillRect( upperX, upperY, width, height );
133                break;
134
135             case OVAL:
136                g.fillOval( upperX, upperY, width, height );
137                break;
138
139             case LINE:
140                g.drawLine( topX, topY, bottomX, bottomY );
141                break;
142
143             case ROUNDRECTANGLE:
144               g.fillRoundRect( upperX, upperY, width, height, 10, 10 );
145                break;
146          }
147       else
148          switch ( shape ) {
149
150             case RECTANGLE:
151                g.drawRect( upperX, upperY, width, height );
152                break;
153
154             case OVAL:
155                g.drawOval( upperX, upperY, width, height );
156                break;
157
158             case LINE:
159                g.drawLine( topX, topY, bottomX, bottomY );
160                break;
```

```
161
162             case ROUNDRECTANGLE:
163               g.drawRoundRect( upperX, upperY, width, height, 10, 10 );
164                break;
165          }
166    }
167
168    public static void main( String args[] )
169    {
170       Draw5 app = new Draw5();
171       app.setDefaultCloseOperation( EXIT_ON_CLOSE );
172    }
173
174    // inner class finds coordinates of next rectangle
175    private class MouseHandler extends MouseAdapter {
176
177       // find initial coordinates
178       public void mouseReleased( MouseEvent e )
179       {
180          setBottomX( e.getX() );
181          setBottomY( e.getY() );
182          repaint();
183       }
184
185       // find final coordinates
186       public void mousePressed( MouseEvent e )
187       {
188          setTopX( e.getX() );
189          setTopY( e.getY() );
190       }
191
192    }  // end inner class MouseHandler
193
194    // inner class monitors coordinates of rectangle
195    private class MouseMotionHandler extends MouseMotionAdapter {
196
197       // find current coordinates
198       public void mouseDragged( MouseEvent e )
199       {
200          setBottomX( e.getX() );
201          setBottomY( e.getY() );
202          repaint();
203       }
204
205    }  // end inner class MouseMotionHandler
206
207 }  // end class Draw5
```



14

Graphical User Interface Components: Part 2

Objectives

- To create and manipulate text areas, sliders, menus, popup menus and windows.
- To be able to create customized `JPanel` objects.
- To be able to change the look-and-feel of a GUI, using Swing's pluggable look-and-feel (PLAF).
- To be able to create a multiple-document interface with `JDesktopPane` and `JInternalFrame`.
- To be able to use additional layout managers.

I claim not to have controlled events, but confess plainly that events have controlled me.
Abraham Lincoln

A good symbol is the best argument, and is a missionary to persuade thousands.
Ralph Waldo Emerson

Capture its reality in paint!
Paul Cézanne

SELF-REVIEW EXERCISES

14.1 Fill in the blanks in each of the following statements:

a) The _____ class is used to create a menu object.

ANS: `JMenu`

b) The _____ method places a separator bar in a menu.

ANS: `addSeparator`

c) Passing `false` to a `TextArea`'s _____ method prevents its text from being modified by the user.

ANS: `setEditable`

d) `JSlider` events are handled by the _____ method of interface _____.

ANS: `stateChanged`, `ChangeListener`

e) The `GridBagConstraints` instance variable _____ is set to `CENTER` by default.

ANS: `anchor`

14.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) When the programmer creates a `JFrame`, a minimum of one menu must be created and added to the `JFrame`.

ANS: False. A `JFrame` does not require any menus.

b) The variable `fill` belongs to the `GridBagLayout` class.

ANS: False. The variable `fill` belongs to the `GridBagConstraints` class.

c) Drawing on a GUI component is performed with respect to the (0, 0) upper-left corner coordinate of the component.

ANS: True.

d) A `JTextArea`'s text is always read-only.

ANS: False. `JTextArea`s are editable by default.

e) Class `JTextArea` is a direct subclass of class `Component`.

ANS: False. `JTextArea` derives from class `JTextComponent`.

f) The default layout for a `Box` is `BoxLayout`.

ANS: True.

14.3 Find the error(s) in each of the following and explain how to correct the error(s).

a)
```
JMenubar b;
```
ANS: `JMenubar` should be `JMenuBar`.

b)
```
mySlider = JSlider( 1000, 222, 100, 450 );
```
ANS: The first argument to the constructor should be either `SwingConstants.HORIZONTAL` or `SwingConstants.VERTICAL`, and the keyword `new` must be used after the = operator.

c)
```
gbc.fill = GridBagConstraints.NORTHWEST;  // set fill
```
ANS: The constant should be either `BOTH`, `HORIZONTAL`, `VERTICAL` or `NONE`.

d)
```
// override to paint on a customized Swing component
public void paintcomponent( Graphics g )
{
   g.drawString( "HELLO", 50, 50 );
}
```
ANS: `paintcomponent` should be `paintComponent`, and the method should call `super.paintComponent( g )` as its first statement.

e)
```
// create a JFrame and display it
JFrame f = new JFrame( "A Window" );
f.setVisible( true );
```
ANS: The `JFrame`'s `setSize` method must also be called to establish the size of the window.

EXERCISES

14.4 Fill in the blanks in each of the following statements:

a) A dedicated drawing area can be declared as a subclass of _____.
ANS: `JPanel`

b) A `JMenuItem` that is a `JMenu` is called a(n) _____.
ANS: submenu

c) Both `JTextFields` and `JTextAreas` directly extend class _____.
ANS: `JTextComponent`

d) Method _____ attaches a `JMenuBar` to a `JFrame`.
ANS: `setJMenuBar`

e) Container class _____ has a default `BoxLayout`.
ANS: `Box`

f) A(n) _____ manages a set of child windows declared with class `JInternalFrame`.
ANS: `JDesktopPane`

14.5 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Menus require a `JMenuBar` object so they can be attached to a `JFrame`.
ANS: True

b) A `JPanel` object is capable of receiving mouse events.
ANS: True

c) `BoxLayout` is the default layout manager for a `JFrame`.
ANS: False. `BorderLayout` is the default layout manager for `JFrame`.

d) Method `setEditable` is a `JTextComponent` method.
ANS: True

e) `JPanel` objects are containers to which other GUI components can be attached.
ANS: True

f) Class `JFrame` directly extends class `Container`.
ANS: False. `JFrame` inherits directly from `Frame`.

g) `JApplets` can contain menus.
ANS: True

14.6 Find the error(s) in each of the following. Explain how to correct the error(s).

a)
```
x.add( new JMenuItem( "Submenu Color" ) ); // create submenu
```
ANS: A `JMenu` is used to create a submenu, not a `JMenuItem`.

b)
```
container.setLayout( m = new GridbagLayout() );
```
ANS: `GridbagLayout` should be `GridBagLayout`.

c)
```
String s = JTextArea.getText();
```
ANS: `getText` is not a `static` method of `JTextArea`.

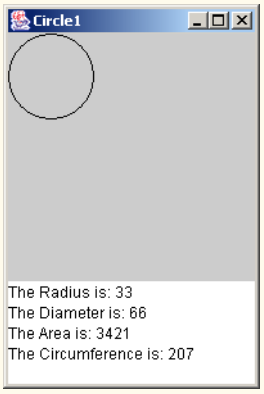
14.7 Write a program that displays a circle of random size and calculates and displays the area, radius, diameter and circumference. Use the following equations: *diameter* = *2* × *radius*, *area* = π × *radius*^{2}, *circumference* = *2* × π × *radius*. Use the constant `Math.PI` for pi (π). All drawing should be done on a subclass of `JPanel`, and the results of the calculations should be displayed in a read-only `JTextArea`.

ANS:

```
1  // Exercise 14.7 Solution: Circle1.java
2  // Program draws a circle of a random diameter and displays the radius,
3  // diameter, area and circumference.
4  import java.awt.*;
5  import java.awt.event.*;
```

```
6  import javax.swing.*;
7
8  public class Circle1 extends JFrame
9  {
10    private JTextArea display;
11
12    public Circle1()
13    {
14       super( "Circle1" );
15
16       // the panel on which all drawing is done
17       CircleCanvas theCanvas = new CircleCanvas();
18
19       // the area for writing information about the circle
20       display = new JTextArea( 5, 30 );
21       display.setEditable( false );
22
23       display.setText( "The Radius is: " + theCanvas.getRadius() +
24          "\nThe Diameter is: " + theCanvas.getDiameter() +
25          "\nThe Area is: " + theCanvas.getArea() +
26          "\nThe Circumference is: " + theCanvas.getCircumference() );
27
28       Container container = getContentPane();
29       container.add( theCanvas, BorderLayout.CENTER );
30       container.add( display, BorderLayout.SOUTH );
31
32       setSize( 200, 300 );
33       setVisible( true );
34    }
35
36    public static void main( String args[] )
37    {
38       Circle1 application = new Circle1();
39       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
40    }
41
42 }  // end class Circle1
43
44 // class to display circle and calculate circle's dimensions
45 class CircleCanvas extends JPanel {
46    private int radius;
47
48    // generate random radius and set panel size
49    public CircleCanvas()
50    {
51       radius = ( int )( 1 + Math.random() * 100 );
52       setSize( 200, 200 );
53    }
54
55    // draw circle
56    public void paintComponent( Graphics g )
57    {
58       g.drawOval( 0, 0, 2 * radius, 2 * radius );
59    }
```

```
60
61      // get diameter of circle
62      public int getDiameter()
63      {
64         return ( 2 * radius );
65      }
66
67      // get circumference of circle
68      public int getCircumference()
69      {
70         return ( int )( 2 * Math.PI * radius );
71      }
72
73      // get area of circle
74      public int getArea()
75      {
76         return ( int )( Math.PI * radius * radius );
77      }
78
79      // get radius of circle
80      public int getRadius()
81      {
82         return radius;
83      }
84
85   }  // end class CircleCanvas
```



14.8 Enhance the program of Exercise 14.7 by allowing the user to alter the radius with a `JSlider`. The program should work for all radii in the range from 100–200. As the radius changes, the diameter, area and circumference should be updated and displayed. The initial radius should be 150. Use the equations of Exercise 14.7. All drawing should be done on a subclass of `JPanel`, and the results of the calculations should be displayed in a read-only `JTextArea`.

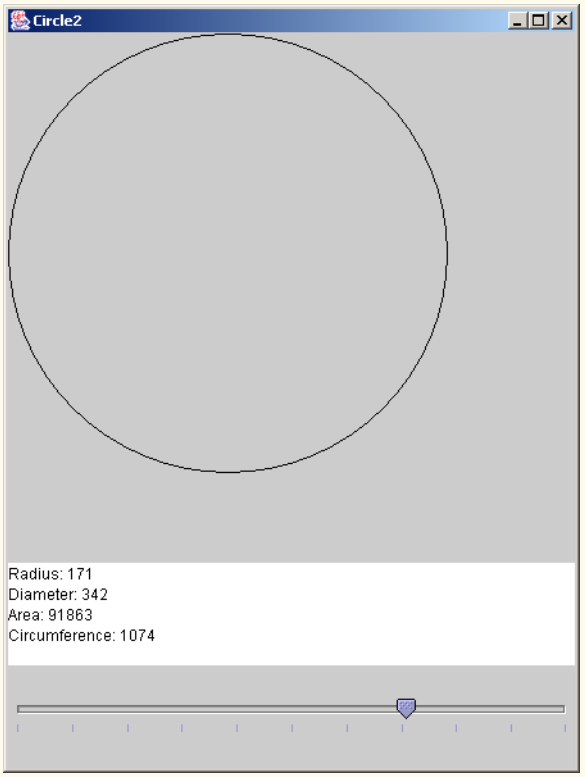
ANS:

```
1  // Exercise 14.8 Solution: Circle2.java
2  // Program draws a circle with radius specified by slider
3  // value and displays the area, diameter and circumference.
```

```
4  import java.awt.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7  import javax.swing.event.*;
8
9  public class Circle2 extends JFrame {
10    private CircleCanvas2 theCanvas;
11    private JTextArea display;
12    private JPanel controlPanel;
13    private JSlider radiusSlider;
14
15    // set up GUI
16    public Circle2()
17    {
18       super( "Circle2" );
19
20       theCanvas = new CircleCanvas2();
21       display = new JTextArea( 5, 30 );
22
23       display.setText( "Radius: " + theCanvas.getRadius() +
24          "\nDiameter: " + theCanvas.getDiameter() + "\nArea: " +
25          theCanvas.getArea() + "\nCircumference: " +
26          theCanvas.getCircumference() );
27
28       radiusSlider =
29          new JSlider( SwingConstants.HORIZONTAL, 100, 200, 150 );
30       radiusSlider.setMajorTickSpacing( 10 );
31       radiusSlider.setPaintTicks( true );
32       radiusSlider.addChangeListener(
33
34          new ChangeListener() { // anonymous inner class
35
36             public void stateChanged( ChangeEvent e )
37             {
38                theCanvas.setRadius( radiusSlider.getValue() );
39                display.setText( "Radius: " + theCanvas.getRadius() +
40                   "\nDiameter: " + theCanvas.getDiameter() + "\nArea: " +
41                   theCanvas.getArea() + "\nCircumference: " +
42                   theCanvas.getCircumference() );
43                repaint();
44             }
45
46          }  // end anonymous inner class
47
48       ); // end call to addChangeListener
49
50       controlPanel = new JPanel();
51       controlPanel.setLayout( new GridLayout( 2, 1 ) );
52       controlPanel.add( display );
53       controlPanel.add( radiusSlider );
54
55       Container container = getContentPane();
56       container.setLayout( new BorderLayout() );
57       container.add( theCanvas, BorderLayout.CENTER );
```

```
58          container.add( controlPanel, BorderLayout.SOUTH );
59
60          setSize( 450, 600 );
61          setVisible( true );
62
63       }  // end constructor
64
65       public static void main( String args[] )
66       {
67          Circle2 application = new Circle2();
68          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
69       }
70
71    }  // end class Circle2
72
73    // class to display circle and calculate circle's dimensions
74    class CircleCanvas2 extends JPanel {
75       private int radius;
76
77       // generate random radius and set panel size
78       public CircleCanvas2()
79       {
80          radius = 150;
81          setSize( 400, 400 );
82       }
83
84       // draw circle
85       public void paintComponent( Graphics g )
86       {
87          g.drawOval( 0, 0, 2 * radius, 2 * radius );
88       }
89
90       // get diameter of circle
91       public int getDiameter()
92       {
93          return ( 2 * radius );
94       }
95
96       // get circumference of circle
97       public int getCircumference()
98       {
99          return ( int )( 2 * Math.PI * radius );
100      }
101
102      // get area of circle
103      public int getArea()
104      {
105         return ( int )( Math.PI * radius * radius );
106      }
107
108      // get radius of circle
109      public int getRadius()
110      {
111         return radius;
112      }
```

```
113
114    // set radius of circle
115    public void setRadius( int r )
116    {
117       radius = r;
118    }
119
120 }  // end class CircleCanvas
```



14.9 Explore the effects of varying the `weightx` and `weighty` values of the program of Fig. 14.19. What happens when a slot has a nonzero weight, but is not allowed to fill the whole area (i.e., the `fill` value is not `BOTH`)?

14.10 Write a program that uses the `paintComponent` method to draw the current value of a `JSlider` on a subclass of `JPanel`. In addition, provide a `JTextField` where a specific value can be entered. The `JTextField` should display the current value of the `JSlider` at all times. A `JLabel` should be used to identify the `JTextField`. The `JSlider` methods `setValue` and `getValue` should be used. [Note: The `setValue` method is a `public` method that does not return a value and takes one integer argument—the `JSlider` value, which determines the position of the thumb.]

ANS:

```
// Exercise 14.10 Solution: DrawValue.java
// Program draws the value of the scrollbar on a canvas.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class DrawValue extends JFrame {
   private DrawCanvas canvas;
   private JTextField display;
   private JLabel label;
   private JSlider slider;
   private JPanel panel;

   public DrawValue()
   {
      super( "DrawValue" );

      // create a panel and set its layout
      panel = new JPanel();
      panel.setLayout( new GridLayout( 1, 3 ) );

      label = new JLabel( "Value:" );

      canvas = new DrawCanvas();

      // set up slider
      slider = new JSlider( SwingConstants.HORIZONTAL, 0, 100, 50 );

      // register JSlider event listener
      slider.addChangeListener(

         new ChangeListener() { // anonymous inner class

            // handle change in slider value
            public void stateChanged( ChangeEvent e )
            {
               canvas.setNumber( slider.getValue() );
               display.setText( String.valueOf( slider.getValue() ) );
            }

         }  // end anonymous inner class

      ); // end call to addChangeListener

      display = new JTextField( "50", 5 );
      display.addActionListener(

         new ActionListener() { // anonymous inner class

            // handle action performed event
            public void actionPerformed( ActionEvent e )
            {
```

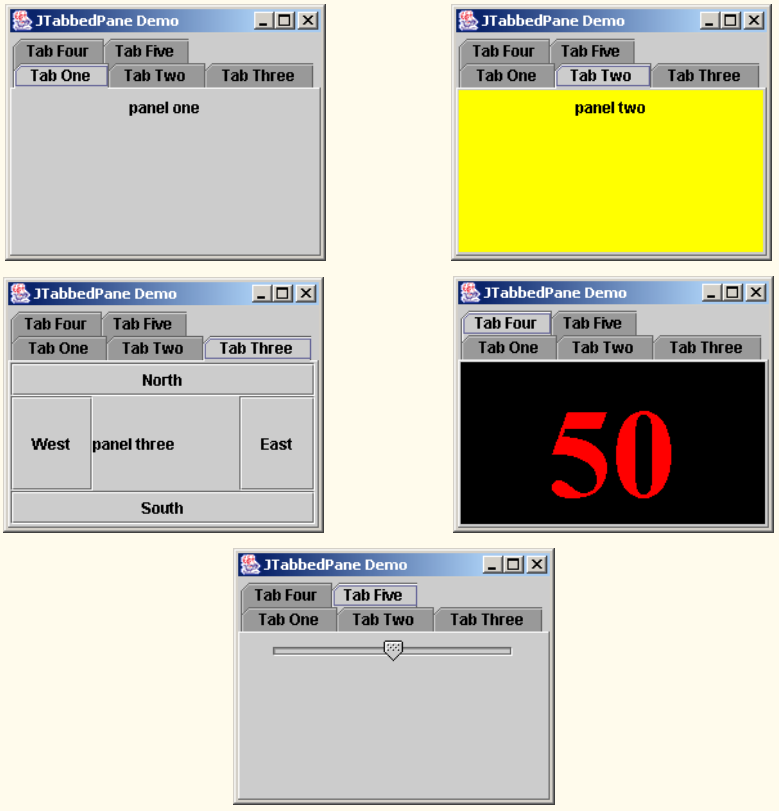
```
54                int value = Integer.parseInt( display.getText() );
55
56                if ( value < slider.getMinimum() ||
57                   value > slider.getMaximum() )
58
59                   return;
60
61                canvas.setNumber( value );
62                slider.setValue( value );
63             }
64
65          }  // end anonymous inner class
66
67       ); // end call to addActionListener
68
69       // add components to panel
70       panel.add( label );
71       panel.add( display );
72       panel.add( slider );
73
74       // add panel to GUI
75       Container container = getContentPane();
76       container.add( canvas, BorderLayout.CENTER );
77       container.add( panel, BorderLayout.NORTH );
78
79       setSize( 300, 300 );
80       setVisible( true );
81
82    }  // end DrawValue constructor
83
84    public static void main( String args[] )
85    {
86       DrawValue application = new DrawValue();
87       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
88    }
89
90 }  // end class DrawValue
91
92 // class to display slider value
93 class DrawCanvas extends JPanel {
94    private int number;
95
96    // set up panel
97    public DrawCanvas()
98    {
99       number = 50;
100      setBackground( Color.black );
101      setSize( 200, 200 );
102   }
103
104   // set number
105   public void setNumber( int theNumber )
106   {
107      number = theNumber;
```

```
108         repaint();
109      }
110
111      // draw number on panel
112      public void paintComponent( Graphics g )
113      {
114         super.paintComponent( g );
115         g.setFont( new Font( "Serif", Font.BOLD, 99 ) );
116         g.setColor( Color.red );
117         g.drawString( String.valueOf( number ),
118            getSize().width / 2 - 40, getSize().height / 2 );
119      }
120
121   }  // end class DrawCanvas
```

14.11 Modify the program of Fig. 14.13 by adding a minimum of two new tabs.
ANS:

```
1  // Exercise 14.11 Solution: JTabbedPaneDemo.java
2  // Demonstrating JTabbedPane.
3  import java.awt.*;
4  import javax.swing.*;
5
6  public class JTabbedPaneDemo extends JFrame  {
7
8     // set up GUI
9     public JTabbedPaneDemo()
10    {
11       super( "JTabbedPane Demo " );
12
13       // create JTabbedPane
14       JTabbedPane tabbedPane = new JTabbedPane();
15
16       // set up pane11 and add it to JTabbedPane
17       JLabel label1 = new JLabel( "panel one", SwingConstants.CENTER );
18       JPanel panel1 = new JPanel();
19       panel1.add( label1 );
```

```
20          tabbedPane.addTab( "Tab One", null, panel1, "First Panel" );
21
22          // set up panel2 and add it to JTabbedPane
23          JLabel label2 = new JLabel( "panel two", SwingConstants.CENTER );
24          JPanel panel2 = new JPanel();
25          panel2.setBackground( Color.yellow );
26          panel2.add( label2 );
27          tabbedPane.addTab( "Tab Two", null, panel2, "Second Panel" );
28
29          // set up panel3 and add it to JTabbedPane
30          JLabel label3 = new JLabel( "panel three" );
31          JPanel panel3 = new JPanel();
32          panel3.setLayout( new BorderLayout() );
33          panel3.add( new JButton( "North" ), BorderLayout.NORTH );
34          panel3.add( new JButton( "West" ), BorderLayout.WEST );
35          panel3.add( new JButton( "East" ), BorderLayout.EAST );
36          panel3.add( new JButton( "South" ), BorderLayout.SOUTH );
37          panel3.add( label3, BorderLayout.CENTER );
38          tabbedPane.addTab( "Tab Three", null, panel3, "Third Panel" );
39
40          // set up panel4 and add it to JTabbedPane
41          JLabel label4 = new JLabel( "panel four" );
42          JPanel panel4 = new JPanel();
43          panel4.setBackground( Color.black );
44          JLabel number = new JLabel( "50" );
45          number.setFont( new Font( "Serif", Font.BOLD, 99 ) );
46          number.setForeground( Color.red );
47          panel4.add( number );
48          tabbedPane.addTab( "Tab Four", null, panel4, "Fourth Panel" );
49
50          // set up panel5 and add it to JTabbedPane
51          JLabel label5 = new JLabel( "panel five" );
52          JPanel panel5 = new JPanel();
53          JSlider slider =
54             new JSlider( SwingConstants.HORIZONTAL, 0, 100, 50 );
55          panel5.add( slider );
56          tabbedPane.addTab( "Tab Five", null, panel5, "Fifth Panel" );
57
58          // add JTabbedPane to container
59          getContentPane().add( tabbedPane );
60
61          setSize( 250, 200 );
62          setVisible( true );
63
64       } // end constructor
65
66       public static void main( String args[] )
67       {
68          JTabbedPaneDemo tabbedPaneDemo = new JTabbedPaneDemo();
69          tabbedPaneDemo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
70       }
71
72    } // end class JTabbedPaneDemo
```



14.12 Declare a subclass of `JPanel` called `MyColorChooser` that provides three `JSlider` objects and three `JTextField` objects. Each `JSlider` represents the values from 0–255 for the red, green and blue parts of a color. Use the red, green and blue values as the arguments to the `Color` constructor to create a new `Color` object. Display the current value of each `JSlider` in the corresponding `JTextField`. When the user changes the value of the `JSlider`, the `JTextField` should be changed accordingly. Declare class `MyColorChooser` so it can be reused in other applications or applets. Use your new GUI component as part of an applet that displays the current `Color` value by drawing a filled rectangle.

ANS:

```
1  // Exercise 14.12 Solution: MyColorChooser.java
2  // JPanel subclass
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class MyColorChooser extends JPanel {
9     private JSlider redSlider, blueSlider, greenSlider;
10    private JTextField redDisplay, blueDisplay, greenDisplay;
11    private JLabel redLabel, blueLabel, greenLabel;
```

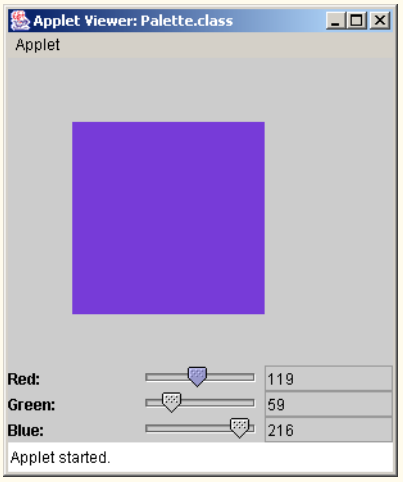
```
12    private Color color;
13
14    // set up GUI
15    public MyColorChooser()
16    {
17       // create sliders and labels
18       redLabel = new JLabel( "Red:" );
19       redSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
20       redDisplay = new JTextField( "0", 4 );
21       redDisplay.setEditable( false );
22
23       greenLabel = new JLabel( "Green:" );
24       greenSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
25       greenDisplay = new JTextField( "0", 4 );
26       greenDisplay.setEditable( false );
27
28       blueLabel = new JLabel( "Blue:" );
29       blueSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
30       blueDisplay = new JTextField( "0", 4 );
31       blueDisplay.setEditable( false );
32
33       setLayout( new GridLayout( 3, 3 ) );
34
35       // add sliders and labels to layout
36       add( redLabel );
37       add( redSlider );
38       add( redDisplay );
39       add( greenLabel );
40       add( greenSlider );
41       add( greenDisplay );
42       add( blueLabel );
43       add( blueSlider );
44       add( blueDisplay );
45
46       // add listeners to Sliders
47       redSlider.addChangeListener( new ChangeHandler() );
48       greenSlider.addChangeListener( new ChangeHandler() );
49       blueSlider.addChangeListener( new ChangeHandler() );
50
51       color = Color.black;
52    }
53
54    // get color
55    public Color getColor()
56    {
57       return color;
58    }
59
60    // get red slider
61    public JSlider getRedSlider()
62    {
63       return redSlider;
64    }
65
```

```
66       // get green slider
67       public JSlider getGreenSlider()
68       {
69          return greenSlider;
70       }
71
72       // get blue slider
73       public JSlider getBlueSlider()
74       {
75          return blueSlider;
76       }
77
78       // inner class to handle slider events
79       private class ChangeHandler implements ChangeListener {
80
81          // handle change in slider value
82          public void stateChanged( ChangeEvent e )
83          {
84             int red = redSlider.getValue();
85             int green = greenSlider.getValue();
86             int blue = blueSlider.getValue();
87
88             color = new Color( red, green, blue );
89
90             redDisplay.setText( String.valueOf( red ) );
91             greenDisplay.setText( String.valueOf( green ) );
92             blueDisplay.setText( String.valueOf( blue ) );
93          }
94
95       }  // end private inner class ChangeHandler
96
97    }  // end class MyColorChooser
```

ANS:

```
1   // Exercise 14.12 Solution: Palette.java
2   // Program allows the user to create a custom color.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6   import javax.swing.event.*;
7
8   public class Palette extends JApplet implements ChangeListener {
9      private MyColorChooser colorChooser;
10     private JPanel drawPanel;
11
12     // set up GUI
13     public void init()
14     {
15        // create a new color chooser
16        colorChooser = new MyColorChooser();
17
```

```
18          // create sliders
19          JSlider red = colorChooser.getRedSlider();
20          red.addChangeListener( this );
21
22          JSlider green = colorChooser.getGreenSlider();
23          green.addChangeListener( this );
24
25          JSlider blue = colorChooser.getBlueSlider();
26          blue.addChangeListener( this );
27
28          drawPanel = new JPanel();
29
30          // add components to GUI
31          Container container = getContentPane();
32          container.add( colorChooser, BorderLayout.SOUTH );
33          container.add( drawPanel, BorderLayout.CENTER );
34
35          this.repaint();
36       }
37
38       public void stateChanged( ChangeEvent event )
39       {
40          draw();
41       }
42
43       // draw directly on JPanel
44       private void draw()
45       {
46          Graphics g = drawPanel.getGraphics();
47          g.setColor( colorChooser.getColor() );
48          g.fillRect( 50, 50, 150, 150 );
49       }
50
51       // draw directly on JPanel
52       public void paint( Graphics gg )
53       {
54          super.paint( gg );
55          draw();
56       }
57
58    }  // end class Palette
```



14.13 Modify the `MyColorChooser` class of Exercise 14.12 to allow the user to enter an integer value into a `JTextField` to set the red, green or blue value. When the user presses *Enter* in the `JTextField`, the corresponding `JSlider` should be set to the appropriate value.

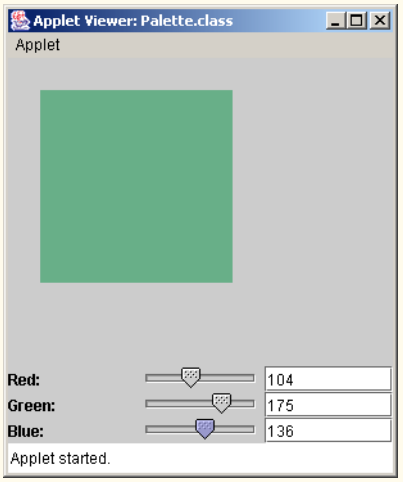
ANS:

```
1  // Exercise 14.13 Solution: MyColorChooser2.java
2  // JPanel subclass
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class MyColorChooser2 extends JPanel {
9     private JSlider redSlider, blueSlider, greenSlider;
10    private JTextField redDisplay, blueDisplay, greenDisplay;
11    private JLabel redLabel, blueLabel, greenLabel;
12    private Color color;
13
14    // set up GUI
15    public MyColorChooser2()
16    {
17       redLabel = new JLabel( "Red:" );
18       redSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
19       redDisplay = new JTextField( "0", 4 );
20
21       greenLabel = new JLabel( "Green:" );
22       greenSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
23       greenDisplay = new JTextField( "0", 4 );
24
25       blueLabel = new JLabel( "Blue:" );
26       blueSlider = new JSlider( SwingConstants.HORIZONTAL, 0, 255, 1 );
27       blueDisplay = new JTextField( "0", 4 );
28
```

```
29          setLayout( new GridLayout( 3, 3 ) );
30
31          add( redLabel );
32          add( redSlider );
33          add( redDisplay );
34          add( greenLabel );
35          add( greenSlider );
36          add( greenDisplay );
37          add( blueLabel );
38          add( blueSlider );
39          add( blueDisplay );
40
41          redSlider.addChangeListener( new ChangeHandler() );
42          greenSlider.addChangeListener( new ChangeHandler() );
43          blueSlider.addChangeListener( new ChangeHandler() );
44
45          redDisplay.addActionListener( new ActionHandler() );
46          greenDisplay.addActionListener( new ActionHandler() );
47          blueDisplay.addActionListener( new ActionHandler() );
48
49          color = Color.black;
50       }
51
52       // set slider and text field values
53       public void setColor( Color c )
54       {
55          color = c;
56
57          redSlider.setValue( c.getRed() );
58          redDisplay.setText( String.valueOf( c.getRed() ) );
59
60          greenSlider.setValue( c.getGreen() );
61          greenDisplay.setText( String.valueOf( c.getGreen() ) );
62
63          blueSlider.setValue( c.getBlue() );
64          blueDisplay.setText( String.valueOf( c.getBlue() ) );
65       }
66
67       // get color
68       public Color getColor()
69       {
70          return color;
71       }
72
73       // get red slider
74       public JSlider getRedSlider()
75       {
76          return redSlider;
77       }
78
79       // get green slider
80       public JSlider getGreenSlider()
81       {
82          return greenSlider;
83       }
```

```
84
85       // get blue slider
86       public JSlider getBlueSlider()
87       {
88          return blueSlider;
89       }
90
91       // get red text field
92       public JTextField getRedDisplay()
93       {
94          return redDisplay;
95       }
96
97       // get green text field
98       public JTextField getGreenDisplay()
99       {
100         return greenDisplay;
101      }
102
103      // get blue text field
104      public JTextField getBlueDisplay()
105      {
106         return blueDisplay;
107      }
108
109      // inner class to handle slider events
110      private class ChangeHandler implements ChangeListener {
111
112         // handle change in slider value
113         public void stateChanged( ChangeEvent event )
114         {
115            int red = redSlider.getValue();
116            int green = greenSlider.getValue();
117            int blue = blueSlider.getValue();
118
119            color = new Color( red, green, blue );
120
121            redDisplay.setText( String.valueOf( red ) );
122            greenDisplay.setText( String.valueOf( green ) );
123            blueDisplay.setText( String.valueOf( blue ) );
124         }
125
126      }  // end private inner class ChangeHandler
127
128      // inner class to handle action events
129      private class ActionHandler implements ActionListener {
130
131         // handle text field input
132         public void actionPerformed( ActionEvent event )
133         {
134            int red = Integer.parseInt( redDisplay.getText() );
135            int green = Integer.parseInt( greenDisplay.getText() );
136            int blue = Integer.parseInt( blueDisplay.getText() );
137
```

```
            color = new Color( red, green, blue );

            redSlider.setValue( red );
            greenSlider.setValue( green );
            blueSlider.setValue( blue );
         }

      }  // end private inner class ActionHandler

}  // end class MyColorChooser2
```



14.14 Modify the applet of Exercise 14.13 to draw the current color as a rectangle on an instance of a subclass of `JPanel` called `DrawPanel`. Class `DrawPanel` should provide its own `paintComponent` method to draw the rectangle and should provide *set* methods to set the red, green and blue values for the current color. When any *set* method is invoked for the class `DrawPanel`, the object should automatically `repaint` itself.

ANS:

```
// Exercise 14.14 Solution: DrawPanel2.java
// Definition of class DrawPanel
import java.awt.*;
import javax.swing.*;

public class DrawPanel2 extends JPanel {
   private Color color;
   private int red, green, blue;

   public DrawPanel2( Color c )
   {
      color = c;

      red = color.getRed();
      green = color.getGreen();
      blue = color.getBlue();
```

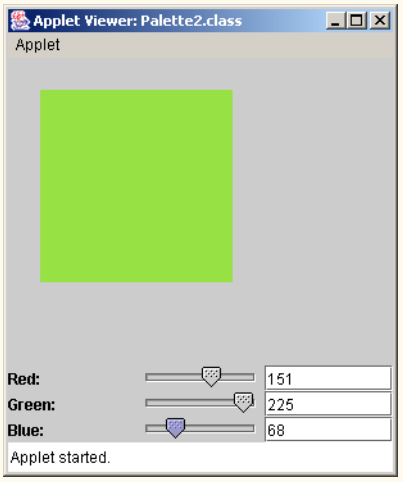
```
17      }
18
19      // draw colored rectangle
20      public void paintComponent( Graphics g )
21      {
22         super.paintComponent( g );
23
24         g.setColor( color );
25         g.fillRect( 25, 25, 150, 150 );
26      }
27
28      // set red
29      public void setRed( int r )
30      {
31         red = r;
32         changeColor();
33      }
34
35      // set green
36      public void setGreen( int g )
37      {
38         green = g;
39         changeColor();
40      }
41
42      // set blue
43      public void setBlue( int b )
44      {
45         blue = b;
46         changeColor();
47      }
48
49      // set color and repaint
50      private void changeColor()
51      {
52         color = new Color( red, green, blue );
53         repaint();
54      }
55
56   }  // end class DrawPanel2
```

ANS:

```
1   // Exercise 14.14 Solution: Palette2.java
2   // Program allows the user to create a custom color.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6   import javax.swing.event.*;
7
8   public class Palette2 extends JApplet {
9      private MyColorChooser3 colorChooser;
10     private DrawPanel2 drawPanel;
```

```
11
12    // set up GUI
13    public void init()
14    {
15       colorChooser = new MyColorChooser3();
16
17       final JSlider red = colorChooser.getRedSlider();
18       red.addChangeListener(
19
20          new ChangeListener() { // anonymous inner class
21
22             // handle slider event
23             public void stateChanged( ChangeEvent event )
24             {
25                drawPanel.setRed( red.getValue() );
26             }
27
28          }  // end anonymous inner class
29
30       ); // end call to addChangeListener
31
32       final JTextField redField = colorChooser.getRedDisplay();
33       redField.addActionListener(
34
35          new ActionListener() { // anonymous inner class
36
37             // handle text field event
38             public void actionPerformed( ActionEvent event )
39             {
40                drawPanel.setRed(
41                   Integer.parseInt( redField.getText() ) );
42             }
43
44          }  // end anonymous inner class
45
46       ); // end call to addActionListener
47
48       final JSlider green = colorChooser.getGreenSlider();
49       green.addChangeListener(
50
51          new ChangeListener() { // anonymous inner class
52
53             // handle slider event
54             public void stateChanged( ChangeEvent event )
55             {
56                drawPanel.setGreen( green.getValue() );
57             }
58
59          }  // end anonymous inner class
60
61       ); // end call to addChangeListener
62
63       final JTextField greenField = colorChooser.getGreenDisplay();
64       greenField.addActionListener(
```

```
65
66          new ActionListener() { // anonymous inner class
67
68             // handle text field event
69             public void actionPerformed( ActionEvent event )
70             {
71                drawPanel.setGreen(
72                   Integer.parseInt( greenField.getText() ) );
73             }
74
75          }  // end anonymous inner class
76
77       ); // end call to addActionListener
78
79       final JSlider blue = colorChooser.getBlueSlider();
80       blue.addChangeListener(
81
82          new ChangeListener() { // anonymous inner class
83
84             // handle slider event
85             public void stateChanged( ChangeEvent event )
86             {
87                drawPanel.setBlue( blue.getValue() );
88             }
89
90          }  // end anonymous inner class
91
92       ); // end call to addChangeListener
93
94       final JTextField blueField = colorChooser.getBlueDisplay();
95       blueField.addActionListener(
96
97          new ActionListener() { // anonymous inner class
98
99             // handle text field event
100            public void actionPerformed( ActionEvent event )
101            {
102               drawPanel.setBlue(
103                  Integer.parseInt( blueField.getText() ) );
104            }
105
106         }  // end anonymous inner class
107
108      ); // end call to addActionListener
109
110      drawPanel = new DrawPanel2( Color.black );
111
112      Container container = getContentPane();
113      container.add( colorChooser, BorderLayout.SOUTH );
114      container.add( drawPanel, BorderLayout.CENTER );
115   }
116
117 }  // end class Palette2
```



14.15 Modify the applet of Exercise 14.14 to allow the user to drag the mouse across the `DrawPanel` to draw a shape in the current color. Enable the user to choose what shape to draw.

ANS:

```
1  // Exercise 14.15 Solution: MyShape.java
2  // Abstract MyShape class.
3  import java.awt.*;
4  import java.io.*;
5
6  public abstract class MyShape implements Serializable {
7     private int x1, y1, x2, y2;
8     private Color color;
9
10    public MyShape()
11    {
12       this( 0, 0, 0, 0 );
13    }
14
15    public MyShape( int x1, int y1, int x2, int y2 )
16    {
17       this( x1 ,y1, x2, y2, Color.black );
18    }
19
20    public MyShape( int x1, int y1, int x2, int y2, Color c )
21    {
22       setAll( x1, y1, x2, y2, c );
23    }
24
25    // change all the properties at once after the shape has been created
26    public final void setAll( int x1, int y1, int x2, int y2, Color c )
27    {
28       setX1( x1 );
29       setY1( y1 );
30       setX2( x2 );
```

```
31          setY2( y2 );
32          setColor( c );
33       }
34
35       public final void setX1( int x )
36       {
37          x1 = x;
38       }
39
40       public final int getX1()
41       {
42          return x1;
43       }
44
45       public final void setY1( int y )
46       {
47          y1 = y;
48       }
49
50       public final int getY1()
51       {
52          return y1;
53       }
54
55       public final void setX2( int x )
56       {
57          x2 = x;
58       }
59
60       public final int getX2()
61       {
62          return x2;
63       }
64
65       public final void setY2( int y )
66       {
67          y2 = y;
68       }
69
70       public final int getY2()
71       {
72          return y2;
73       }
74
75       public void setColor( Color c )
76       {
77          color = c;
78       }
79
80       public Color getColor()
81       {
82          return color;
83       }
84
```

```
85    public abstract void draw( Graphics g );
86 }
```

ANS:

```
1  // Exercise 14.15 Solution: MyLine.java
2  // MyLine class.
3  import java.awt.*;
4  public class MyLine extends MyShape {
5     public MyLine()
6     {
7        super();
8     }
9
10    public MyLine( int x1, int y1, int x2, int y2 )
11    {
12       super( x1, y1, x2, y2 );
13    }
14
15    public MyLine( int x1, int y1, int x2, int y2, Color c )
16    {
17       super( x1, y1, x2, y2, c );
18    }
19
20    public void draw( Graphics g )
21    {
22       Color c = g.getColor();  // get previous color
23       g.setColor( this.getColor() );
24       g.drawLine( getX1(), getY1(), getX2(), getY2() );
25       g.setColor( c );         // reset previous color
26    }
27
28    public String toString()
29    {
30       return "Line";
31    }
32 }
```

ANS:

```
1  // Exercise 14.15 Solution: MyBounded.java
2  // Abstract MyBounded class.
3  import java.awt.*;
4
5  public abstract class MyBounded extends MyShape {
6     private boolean filled;
7
8     public MyBounded()
9     {
10       super();
11       setFilled( false );
12    }
```

```
13
14    public MyBounded( int x1, int y1, int x2, int y2, boolean f )
15    {
16       super( x1, y1, x2, y2 );
17       setFilled( f );
18    }
19
20    public MyBounded( int x1, int y1, int x2, int y2, Color c, boolean f )
21    {
22       super( x1, y1, x2, y2, c );
23       setFilled( f );
24    }
25
26    public final void setAll( int x1, int y1, int x2, int y2, Color c,
27       boolean f )
28    {
29       super.setAll( x1, y1, x2, y2, c );
30       setFilled( f );
31    }
32
33    public void setFilled( boolean f )
34    {
35       filled = f;
36    }
37
38    public boolean getFilled()
39    {
40       return filled;
41    }
42
43    public int getUpperLeftX()
44    {
45       return Math.min( getX1(), getX2() );
46    }
47
48    public int getUpperLeftY()
49    {
50       return Math.min( getY1(), getY2() );
51    }
52
53    public int getWidth()
54    {
55       return Math.abs( getX1() - getX2() );
56    }
57
58    public int getHeight()
59    {
60       return Math.abs( getY1() - getY2() );
61    }
62 }
```

ANS:

```
1  // Exercise 14.15 Solution: MyOval.java
2  // MyOval class.
3  import java.awt.*;
4
5  public class MyOval extends MyBounded {
6     public MyOval()
7     {
8        super();
9     }
10
11    public MyOval( int x1, int y1, int x2, int y2, boolean f )
12    {
13       super( x1, y1, x2, y2, f );
14    }
15
16    public MyOval( int x1, int y1, int x2, int y2, Color c, boolean f )
17    {
18       super( x1, y1, x2, y2, c, f );
19    }
20
21    public void draw( Graphics g )
22    {
23       Color c = g.getColor();
24       g.setColor( this.getColor() );
25       if ( getFilled() )
26          g.fillOval( getUpperLeftX(), getUpperLeftY(),
27                      getWidth(), getHeight() );
28       else
29          g.drawOval( getUpperLeftX(), getUpperLeftY(),
30                      getWidth(), getHeight() );
31       g.setColor( c );
32    }
33
34    public String toString()
35    {
36       return "Oval";
37    }
38 }
```

ANS:

```
1  // Exercise 14.15 Solution: MyRectangle.java
2  // MyRectangle class.
3  import java.awt.*;
4
5  public class MyRectangle extends MyBounded {
6     public MyRectangle()
7     {
8        super();
9     }

10
11    public MyRectangle( int x1, int y1, int x2, int y2, boolean f )
```

```
   {
      super( x1, y1, x2, y2, f );
   }

   public MyRectangle( int x1, int y1, int x2, int y2, Color c,
      boolean f )
   {
      super( x1, y1, x2, y2, c, f );
   }

   public void draw( Graphics g )
   {
      Color c = g.getColor();
      g.setColor( this.getColor() );
      if ( getFilled() )
         g.fillRect( getUpperLeftX(), getUpperLeftY(),
                     getWidth(), getHeight() );
      else
         g.drawRect( getUpperLeftX(), getUpperLeftY(),
                     getWidth(), getHeight() );
      g.setColor( c );
   }

   public String toString()
   {
      return "Rectangle";
   }
}
```

ANS:

```
// Exercise 14.15 Solution: ShapePanel.java
// This class allows the user to choose a shape and fill setting.
import javax.swing.*;

public class ShapePanel extends JPanel {

   // one of these objects will be used to draw the shape
   private static final String[] shapeList =
      { "Rectangle", "Line", "Oval" };
   private JComboBox choose;
   private JCheckBox fill;

   public ShapePanel()
   {
      choose = new JComboBox( shapeList );
      fill = new JCheckBox( "Filled", false );
      add( choose );
      add( fill );
   }


   public JCheckBox getFill()
   {
```

```
23          return fill;
24       }
25
26       public JComboBox getChooser()
27       {
28          return choose;
29       }
30
31    }
```

ANS:

```
1   // Exercise 14.15 Solution: DrawPanel3.java
2   // This panel draws the shape.
3   import java.awt.*;
4   import javax.swing.*;
5
6   public class DrawPanel3 extends JPanel {
7      private MyShape shape;
8      private boolean fill;
9      private int red, green, blue;
10
11     public DrawPanel3(MyShape s)
12     {
13        shape = s;
14     }
15
16     public DrawPanel3()
17     {
18        shape = null;
19     }
20
21     // draw colored rectangle
22     public void paintComponent( Graphics g )
23     {
24        super.paintComponent( g );
25
26        if ( shape != null )
27           shape.draw( g );
28     }
29
30     // handle a mouse click event
31     public void createShape( int x, int y, String type )
32     {
33        Color color = new Color( red, green, blue );
34
35        if ( type.equals( "Rectangle" ) )
36           shape = new MyRectangle( x, y, x, y, color, fill );
37        else if ( type.equals( "Line" ) )
38           shape = new MyLine( x, y, x, y, color );

39        else
40           shape = new MyOval( x, y, x, y, color, fill );
41
```

```
42          repaint();
43       }
44
45       // handle a mouse movement
46       public void resizeShape( int x, int y )
47       {
48          // change the x2 and y2 values
49          shape.setX2( x );
50          shape.setY2( y );
51          repaint();
52       }
53
54       // set the fill property of the shape
55       public void setFill( boolean f ) {
56
57          // MyLine objects do not have fill variable so we keep track here
58          fill = f;
59
60          // only bounded shapes have a fill variable
61          if ( shape instanceof MyBounded )
62             ((MyBounded)shape).setFilled( f );
63          repaint();
64       }
65
66       // change the shape that is drawn; the type of the shape passed in
67       // determines which shape will be drawn
68       public void setShape( String type ) {
69
70          // get the information from the old shape
71          int x1 = shape.getX1();
72          int y1 = shape.getY1();
73          int x2 = shape.getX2();
74          int y2 = shape.getY2();
75          Color color = new Color( red, green, blue );
76
77          if ( type.equals( "Rectangle" ) )
78             shape = new MyRectangle( x1, y1, x2, y2, color, fill );
79          else if ( type.equals( "Line" ) )
80             shape = new MyLine( x1, y1, x2, y2, color );
81          else
82             shape = new MyOval( x1, y1, x2, y2, color, fill );
83
84          repaint();
85       }
86
87       // set red
88       public void setRed( int r )
89       {
90          red = r;
91          shape.setColor( new Color( red, green, blue ) );
92          repaint();
93       }
94
95       // set green
96       public void setGreen( int g )
```

```
97       {
98          green = g;
99          shape.setColor( new Color( red, green, blue ) );
100         repaint();
101      }
102
103      // set blue
104      public void setBlue( int b )
105      {
106         blue = b;
107         shape.setColor( new Color( red, green, blue ) );
108         repaint();
109      }
110
111   }  // end class DrawPanel3
```

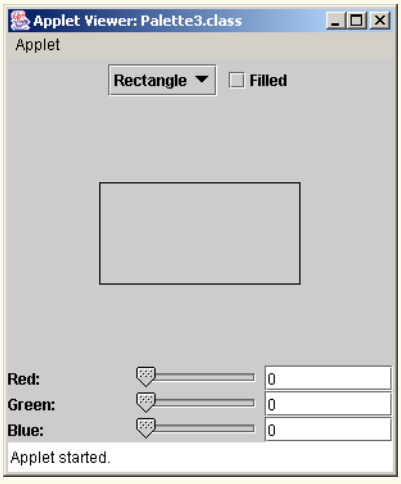
ANS:

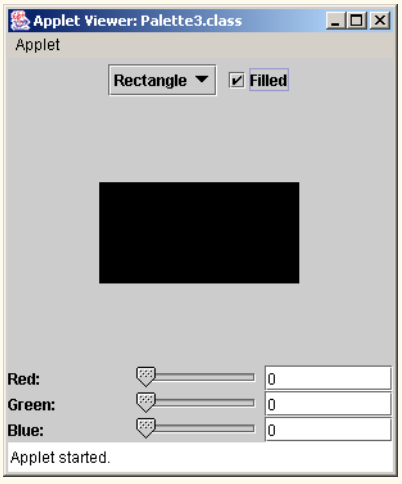
```
1   // Exercise 14.15 Solution: Palette3.java
2   // Program allows the user to draw a shape.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6   import javax.swing.event.*;
7
8   public class Palette3 extends JApplet {
9      private MyColorChooser4 colorChooser;
10     private DrawPanel3 drawPanel;
11     private ShapePanel shapePanel;
12     private boolean filled;
13     private JComboBox choose;
14
15     // set up GUI
16     public void init()
17     {
18        colorChooser = new MyColorChooser4();
19
20        final JSlider red = colorChooser.getRedSlider();
21        red.addChangeListener(
22
23           new ChangeListener() { // anonymous inner class
24
25              // handle slider event
26              public void stateChanged( ChangeEvent event )
27              {
28                 drawPanel.setRed( red.getValue() );
29              }
30
31           }  // end anonymous inner class
32
33       ); // end call to addChangeListener
34
35       final JTextField redField = colorChooser.getRedDisplay();
```

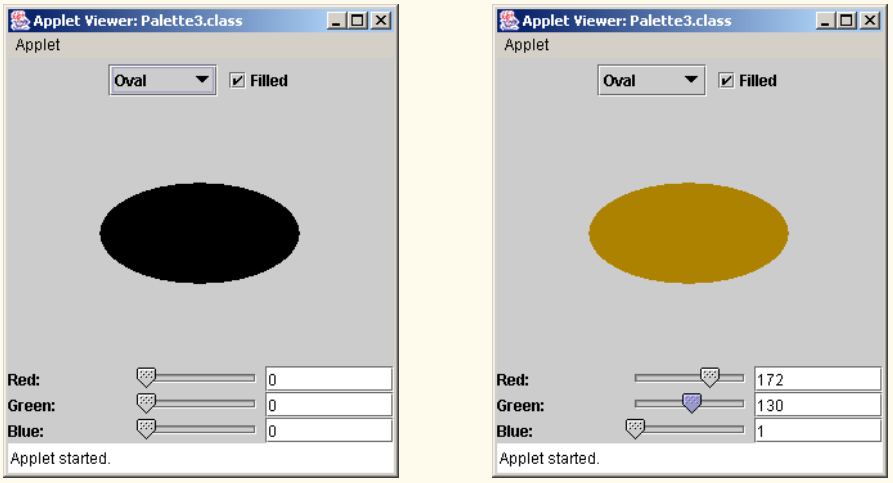
```
36         redField.addActionListener(
37
38            new ActionListener() { // anonymous inner class
39
40               // handle text field event
41               public void actionPerformed( ActionEvent event )
42               {
43                  drawPanel.setRed(
44                     Integer.parseInt( redField.getText() ) );
45               }
46
47            }  // end anonymous inner class
48
49         ); // end call to addActionListener
50
51         final JSlider green = colorChooser.getGreenSlider();
52         green.addChangeListener(
53
54            new ChangeListener() { // anonymous inner class
55
56               // handle slider event
57               public void stateChanged( ChangeEvent event )
58               {
59                  drawPanel.setGreen( green.getValue() );
60               }
61
62            }  // end anonymous inner class
63
64         ); // end call to addChangeListener
65
66         final JTextField greenField = colorChooser.getGreenDisplay();
67         greenField.addActionListener(
68
69            new ActionListener() { // anonymous inner class
70
71               // handle text field event
72               public void actionPerformed( ActionEvent event )
73               {
74                  drawPanel.setGreen(
75                     Integer.parseInt( greenField.getText() ) );
76               }
77
78            }  // end anonymous inner class
79
80         ); // end call to addActionListener
81
82         final JSlider blue = colorChooser.getBlueSlider();
83         blue.addChangeListener(
84
85            new ChangeListener() { // anonymous inner class
86
87               // handle slider event
88               public void stateChanged( ChangeEvent event )
89               {
```

```
90                drawPanel.setBlue( blue.getValue() );
91             }
92
93          }  // end anonymous inner class
94
95       ); // end call to addChangeListener
96
97       final JTextField blueField = colorChooser.getBlueDisplay();
98       blueField.addActionListener(
99
100         new ActionListener() { // anonymous inner class
101
102            // handle text field event
103            public void actionPerformed( ActionEvent event )
104            {
105               drawPanel.setBlue(
106                  Integer.parseInt( blueField.getText() ) );
107            }
108
109         }  // end anonymous inner class
110
111      ); // end call to addActionListener
112
113      drawPanel =
114         new DrawPanel3( new MyRectangle( 25, 25, 150, 150, false ) );
115      drawPanel.addMouseListener(
116
117         new MouseAdapter() { // anonymous inner class
118
119            public void mousePressed( MouseEvent event ) {
120               drawPanel.createShape( event.getX(), event.getY(),
121                  ( String )choose.getSelectedItem() );
122            }
123
124         } // end anonymous inner class
125
126      ); // end call to addMouseListener
127
128      drawPanel.addMouseMotionListener(
129
130         new MouseMotionAdapter() { // anonymous inner class
131
132            public void mouseDragged( MouseEvent event ) {
133               drawPanel.resizeShape( event.getX(), event.getY() );
134            }
135
136         } // end anonymous inner class
137
138      ); // end call to addMouseMotionListener
139
140      shapePanel = new ShapePanel();

141      JCheckBox fill = shapePanel.getFill();
142      fill.addItemListener(
143
```

```
144          new ItemListener() { // anonymous inner class
145
146             public void itemStateChanged( ItemEvent event ) {
147                if ( event.getStateChange() == ItemEvent.SELECTED )
148                   drawPanel.setFill( true );
149                else
150                   drawPanel.setFill( false );
151             }
152
153          } // end anonymous inner class
154
155       ); // end call to addItemListener
156
157       choose = shapePanel.getChooser();
158       choose.addItemListener(
159
160          new ItemListener() { // anonymous inner class
161
162             public void itemStateChanged( ItemEvent event ) {
163                drawPanel.setShape( ( String )choose.getSelectedItem() );
164             }
165
166          } // end anonymous inner class
167
168       ); // end call to addItemListener
169
170       Container container = getContentPane();
171       container.add( colorChooser, BorderLayout.SOUTH );
172       container.add( drawPanel, BorderLayout.CENTER );
173       container.add( shapePanel, BorderLayout.NORTH );
174    }
175
176 }  // end class Palette3
```







14.16 Modify the program of Exercise 14.15 to enable the program to run as an application. The existing applet's code should be modified only by adding a `main` method to launch the application in its own `JFrame`. Provide the user with the ability to terminate the application by clicking the close box on the window that is displayed and by selecting `Exit` from a `File` menu. Use the techniques shown in Fig. 14.9.

ANS:

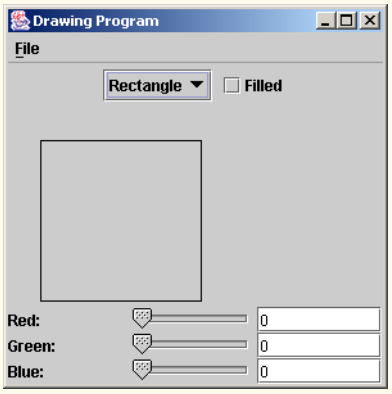
```
1  // Exercise 14.16 Solution: Palette3.java
2  // This class manages GUI components and interacts with the drawing panel.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Palette3 extends JFrame {
9     private DrawPanel3 drawPanel;
10    private ShapePanel shapePanel;
11    private boolean filled;
12    private JComboBox choose;
13
14    public Palette3()
15    {
16       super( "Drawing Program" );
17
18       MyColorChooser4 colorChooser = new MyColorChooser4();
19
20       final JSlider red = colorChooser.getRedSlider();
21       red.addChangeListener(
22
23          new ChangeListener() { // anonymous inner class
24
25             // handle slider event
26             public void stateChanged( ChangeEvent event )
27             {
```

```
28                drawPanel.setRed( red.getValue() );
29             }
30
31          }  // end anonymous inner class
32
33       ); // end call to addChangeListener
34
35       final JTextField redField = colorChooser.getRedDisplay();
36       redField.addActionListener(
37
38          new ActionListener() { // anonymous inner class
39
40             // handle text field event
41             public void actionPerformed( ActionEvent event )
42             {
43                drawPanel.setRed(
44                   Integer.parseInt( redField.getText() ) );
45             }
46
47          }  // end anonymous inner class
48
49       ); // end call to addActionListener
50
51       final JSlider green = colorChooser.getGreenSlider();
52       green.addChangeListener(
53
54          new ChangeListener() { // anonymous inner class
55
56             // handle slider event
57             public void stateChanged( ChangeEvent event )
58             {
59                drawPanel.setGreen( green.getValue() );
60             }
61
62          }  // end anonymous inner class
63
64       ); // end call to addChangeListener
65
66       final JTextField greenField = colorChooser.getGreenDisplay();
67       greenField.addActionListener(
68
69          new ActionListener() { // anonymous inner class
70
71             // handle text field event
72             public void actionPerformed( ActionEvent event )
73             {
74                drawPanel.setGreen(
75                   Integer.parseInt( greenField.getText() ) );
76             }
77
78          }  // end anonymous inner class
79
80       ); // end call to addActionListener
81
```

```
82          final JSlider blue = colorChooser.getBlueSlider();
83          blue.addChangeListener(
84
85             new ChangeListener() { // anonymous inner class
86
87                // handle slider event
88                public void stateChanged( ChangeEvent event )
89                {
90                   drawPanel.setBlue( blue.getValue() );
91                }
92
93             }  // end anonymous inner class
94
95          ); // end call to addChangeListener
96
97          final JTextField blueField = colorChooser.getBlueDisplay();
98          blueField.addActionListener(
99
100            new ActionListener() { // anonymous inner class
101
102               // handle text field event
103               public void actionPerformed( ActionEvent event )
104               {
105                  drawPanel.setBlue(
106                     Integer.parseInt( blueField.getText() ) );
107               }
108
109            }  // end anonymous inner class
110
111         ); // end call to addActionListener
112
113         drawPanel =
114            new DrawPanel3( new MyRectangle( 25, 25, 150, 150, false ) );
115         drawPanel.addMouseListener(
116
117            new MouseAdapter() { // anonymous inner class
118
119               public void mousePressed( MouseEvent event )
120               {
121                  drawPanel.createShape( event.getX(), event.getY(),
122                     ( String )choose.getSelectedItem() );
123               }
124
125            } // end anonymous inner class
126
127         ); // end call to addMouseListener
128
129         drawPanel.addMouseMotionListener(
130
131            new MouseMotionAdapter() { // anonymous inner class
132
133               public void mouseDragged( MouseEvent event )
134               {
135                  drawPanel.resizeShape( event.getX(), event.getY() );
136               }
```

```
137
138          } // end anonymous inner class
139
140       ); // end call to addMouseMotionListener
141
142       shapePanel = new ShapePanel();
143       JCheckBox fill = shapePanel.getFill();
144       fill.addItemListener(
145
146          new ItemListener() { // anonymous inner class
147
148             public void itemStateChanged( ItemEvent event )
149             {
150                if ( event.getStateChange() == ItemEvent.SELECTED )
151                   drawPanel.setFill( true );
152                else
153                   drawPanel.setFill( false );
154             }
155
156          } // end anonymous inner class
157
158       ); // end call to addItemListener
159
160       choose = shapePanel.getChooser();
161       choose.addItemListener(
162
163          new ItemListener() { // anonymous inner class
164
165             public void itemStateChanged( ItemEvent event )
166             {
167                drawPanel.setShape( ( String )choose.getSelectedItem() );
168             }
169
170          } // end anonymous inner class
171
172       ); // end call to addItemListener
173
174       Container container = getContentPane();
175       container.add( colorChooser, BorderLayout.SOUTH );
176       container.add( drawPanel, BorderLayout.CENTER );
177       container.add( shapePanel, BorderLayout.NORTH );
178
179       JMenuBar menuBar = new JMenuBar();
180       JMenu fileMenu = new JMenu( "File" );
181       fileMenu.setMnemonic( 'f' );
182       JMenuItem exitItem = new JMenuItem( "Exit" );
183       exitItem.setMnemonic( 'e' );
184       fileMenu.add( exitItem );
185       menuBar.add( fileMenu );
186       exitItem.addActionListener(
187
188          new ActionListener() { // anonymous inner class
189
190             public void actionPerformed( ActionEvent event )
191             {
```

```
192                System.exit( 0 );
193             }
194
195          } // end anonymous inner class
196
197       ); // end call to addActionListener
198
199       setJMenuBar( menuBar );
200
201       setSize( 300, 300 );
202       setVisible( true );
203    }
204
205    public static void main( String args[] )
206    {
207       Palette3 application = new Palette3();
208       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
209    }
210
211 }  // end class Palette3
```



14.17 *(Complete Drawing Application)* Using the techniques developed in Exercise 13.27–Exercise 13.31 and Exercise 14.12–Exercise 14.16, create a complete drawing program that can execute as both an applet and an application. The program should use the GUI components of Chapter 13 and Chapter 14 to enable the user to select the shape, color and fill characteristics. Each shape should be stored in an array of `MyShape` objects, where `MyShape` is the superclass in your hierarchy of shape classes (see Exercise 10.9 and Exercise 10.10). Use a `JDesktopPane` and `JInternalFrames` to allow the user to create multiple separate drawings in separate child windows. Create the user interface as a separate child window containing all the GUI components that allow the user to determine the characteristics of the shape to be drawn. The user can then click in any `JInternalFrame` to draw the shape.

ANS:

```
1  // Exercise 14.17 Solution: DrawFrame.java
2  // This class creates a frame that allows the user to draw shapes.
3  import java.awt.*;
```

```
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class DrawFrame extends JInternalFrame {
9     private DrawPanel3 drawPanel;
10    private JComboBox choose;
11    private DrawProgram parent;
12    private String name;
13
14    // set up GUI
15    public DrawFrame( String title, DrawProgram caller ) {
16       super( title, true, true, true, true );
17
18       parent = caller;
19       name = title;
20
21       MyColorChooser4 colorChooser = new MyColorChooser4();
22
23       // when the red slider is altered, update the shape
24       final JSlider red = colorChooser.getRedSlider();
25       red.addChangeListener(
26
27          new ChangeListener() { // anonymous inner class
28
29             // handle slider event
30             public void stateChanged( ChangeEvent event )
31             {
32                drawPanel.setRed( red.getValue() );
33             }
34
35          }  // end anonymous inner class
36
37       ); // end call to addChangeListener
38
39       // when the red text field is altered, update the shape
40       final JTextField redField = colorChooser.getRedDisplay();
41       redField.addActionListener(
42
43          new ActionListener() { // anonymous inner class
44
45             // handle text field event
46             public void actionPerformed( ActionEvent event )
47             {
48                drawPanel.setRed(
49                   Integer.parseInt( redField.getText() ) );
50             }
51
52          }  // end anonymous inner class
53
54       ); // end call to addActionListener
55
56       // when the green slider is altered, update the shape
57       final JSlider green = colorChooser.getGreenSlider();
```

```
58          green.addChangeListener(
59
60             new ChangeListener() { // anonymous inner class
61
62                // handle slider event
63                public void stateChanged( ChangeEvent event )
64                {
65                   drawPanel.setGreen( green.getValue() );
66                }
67
68             }  // end anonymous inner class
69
70          ); // end call to addChangeListener
71
72          // when the green text field is altered, update the shape
73          final JTextField greenField = colorChooser.getGreenDisplay();
74          greenField.addActionListener(
75
76             new ActionListener() { // anonymous inner class
77
78                // handle text field event
79                public void actionPerformed( ActionEvent event )
80                {
81                   drawPanel.setGreen(
82                      Integer.parseInt( greenField.getText() ) );
83                }
84
85             }  // end anonymous inner class
86
87          ); // end call to addActionListener
88
89          // when the blue slider is altered, update the shape
90          final JSlider blue = colorChooser.getBlueSlider();
91          blue.addChangeListener(
92
93             new ChangeListener() { // anonymous inner class
94
95                // handle slider event
96                public void stateChanged( ChangeEvent event )
97                {
98                   drawPanel.setBlue( blue.getValue() );
99                }
100
101            }  // end anonymous inner class
102
103         ); // end call to addChangeListener
104
105         // when the blue text field is altered, update the shape
106         final JTextField blueField = colorChooser.getBlueDisplay();
107         blueField.addActionListener(
108
109            new ActionListener() { // anonymous inner class
110
```

```
111             // handle text field event
112             public void actionPerformed( ActionEvent event )
113             {
114                drawPanel.setBlue(
115                   Integer.parseInt( blueField.getText() ) );
116             }
117
118          }  // end anonymous inner class
119
120       ); // end call to addActionListener
121
122       drawPanel = new DrawPanel3();
123       drawPanel.addMouseListener(
124
125          new MouseAdapter() { // anonymous inner class
126
127             // create the chosen shape
128             public void mousePressed( MouseEvent event ) {
129                   drawPanel.createShape( event.getX(), event.getY() );
130             }
131
132             // finish creating the shape
133             public void mouseReleased( MouseEvent event ) {
134                   drawPanel.finishShape();
135             }
136
137          } // end anonymous inner class
138
139       ); // end call to addMouseListener
140
141       drawPanel.addMouseMotionListener(
142
143          new MouseMotionAdapter() { // anonymous inner class
144
145             // resize the shape
146             public void mouseDragged( MouseEvent event ) {
147                   drawPanel.resizeShape( event.getX(), event.getY() );
148             }
149
150          } // end anonymous inner class
151
152       ); // end call to addMouseMotionListener
153
154       ShapePanel shapePanel = new ShapePanel();
155       JCheckBox fill = shapePanel.getFill();
156       fill.addItemListener(
157
158          new ItemListener() { // anonymous inner class
159
160             // toggle the fill field in the drawing panel
161             public void itemStateChanged( ItemEvent event ) {
162                if ( event.getStateChange() == ItemEvent.SELECTED )
163                   drawPanel.setFill( true );
164                else
165                   drawPanel.setFill( false );
```

```
            }

         } // end anonymous inner class

      ); // end call to addItemListener

      choose = shapePanel.getChooser();
      choose.addItemListener(

         new ItemListener() { // anonymous inner class

            // change the shape that is displayed
            public void itemStateChanged( ItemEvent event ) {
               drawPanel.setShape( ( String )choose.getSelectedItem() );
            }

         } // end anonymous inner class

      ); // end call to addItemListener

      Container container = getContentPane();
      container.setLayout( new BorderLayout() );
      container.add( shapePanel, BorderLayout.NORTH );
      container.add( drawPanel, BorderLayout.CENTER );
      container.add( colorChooser, BorderLayout.SOUTH );

      addInternalFrameListener(

         new InternalFrameAdapter() { // anonymous inner class

            // when the frame is closed, alert the DrawProgram
            public void internalFrameClosed( InternalFrameEvent e )
            {
               parent.frameClosed( name );
            }
         } // end anonymous inner class

      ); // end call to addInternalFrameListener

      setSize( 300, 300 );

   } // end constructor

} // end class DrawFrame
```

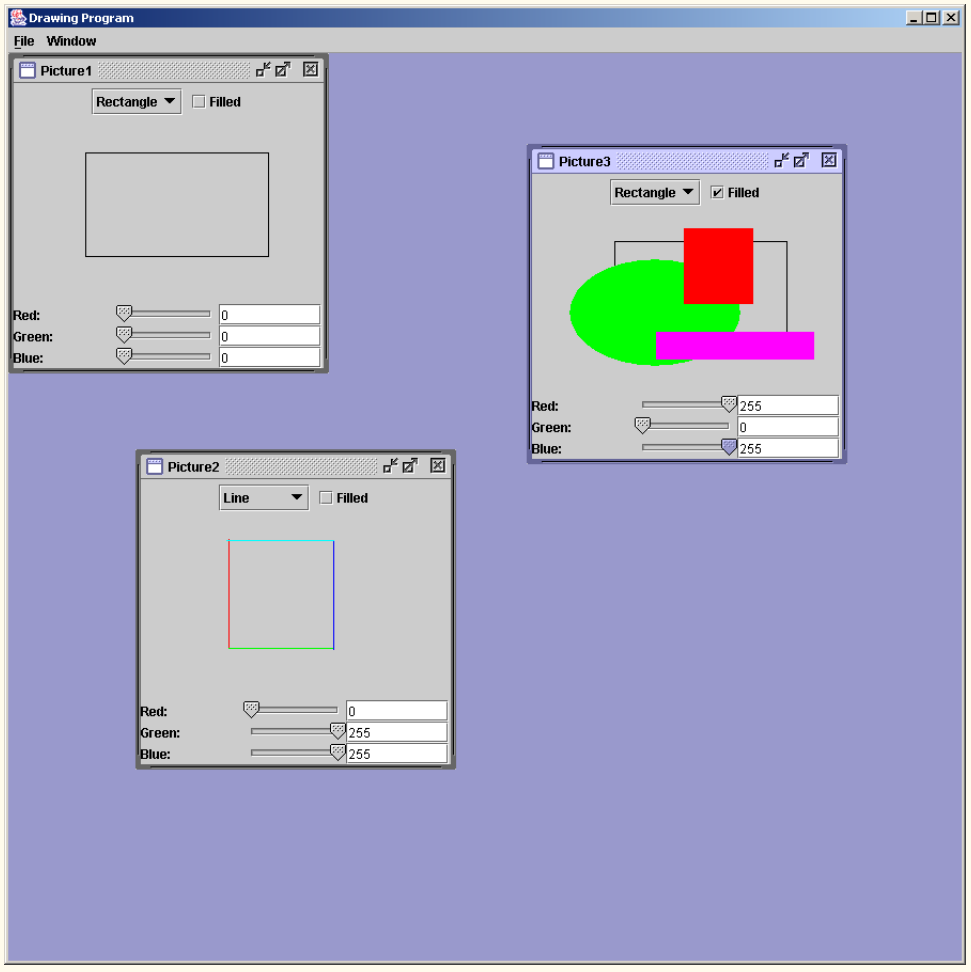
ANS:

```
// Exercise 14.17 Solution: DrawProgram.java
// This class creates the desktop, manages menu items and creates frames.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
```

```
 7
 8  public class DrawProgram extends JFrame {
 9     private static final int MAX_WINDOWS = 20;
10     private JInternalFrame[] frames;
11     private JMenuItem[] frameItems;
12     private int count = 0;
13     private JDesktopPane desktop;
14     private JMenuItem newItem;
15     private JMenu windowMenu;
16
17     public DrawProgram()
18     {
19        super( "Drawing Program" );
20
21        desktop = new JDesktopPane();
22        getContentPane().add( desktop );
23
24        frames = new JInternalFrame[ MAX_WINDOWS ];
25        frameItems = new JMenuItem[ MAX_WINDOWS ];
26
27        JMenuBar menuBar = new JMenuBar();
28        JMenu fileMenu = new JMenu( "File" );
29        fileMenu.setMnemonic( 'f' );
30
31        // create a new frame
32        newItem = new JMenuItem( "New" );
33        newItem.setMnemonic( 'n' );
34        fileMenu.add( newItem );
35        newItem.addActionListener(
36
37           new ActionListener() { // anonymous inner class
38
39              public void actionPerformed( ActionEvent event ) {
40
41                 // the title of the window
42                 String title = "Picture" + ( count + 1 );
43
44                 // create the frame
45                 frames[ count ] = new DrawFrame( title, DrawProgram.this );
46                 desktop.add( frames[ count ] );
47                 frames[ count ].setVisible( true );
48
49                 // create the menu item
50                 frameItems[ count ] = new JMenuItem( title );
51                 windowMenu.add( frameItems[ count ] );
52                 frameItems[ count ].addActionListener(
53
54                    new ActionListener() { // anonymous inner class
55
56                       public void actionPerformed( ActionEvent event ) {
57                          String caller = event.getActionCommand();
58                          int number =
59                             Integer.parseInt( caller.substring( 7 ) );
60
```

```
61                         // set the selected frame to the front
62                         frames[ number - 1 ].toFront();
63                      }
64
65                   } // end anonymous inner class
66
67                ); // end call to addActionListener
68
69                // increment the count
70                count++;
71
72                // if we hit the maximum number of windows,
73                // disable the new menu item
74                if ( count == MAX_WINDOWS )
75                   newItem.setEnabled( false );
76
77             }
78
79          } // end anonymous inner class
80
81       ); // end call to addActionListener
82
83       // exit the application
84       JMenuItem exitItem = new JMenuItem( "Exit" );
85       exitItem.setMnemonic( 'e' );
86       fileMenu.add( exitItem );
87       menuBar.add( fileMenu );
88       exitItem.addActionListener(
89
90          new ActionListener() { // anonymous inner class
91
92             public void actionPerformed( ActionEvent event ) {
93                System.exit( 0 );
94             }
95
96          } // end anonymous inner class
97
98       ); // end call to addActionListener
99
100      // this will bring the selected window to the front
101      windowMenu = new JMenu( "Window" );
102      menuBar.add( windowMenu );
103
104      setJMenuBar( menuBar );
105
106      setSize( 900, 900 ); // set the window's size
107      setVisible( true );
108
109   } // end constructor
110
111   // called when a child frame is closed
112   public void frameClosed( String window )
113   {
114      int number = Integer.parseInt( window.substring( 7 ) );
```

```
115
116        // remove the menu item and set the reference to null
117        windowMenu.remove( frameItems[ number - 1 ] );
118        frameItems[ number - 1 ] = null;
119     }
120
121     public static void main( String args[] )
122     {
123        DrawProgram application = new DrawProgram();
124        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
125     }
126
127  } // end class DrawProgram
```



15

Exception Handling

Objectives

- To understand exception and error handling.
- To use `try`, `throw` and `catch` to detect, indicate and handle exceptions, respectively.
- To use the `finally` clause to release resources.
- To understand the Java exception hierarchy.
- To declare new exception classes.
- To create chained exceptions.

It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all, try something.
Franklin Delano Roosevelt

O! throw away the worser part of it,
And live the purer with the other half.
William Shakespeare

If they're running and they don't look where they're going I have to come out from somewhere and catch them.
Jerome David Salinger

And oftentimes excusing of a fault
Doth make the fault the worse by the excuse.
William Shakespeare

I never forget a face, but in your case I'll make an exception.
Groucho (Julius Henry) Marx

SELF-REVIEW EXERCISES

15.1 List five common examples of exceptions.

ANS: Memory exhaustion, array index out of bounds, arithmetic overflow, division by zero, invalid method parameters.

15.2 Give several reasons why exception-handling techniques should not be used for conventional program control.

ANS: (a) Exception handling is designed to handle infrequently occurring situations that often result in program termination, so compiler writers are not required to implement exception handling to perform optimally. (b) Flow of control with conventional control structures is generally clearer and more efficient than with exceptions. (c) Problems can occur because the stack is unwound when an exception occurs and resources allocated prior to the exception may not be freed. (d) The "additional" exceptions can get in the way of genuine error-type exceptions. It becomes more difficult for the programmer to keep track of the larger number of exception cases.

15.3 Why are exceptions particularly appropriate for dealing with errors produced by methods of classes in the Java API?

ANS: It is unlikely that methods of classes in the Java API could perform error processing that would meet the unique needs of all users.

15.4 What is a "resource leak?"

ANS: A "resource leak" occurs when an executing program does not properly release a resource when the resource is no longer needed. If the program attempts to use the resource again in the future, the program may not be able to access the resource.

15.5 If no exceptions are thrown in a `try` block, where does control proceed to when the `try` block completes execution?

ANS: The `catch` clauses for that `try` statement are skipped, and the program resumes execution after the last `catch` clause. If there is a `finally` clause, it is executed first then the program resumes execution after the `finally` clause.

15.6 Give a key advantage of using `catch( Exception e )`.

ANS: The form `catch( Exception e )` catches any type of exception thrown in a `try` statement. An advantage is that no thrown `Exception` can slip by without at least being caught. The programmer can then decide to handle the exception or possibly rethrow the exception.

15.7 Should a conventional applet or application catch `Error` objects?

ANS: `Error`s are usually serious problems with the underlying Java system; most programs will not want to catch `Error`s because the program will not be able to recover from such problems.

15.8 What happens if no catch handler matches the type of a thrown object?

ANS: This causes the search for a match to continue in the next enclosing `try` statement.

15.9 What happens if several handlers match the type of the thrown object?

ANS: The first matching `catch` clause after the `try` block is executed.

15.10 Why would a programmer specify a superclass type as the type in a `catch` handler?

ANS: This enables a program to catch related types of exceptions and process them in a uniform manner. However, it is often useful to process the subclass types individually for more precise exception handling.

15.11 What is the key reason for using `finally` clauses?

ANS: The `finally` clause is the preferred means for preventing resource leaks.

15.12 What happens when a `catch` handler throws an `Exception`?
ANS: The exception will be processed by a `catch` handler (if one exists) associated with an enclosing `try` block (if one exists).

15.13 What does the statement `throw` *exceptionReference* do?
ANS: It rethrows the exception for processing by an exception handler of an enclosing `try` block.

15.14 What happens to a local reference in a `try` block when that block throws an `Exception`?
ANS: The reference goes out of scope, and the reference count for the object is decremented. If the reference count becomes zero, the object is marked for garbage collection.

EXERCISES

15.15 List the various exceptional conditions that have occurred in programs throughout this text. List as many additional exceptional conditions as you can. For each of these, describe briefly how a program typically would handle the exception by using the exception-handling techniques discussed in this chapter. Some typical exceptions are division by zero, arithmetic overflow, array index out of bounds, etc.

ANS: A few examples are: **Division by zero** - check the denominator to see if it is equal to zero, and throw an exception before the division occurs. **Array subscript out of bounds** - catch the exception and print an error message telling the user what index was trying to be referenced, and exit the program in a controlled manner. **Bad cast** - catch the exception and either cast it to the proper type if that can be determined, or print an error message indicating what the bad cast was, and exit the program.

15.16 Until this chapter, we have found that dealing with errors detected by constructors is a bit awkward. Explain why exception handling is an effective means for dealing with constructor failure.

ANS: A thrown exception passes to the outside world the information about the failed constructor and the responsibility to deal with the failure. Exceptions thrown in constructors cause objects built as part of the object being constructed to be marked for eventual garbage collection.

15.17 Suppose a program throws an exception and the appropriate exception handler begins executing. Now suppose that the exception handler itself throws the same exception. Does this create an infinite recursion? Explain your answer.

ANS: No. The previous exception is lost and the most recent exception is thrown by the `catch` block.

15.18 Use inheritance to create an exception superclass and various exception subclasses. Write a program to demonstrate that the `catch` specifying the superclass catches subclass exceptions.

ANS:

```
1  // Exercise 15.18 Solution: Demo.java
2  // Program demonstrates that the exception
3  // superclass will catch the subclass exceptions
4
5  // exception subclasses
6  class ExceptionA extends Exception {}
7
8  class ExceptionB extends ExceptionA {}
9
10 class ExceptionC extends ExceptionB {}
11
```

```
12  public class Demo {
13
14     public static void main( String args[] )
15     {
16        // throw ExceptionC
17        try {
18           throw new ExceptionC();
19        }
20
21        // catch ExceptionA and all subclasses
22        catch( ExceptionA exception1 ) {
23           System.err.println( "First Exception subclass caught. \n" );
24        }
25
26        // throw ExceptionB
27        try {
28           throw new ExceptionB();
29        }
30
31        // catch ExceptionA and all subclasses
32        catch( ExceptionA exception2 ) {
33           System.err.println( "Second Exception subclass caught. \n" );
34        }
35
36     } // end method main
37
38  } // end class Demo
```

```
First Exception subclass caught.

Second Exception subclass caught.
```

15.19 Write a program that demonstrates how various exceptions are caught with

```
catch ( Exception exception )
```

ANS:

```
1  // Exercise 15.19 Solution: Demo2.java
2  // Program demonstrates catching Exception e
3  import java.io.*;
4
5  public class Demo2 {
6
7     // execute application
8     public static void main( String args[] )
9     {
10       try {
11          throw new ExceptionA();
12       }
13
```

```
14         catch( Exception exception ) {
15            System.out.println( exception.toString() );
16         }
17
18         try {
19            throw new ExceptionB();
20         }
21
22         catch( Exception exception ) {
23            System.out.println( exception.toString() );
24         }
25
26         try {
27            throw new NullPointerException();
28         }
29
30         catch( Exception exception ) {
31            System.out.println( exception.toString() );
32         }
33
34         try {
35            throw new IOException();
36         }
37
38         catch( Exception exception ) {
39            System.out.println( exception.toString() );
40         }
41
42      } // end method main
43
44   } // end class Demo2
45
46   // subclass of Exception
47   class ExceptionA extends Exception {}
48
49   // subclass of ExceptionA
50   class ExceptionB extends ExceptionA {}
```

```
ExceptionA
ExceptionB
java.lang.NullPointerException
java.io.IOException
```

15.20 Write a program that shows that the order of `catch` clauses is important. If you try to catch a superclass exception type before a subclass type, the compiler should generate errors. Explain why these errors occur.

ANS:

```
1  // Exercise 15.20 part II Solution: CompileError.java
2  // Program generates a compiler error.
3  import java.io.*;
```

```
4
5  public class CompileError {
6
7     public static void main( String args[] )
8     {
9        try {
10          throw new IOException();
11       }
12
13       // superclass exception
14       catch ( Exception exception ) {
15          exception.printStackTrace();
16       }
17
18       // subclass exception
19       catch ( IOException ioException ) {
20          System.err.println( "IOException" );
21       }
22    }
23
24 }  // end class CompileError
```

```
CompileError.java:19: exception java.io.IOException has already been caught
      catch ( IOException ioException ) {
      ^
1 error
```

15.21 Write a program that shows a constructor passing information about constructor failure to an exception handler.

ANS:

```
1  // Exercise 15.21 Solution: Demo3.java
2  // Program demonstrates a constructor that throws an exception.
3
4  class SomeException {
5
6     // constructor throws exception
7     public SomeException() throws Exception
8     {
9        throw new Exception();
10    }
11
12 }  // end class SomeException
13
14 public class Demo3 {
15
16    public static void main( String args[] )
17    {
18       SomeException testException;
19
20       // instantiate SomeException object
21       try {
```

```
22             testException = new SomeException();
23          }
24
25          // catch Exception
26          catch( Exception exception ) {
27             System.out.println( exception.toString() );
28          }
29       }
30
31    }  // end class Demo3
```

```
java.lang.Exception
```

15.22 Write a program that illustrates rethrowing an exception.
ANS:

```
1   // Exercise 15.22 Solution: Demo4.java
2   // Program demonstrates rethrowing an exception
3
4   public class Demo4 {
5
6      public static void main( String args[] )
7      {
8         // call someMethod
9         try {
10           someMethod();
11        }
12
13        // catch Exceptions thrown from someMethod
14        catch( Exception exception ) {
15            System.err.println( exception.getMessage() + "\n" );
16            exception.printStackTrace();
17        }
18     }
19
20     // call someMethod2; rethrow Exceptions back to main
21     public static void someMethod() throws Exception
22     {
23        // call someMethod2
24        try {
25           someMethod2();
26        }
27
28        // catch Exceptions thrown from someMethod2
29        catch( Exception exception2 ) {
30           throw exception2;   // rethrow the Exception
31        }
32     }
33
34     // throw Exception back to someMethod
35     public static void someMethod2() throws Exception
36     {
```

```
37         throw new Exception( "Exception thrown in someMethod2" );
38      }
39
40   } // end class Demo4
```

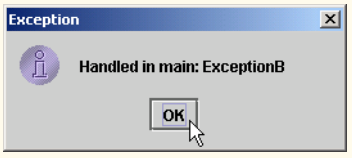
```
Exception thrown in someMethod2

java.lang.Exception: Exception thrown in someMethod2
        at Demo4.someMethod2(Demo4.java:37)
        at Demo4.someMethod(Demo4.java:25)
        at Demo4.main(Demo4.java:10)
```

15.23 Write a program that shows that a method with its own `try` block does not have to catch every possible error generated within the `try`. Some exceptions can slip through to, and be handled in, other scopes.

ANS:

```
1   // Exercise 15.23 Solution: Demo5.java
2   // Program demonstrates rethrowing an exception.
3   import javax.swing.*;
4
5   public class Demo5 {
6
7      public static void main( String args[] )
8      {
9         // call method someMethod
10        try {
11           someMethod();
12        }
13
14        // catch Exception
15        catch( ExceptionB exception ) {
16           JOptionPane.showMessageDialog( null,
17              "Handled in main: " + exception, "Exception",
18              JOptionPane.INFORMATION_MESSAGE );
19        }
20
21        // catch other exceptions
22        catch( Exception exception ) {
23           exception.printStackTrace();
24           System.exit( 1 );
25        }
26
27        System.exit( 0 );
28
29     } // end method main
30
31     // call method someMethod2
32     public static void someMethod() throws Exception
33     {
34        try {
```

```
35             someMethod2();
36          }
37
38          // doesn't catch Exception, just catches ExceptionA
39          catch( ExceptionA exception ) {
40             JOptionPane.showMessageDialog( null,
41                "Handled execeptionA in someMethod: " + exception,
42                "Exception", JOptionPane.INFORMATION_MESSAGE );
43          }
44       }
45
46       // throw Exception
47       public static void someMethod2() throws Exception
48       {
49          throw new ExceptionB();
50       }
51
52    }  // end class Demo5
53
54    // subclass of Exception
55    class ExceptionA extends Exception {}
56    class ExceptionB extends Exception {}
```



16

Multithreading

Objectives

- To understand multithreaded programming.
- To appreciate how multithreading can improve program performance.
- To understand the life cycle of a thread.
- To understand thread priorities and scheduling.
- To understand how to create, manage and destroy threads.
- To understand thread synchronization.
- To understand daemon threads.
- To be able to stop and suspend threads.
- To be able to display output from multiple threads in a Swing GUI.

The spider's touch, how exquisitely fine!
Feels at each thread, and lives along the line.
Alexander Pope

A person with one watch knows what time it is; a person with two watches is never sure.
Proverb

Learn to labor and to wait.
Henry Wadsworth Longfellow

The most general definition of beauty…Multeity in Unity.
Samuel Taylor Coleridge

SELF-REVIEW EXERCISES

16.1 Fill in the blanks in each of the following statements:

a) C and C++ are _____-threaded languages whereas Java is a _____-threaded language.

ANS: single, multi.

b) Three reasons an alive thread might not be *Runnable* are that it is _____, _____ and _____.

ANS: *Waiting*, *Sleeping*, *Blocked* for input/output.

c) A thread enters the *Dead* state when _____.

ANS: its `run` method terminates.

d) A thread's priority can be changed with method _____.

ANS: `setPriority`

e) A thread may give up the processor to a thread of the same priority by calling `Thread` method _____.

ANS: `yield`

f) To pause for a designated number of milliseconds and resume execution, a thread should call method _____.

ANS: `sleep`

g) Method _____ moves a single thread in an object's *Waiting* state to the *Ready* state.

ANS: `notify`

h) Method _____ moves every thread in an object's *Waiting* state to the *Ready* state.

ANS: `notifyAll`

16.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) A thread is not *Runnable* if it is dead.

ANS: True

b) In Java, a higher priority *Runnable* thread should preempt threads of lower priority.

ANS: True

c) Some operating systems use timeslicing with threads. Therefore, they can enable threads to preempt threads of the same priority.

ANS: False. Timeslicing allows a thread to execute until its timeslice (or quantum) expires. Then other threads of equal priority can execute.

d) Threads may `yield` to threads of lower priority.

ANS: False. Threads can only yield to threads of equal priority.

EXERCISES

16.3 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Method `sleep` does not consume processor time while a thread sleeps.

ANS: True.

b) Declaring a method `synchronized` guarantees that deadlock cannot occur.

ANS: False. Deadlocks can occur if the lock on an object is never released.

c) `Thread` methods `suspend`, `resume` and `stop` are deprecated.

ANS: True.

16.4 Define each of the following terms.

a) thread

ANS: An individual execution context of a program.

b) multithreading

ANS: The ability of more than one thread to execute concurrently.

c) *Ready* state

ANS: A state in which the thread is capable of running (if the processor becomes available).

d) *Blocked* state

ANS: A state in which the thread cannot use the processor. For example, the *blocked* state occurs when the thread issues an I/O request.

e) preemptive scheduling

ANS: A thread of higher priority enters a *running* state and is assigned the processor. The thread "preempted" from the processor is placed back in the *ready* state according to its priority.

f) `Runnable` interface

ANS: An interface that provides a `run` method. By implementing the `Runnable` interface, any class can be executed as a separate thread.

g) monitor

ANS: A monitor "watches" shared data between threads. A monitor is responsible for locking an object (i.e., allowing only one thread at a time to execute `synchronized` methods on the object).

h) `notify` method

ANS: Notifys a waiting thread that an object's lock has been released and that the waiting thread can now attempt to obtain the lock for itself.

i) producer/consumer relationship

ANS: A relationship in which a producer and a consumer share common data. The producer typically wants to "produce" (add information) and the consumer wants to "eat" (remove information).

16.5 List each the reasons given in the text for entering the *Blocked* state. For each of these, describe how the program will normally leave the *Blocked* state and enter the *Ready* state.

16.6 What is timeslicing? Give a fundamental difference in how scheduling is performed on Java systems that support timeslicing vs. scheduling on Java systems that do not support timeslicing. Why would a thread ever want to call `yield`?

ANS: Timeslicing specifies that each thread can use the processor for a limited amount of time. When a thread's timeslice expires, a thread of equal priority gets a chance to execute. Systems that do not support timeslicing will not preempt a thread with the same priority. The waiting thread cannot execute until the thread in the processor has completed its task or removes itself from the processor.

16.7 Distinguish among each of the following means of pausing threads:

a) sleep

ANS: A thread sleeps for a specified number of milliseconds and then enters the ready state.

b) blocking I/O

ANS: A thread is suspended until a pending I/O operation is completed.

16.8 Discuss each of the following terms in the context of Java's threading mechanisms:

a) monitor

ANS: Determines when the producer can produce and when the consumer can consume.

b) producer

ANS: A thread that writes data to a shared memory resource.

c) consumer

ANS: A thread that reads data from a shared memory resource.

d) `wait`

ANS: Places a thread in the wait state until it is notified the lock has been released.

e) `notify`

ANS: Notify the waiting thread that the lock is being released.

f) `InterruptedException`

ANS: The `wait` method is capable of throwing an `InterruptedException`.

g) `synchronized`

ANS: When a method is declared `synchronized` and it is running the object is locked. Other threads cannot access the other `synchronized` methods of the object until the lock is released.

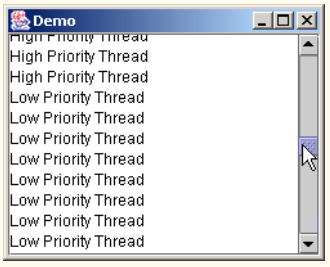
16.9 Write a Java program to demonstrate that as a high-priority thread executes, it will delay the execution of all lower priority threads.

ANS:

```
1  // Exercise 16.9 Solution: Demo.java
2  // Program priority of threads.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Demo extends JFrame {
8     private HighThread high;
9     private LowThread low;
10    private JTextArea output;
11
12    // Demo constructor
13    public Demo()
14    {
15       super( "Demo" );
16
17       // create GUI
18       output = new JTextArea( 10, 20 );
19       JScrollPane scrollPane = new JScrollPane( output );
20       Container container = getContentPane();
21       container.add( scrollPane );
22       setSize( 250, 200 );
23       setVisibility( true );
24
25       // initialize threads
26       low = new LowThread( output );
27       high = new HighThread( output );
28
29       // start threads
30       low.start();
31       high.start();
32    }
33
34    public static void main( String args[] )
35    {
36       Demo application = new Demo();
37       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
38    }
39
40 }  // end class Demo
41
42 // subclass of Thread
43 class HighThread extends Thread {
44    private JTextArea display;
```

```
45
46     // HighThread constructor
47     public HighThread( JTextArea textArea )
48     {
49        display = textArea;
50        setPriority( Thread.MAX_PRIORITY );
51     }
52
53     // action to perform on execution
54     public void run()
55     {
56        for ( int x = 0; x < 100; x++ )
57           SwingUtilities.invokeLater(
58
59              new Runnable() { // anonymous inner class
60
61                 public void run()
62                 {
63                    display.append( "High Priority Thread\n" );
64                 }
65
66              } // end anonymous inner class
67
68           ); // end call to invokeLater
69
70     } // end method run
71
72  }  // end class HighThread
73
74  // subclass of Thread
75  class LowThread extends Thread {
76     private JTextArea display;
77
78     // LowThread constructor
79     public LowThread( JTextArea textArea )
80     {
81        display = textArea;
82        setPriority( Thread.MIN_PRIORITY );
83     }
84
85     // action to perform on execution
86     public void run()
87     {
88        for ( int y = 0; y < 100; y++ )
89           SwingUtilities.invokeLater(
90
91              new Runnable() { // anonymous inner class
92
93                 public void run()
94                 {
95                    display.append( "Low Priority Thread\n" );
96                 }
97
98              } // end anonymous inner class
```

```
99
100          ); // end call to invokeLater
101
102    } // end method run
103
104 }  // end class LowThread
```



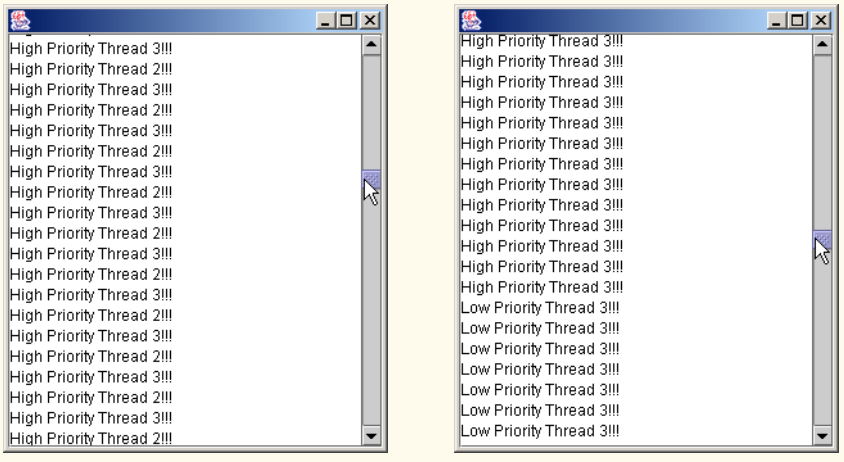
16.10 If your system supports timeslicing, write a Java program that demonstrates timeslicing among several equal-priority threads. Show that a lower priority thread's execution is deferred by the timeslicing of the higher-priority threads.

ANS:

```
1  // Exercise 16.10 Solution: Demo2.java
2  // Program demonstrates priority threads and time slicing.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Demo2 extends JFrame {
8     private HighThread high[];
9     private LowThread low[];
10    private JTextArea output;
11
12    public Demo2()
13    {
14       output = new JTextArea( 10, 20 );
15       getContentPane().add( new JScrollPane( output ) );
16
17       high = new HighThread[ 3 ];
18       low = new LowThread[ 3 ];
19
20       setSize( 300, 350 );
21       setVisible( true );
22    }
23
24    // start Threads
25    public void start()
26    {
27       // instantiate and start Threads
28       for ( int i = 0; i < high.length; i++ ) {
29
```

```
30            // LowThreads
31            if ( low[ i ] == null ) {
32               low[ i ] = new LowThread( output, i + 1 );
33               low[ i ].start();
34            }
35
36            // HighThreads
37            if ( high[ i ] == null ) {
38               high[ i ] = new HighThread( output, i + 1 );
39               high[ i ].start();
40            }
41         }
42      }
43
44      public static void main( String args[] )
45      {
46         Demo2 application = new Demo2();
47         application.start();
48         application.setDefaultCloseOperation( EXIT_ON_CLOSE );
49      }
50
51   }  // end class Demo2
52
53   // high priority Thread
54   class HighThread extends Thread {
55      private JTextArea display;
56      private int number;
57
58      public HighThread( JTextArea textArea, int n )
59      {
60         display = textArea;
61         number = n;
62         setPriority( Thread.MAX_PRIORITY );
63      }
64
65      // run Thread
66      public void run()
67      {
68         for ( int x = 1; x <= 300; x++ )
69            SwingUtilities.invokeLater(
70
71               new Runnable() { // anonymous inner class
72
73                  public void run() {
74                     display.append(
75                        "High Priority Thread " + number + "!!!\n" );
76                  }
77
78               } // end anonymous inner class
79
80            ); // end call to invokeLater
81
82      } // end method run
83
84   }  // end class HighThread
```

```
85
86  // low priority Thread
87  class LowThread extends Thread {
88     private JTextArea display;
89     private int number;
90
91     public LowThread( JTextArea textArea, int n )
92     {
93        display = textArea;
94        number = n;
95        setPriority( Thread.MIN_PRIORITY );
96     }
97
98     // run Thread
99     public void run()
100    {
101       for ( int x = 1; x <= 300; x++ )
102          SwingUtilities.invokeLater(
103
104             new Runnable() { // anonymous inner class
105
106                public void run() {
107                   display.append(
108                      "Low Priority Thread " + number + "!!!\n" );
109                }
110
111             } // end anonymous inner class
112
113          ); // end call to invokeLater
114
115    } // end method run
116
117 }  // end class LowThread
```



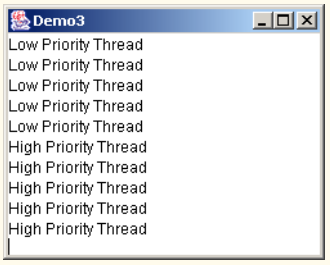
16.11 Write a Java program that demonstrates a high priority thread using `sleep` to give lower priority threads a chance to run.

ANS:

```
1  // Exercise 16.11 Solution: Demo3.java
2  // Program demonstrates high priority threads
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Demo3 extends JFrame {
8     private HighThread high;
9     private LowThread low;
10    private JTextArea output;
11
12    public Demo3()
13    {
14       super( "Demo3" );
15
16       // set up GUI
17       output = new JTextArea( 10, 20 );
18       getContentPane().add( output );
19       setSize( 250, 200 );
20       setVisible( true );
21
22       // initialize threads
23       high = new HighThread( output );
24       low = new LowThread( output );
25
26       // start threads
27       high.start();
28       low.start();
29    }
30
31    public static void main( String args[] )
32    {
33       Demo3 application = new Demo3();
34       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
35    }
36
37 } // end class Demo3
38
39 // subclass of Thread
40 class HighThread extends Thread {
41    private JTextArea display;
42
43    public HighThread( JTextArea textArea )
44    {
45       display = textArea;
46       setPriority( Thread.MAX_PRIORITY );
47    }
48
49    // action to perform on execution
50    public void run()
51    {
52       for ( int x = 0; x < 5; x++ ) {
```

```
53          try {
54             sleep( 100 );
55          }
56
57          // process exception from sleep
58          catch ( Exception exception ) {
59             JOptionPane.showMessageDialog( null, exception.toString(),
60                "Exception", JOptionPane.ERROR_MESSAGE );
61          }
62
63          SwingUtilities.invokeLater(
64
65             new Runnable() { // anonymous inner class
66
67                public void run()
68                {
69                   display.append( "High Priority Thread\n" );
70                }
71
72             } // end anonymous inner class
73
74          ); // end call to invokeLater
75
76       } // end for
77
78    } // end method run
79
80 }  // end class HighThread
81
82 // subclass of Thread
83 class LowThread extends Thread {
84    private JTextArea display;
85
86    public LowThread( JTextArea textArea )
87    {
88       display = textArea;
89       setPriority( Thread.MIN_PRIORITY );
90    }
91
92    // action to perform on execution
93    public void run()
94    {
95       for ( int y = 0; y < 5; y++ )
96          SwingUtilities.invokeLater(
97
98             new Runnable() { // anonymous inner class
99
100               public void run()
101               {
102                  display.append( "Low Priority Thread\n" );
103               }
104
105            } // end anonymous inner class
106
107         ); // end call to invokeLater
```

```
108
109     } // end method run
110
111 }  // end class LowThread
```



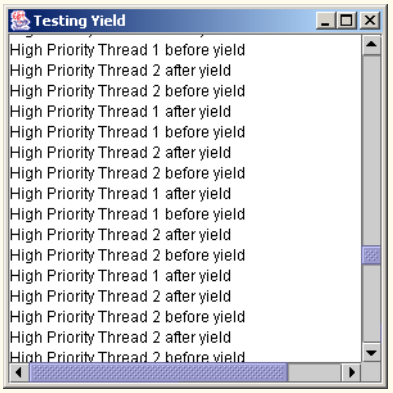
16.12 If your system does not support timeslicing, write a Java program that demonstrates two threads using `yield` to enable one another to execute.

ANS:

```
1  // Exercise 16.12 Solution: YieldTest.java
2  // Program demonstrates yielding between threads.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class YieldTest extends JFrame {
8     private HighThread high[];
9     private JTextArea output;
10
11    public YieldTest()
12    {
13       super( "Testing Yield" );
14
15       output = new JTextArea( 10, 30 );
16
17       Container container = getContentPane();
18       container.add( new JScrollPane( output ) );
19
20       setSize( 300, 300 );
21       setVisible( true );
22    }
23
24    // instantiate and start Threads
25    public void start()
26    {
27       HighThread thread1 = new HighThread( output, 1 );
28       HighThread thread2 = new HighThread( output, 2 );
29       thread1.start();
30       thread2.start();
31    }
32
```

```
33     public static void main( String args[] )
34     {
35        YieldTest application = new YieldTest();
36        application.start();
37        application.setDefaultCloseOperation( EXIT_ON_CLOSE );
38     }
39
40  }  // end class YieldTest
41
42  // high priority Thread
43  class HighThread extends Thread {
44     private JTextArea display;
45     private int number;
46
47     public HighThread( JTextArea textField, int n )
48     {
49        display = textField;
50        number = n;
51        setPriority( Thread.MAX_PRIORITY );
52     }
53
54     // run Thread
55     public void run()
56     {
57        for ( int i = 0; i < 500; i ++ ) {
58           SwingUtilities.invokeLater(
59
60              new Runnable() { // anonymous inner class
61
62                 public void run() {
63                    display.append( "High Priority Thread " +
64                       number + " before yield\n" );
65                 }
66
67              } // end anonymous inner class
68
69           ); // end call to invokeLater
70
71           yield();
72           SwingUtilities.invokeLater(
73
74              new Runnable() { // anonymous inner class
75
76                 public void run() {
77                    display.append( "High Priority Thread " +
78                       number + " after yield\n" );
79                 }
80
81              } // end anonymous inner class
82
83           ); // end call to invokeLater
84
85        } // end for
86
87     } // end method run
```

```
88
89   }  // end class HighThread
```



16.13 Two problems that can occur in systems like Java, that allow threads to wait, are deadlock, in which one or more threads will wait forever for an event that cannot occur, and indefinite postponement, in which one or more threads will be delayed for some unpredictably long time. Give an example of how each of these problems can occur in a multithreaded Java program.

16.14 (Readers and Writers) This exercise asks you to develop a Java monitor to solve a famous problem in concurrency control. This problem was first discussed and solved by P. J. Courtois, F. Heymans and D. L. Parnas in their research paper, "Concurrent Control with Readers and Writers," *Communications of the ACM*, Vol. 14, No. 10, October 1971, pp. 667–668. The interested student might also want to read C. A. R. Hoare's seminal research paper on monitors, "Monitors: An Operating System Structuring Concept," *Communications of the ACM*, Vol. 17, No. 10, October 1974, pp. 549–557. Corrigendum, *Communications of the ACM*, Vol. 18, No. 2, February 1975, p. 95. [The Readers and Writers problem is discussed at length in Chapter 5 of the author's book: Deitel, H. M., *Operating Systems*, Reading, MA: Addison-Wesley, 1990.]

a) With multithreading, many threads can access shared objects; as we have seen, access to shared objects needs to be synchronized carefully to avoid corrupting the objects.

b) Consider an airline-reservation system in which many clients are attempting to book seats on particular flights between particular cities. All of the information about flights and seats is stored in a common database in memory. The database consists of many entries, each representing a seat on a particular flight for a particular day between particular cities. In a typical airline-reservation scenario, the client will probe around in the database looking for the "optimal" flight to meet that client's needs. So a client may probe the database many times before deciding to book a particular flight. A seat that was available during this probing phase could easily be booked by someone else before the client has a chance to book it. In that case, when the client attempts to make the reservation, the client will discover that the data has changed and the flight is no longer available.

c) The client probing around the database is called a reader. The client attempting to book the flight is called a writer. Clearly, any number of readers can be probing the shared object at once, but each writer needs exclusive access to the shared object to prevent the object from being corrupted.

d) Write a multithreaded Java program that launches multiple reader threads and multiple writer threads, each attempting to access a single reservation record. A writer thread has

two possible transactions, `makeReservation` and `cancelReservation`. A reader has one possible transaction, `queryReservation`.

e) First implement a version of your program that allows unsynchronized access to the reservation record. Show how the integrity of the database can be corrupted. Next implement a version of your program that uses Java monitor synchronization with `wait` and `notify` to enforce a disciplined protocol for readers and writers accessing the shared reservation data. In particular, your program should allow multiple readers to access the shared object simultaneously when no writer is active. But if a writer is active, then no readers should be allowed to access the shared object.

f) Be careful. This problem has many subtleties. For example, what happens when there are several active readers and a writer wants to write? If we allow a steady stream of readers to arrive and share the object, they could indefinitely postpone the writer (who may become tired of waiting and take his or her business elsewhere). To solve this problem, you might decide to favor writers over readers. But here, too, there is a trap, because a steady stream of writers could then indefinitely postpone the waiting readers, and they, too, might choose to take their business elsewhere! Implement your monitor with the following methods: `startReading`, which is called by any reader who wants to begin accessing a reservation; `stopReading` to be called by any reader who has finished reading a reservation; `startWriting` to be called by any writer who wants to make a reservation and `stopWriting` to be called by any writer who has finished making a reservation.

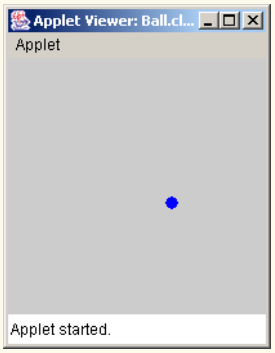
16.15 Write a program that bounces a blue ball inside an applet. The ball should begin moving with a `mousePressed` event. When the ball hits the edge of the applet, the ball should bounce off the edge and continue in the opposite direction.

ANS:

```
1  // Exercise 16.15 Solution: Ball.java
2  // Program bounces a ball around the applet
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Ball extends JApplet implements Runnable
8  {
9     private Thread blueBall;
10    private boolean xUp, yUp, bouncing;
11    private int x, y, xDx, yDy;
12    private final int MAX_X = 200, MAX_Y = 200;
13
14    public void init()
15    {
16       // initialize values
17       xUp = false;
18       yUp = false;
19       xDx = 1;
20       yDy = 1;
21       bouncing = false;
22
23       // let Ball Applet be its own MouseListener
24       addMouseListener(
25
26          new MouseListener() {
```

```
27
28             public void mousePressed( MouseEvent event )
29             {
30                createBall( event ); // delegate call to ball starter
31             }
32
33             public void mouseExited( MouseEvent event ) {}
34
35             public void mouseClicked( MouseEvent event ) {}
36
37             public void mouseReleased( MouseEvent event ) {}
38
39             public void mouseEntered( MouseEvent event ) {}
40          }
41       );
42
43       setSize( MAX_X, MAX_Y ); // set size of Applet
44    }
45
46    // creates a ball and sets it in motion if no ball exists
47    private void createBall( MouseEvent event )
48    {
49       if ( blueBall == null ) {
50          x = event.getX();
51          y = event.getY();
52          blueBall = new Thread( this );
53
54          bouncing = true; // start ball's bouncing
55          blueBall.start();
56       }
57    }
58
59    // called if applet is closed.  by setting blueBall to null,
60    // threads will be ended.
61    public void stop()
62    {
63       blueBall = null;
64    }
65
66    // draws ball at current position
67    public void paint( Graphics g )
68    {
69       super.paint( g );
70
71       if ( bouncing ) {
72          g.setColor( Color.blue );
73          g.fillOval( x, y, 10, 10 );
74       }
75    }
76
77    // action to perform on execution, bounces ball
78    // perpetually until applet is closed.
79    public void run()
80    {
```

```
81          while ( true ) {
82
83             // sleep for a random interval
84             try {
85                blueBall.sleep( 20 );
86             }
87
88             // process InterruptedException during sleep
89             catch ( InterruptedException exception ) {
90                System.err.println( exception.toString() );
91             }
92
93             // determine new x position
94             if ( xUp == true )
95                x += xDx;
96             else
97                x -= xDx;
98
99             // determine new y position
100            if ( yUp == true )
101               y += yDy;
102            else
103               y -= yDy;
104
105            // randomize variables for creating next move
106            if ( y <= 0 ) {
107               yUp = true;
108               yDy = ( int ) ( Math.random() * 5 + 2 );
109            }
110
111            else if ( y >= MAX_Y - 10 ) {
112               yDy = ( int ) ( Math.random() * 5 + 2 );
113               yUp = false;
114            }
115
116            if ( x <= 0 ) {
117               xUp = true;
118               xDx = ( int ) ( Math.random() * 5 + 2 );
119            }
120
121            else if ( x >= MAX_X - 10 ) {
122               xUp = false;
123               xDx = ( int ) ( Math.random() * 5 + 2 );
124            }
125
126            repaint();
127
128         }  // end while
129
130      }  // end method run
131
132   }  // end class Ball
```



16.16 Modify the program of Exercise 16.15 to add a new ball each time the user clicks the mouse. Provide for a minimum of 20 balls. Randomly choose the color for each new ball.

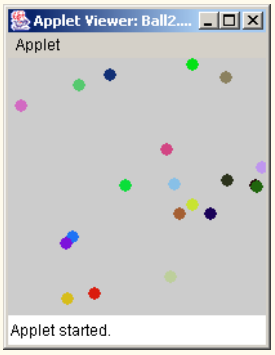
ANS:

```
1  // Exercise 16.16 Solution: Ball2.java
2  // Program bounces a ball around the applet
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Ball2 extends JApplet
8  {
9     private final int MAX_X = 200, MAX_Y = 200;
10    private final int MAX_BALLS = 20;
11    private BallThread[] balls;
12    private int count;
13
14    public void init()
15    {
16       count = 0; // initialize values
17
18       balls = new BallThread[ 20 ];
19
20       // let Ball Applet be its own MouseListener
21       addMouseListener(
22
23          new MouseAdapter() { // anonymous inner class
24
25             public void mousePressed( MouseEvent event )
26             {
27                createBall( event ); // delegate call to ball starter
28             }
29
30          } // end anonymous inner class
31
32       ); // end call to addMouseListener
33
34       setSize( MAX_X, MAX_Y ); // set size of Applet
35    }
```

```
36
37       // creates a ball and sets it in motion
38       private void createBall( MouseEvent event )
39       {
40          if ( count < MAX_BALLS ) {
41             int x = event.getX();
42             int y = event.getY();
43             balls[ count ] = new BallThread( x, y );
44
45             balls[ count ].start(); // start ball's bouncing
46             count++;
47          }
48       }
49
50       // called if applet is closed.  by setting ball to null,
51       // threads will be ended.
52       public void stop()
53       {
54          for ( int i = 0; i < count; i++ )
55             balls[ i ] = null;
56
57          count = 0;
58       }
59
60       // draws ball at current position
61       public void paint( Graphics g )
62       {
63          super.paint( g );
64
65          for ( int i = 0; i < count; i++ ) {
66             g.setColor( balls[ i ].getColor() );
67             g.fillOval( balls[ i ].getX(), balls[ i ].getY(), 10, 10 );
68          }
69       }
70
71       // class that represents one ball
72       private class BallThread extends Thread {
73          private Color color;
74          private int x, y, xDx, yDy;
75          private boolean xUp, yUp;
76          private final int MAX_X = 200, MAX_Y = 200;
77
78          public BallThread( int xCoord, int yCoord ) {
79             // initialize all the variables
80             xUp = false;
81             yUp = false;
82             xDx = 1;
83             yDy = 1;
84             x = xCoord;
85             y = yCoord;
86
87             // create a random color for the ball
88             color = new Color( (float)Math.random(), (float)Math.random(),
89                                (float)Math.random() );
90          }
```

```
91
92          // return the color
93          public Color getColor() {
94             return color;
95          }
96
97          // return the x position
98          public int getX() {
99             return x;
100         }
101
102         // return the y position
103         public int getY() {
104            return y;
105         }
106
107         // action to perform on execution, bounces ball
108         // perpetually until applet is closed.
109         public void run()
110         {
111            while ( true ) {
112
113               // sleep for a random interval
114               try {
115                  this.sleep( 20 );
116               }
117
118               // process InterruptedException during sleep
119               catch ( InterruptedException exception ) {
120                  // applet is closing
121               }
122
123               // determine new x position
124               if ( xUp == true )
125                  x += xDx;
126               else
127                  x -= xDx;
128
129               // determine new y position
130               if ( yUp == true )
131                  y += yDy;
132               else
133                  y -= yDy;
134
135               // randomize variables for creating next move
136               if ( y <= 0 ) {
137                  yUp = true;
138                  yDy = ( int ) ( Math.random() * 5 + 2 );
139               }
140
141               else if ( y >= MAX_Y - 10 ) {
142                  yDy = ( int ) ( Math.random() * 5 + 2 );
143                  yUp = false;
144               }
```

```
145
146             if ( x <= 0 ) {
147                xUp = true;
148                xDx = ( int ) ( Math.random() * 5 + 2 );
149             }
150
151             else if ( x >= MAX_X - 10 ) {
152                xUp = false;
153                xDx = ( int ) ( Math.random() * 5 + 2 );
154             }
155
156             repaint();
157
158          }  // end while
159
160       }  // end method run
161
162    } // end class BallThread
163
164 }  // end class Ball2
```



16.17 Modify the program of Exercise 16.16 to add shadows. As a ball moves, draw a solid black oval at the bottom of the applet. You may consider adding a 3-D effect by increasing or decreasing the size of each ball when a ball hits the edge of the applet.

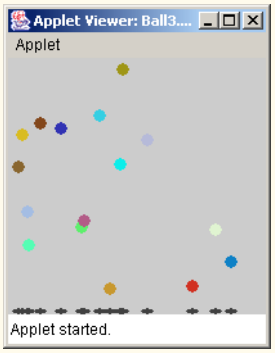
ANS:

```
1  // Exercise 16.17 Solution: Ball3.java
2  // Program bounces a ball around the applet
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Ball3 extends JApplet
8  {
9     private final int MAX_X = 200, MAX_Y = 200;
10    private final int MAX_BALLS = 20;
11    private BallThread[] balls;
12    private int count;
13
```

```
14     public void init()
15     {
16        count = 0;
17
18        balls = new BallThread[ 20 ];
19
20        // let Ball Applet be its own MouseListener
21        addMouseListener(
22
23           new MouseAdapter() { // anonymous inner class
24
25              public void mousePressed( MouseEvent event )
26              {
27                 createBall( event ); // delegate call to ball starter
28              }
29
30           } // end anonymous inner class
31
32        ); // end call to addMouseListener
33
34        setSize( MAX_X, MAX_Y ); // set size of Applet
35     }
36
37     // creates a ball and sets it in motion
38     private void createBall( MouseEvent event )
39     {
40        if ( count < MAX_BALLS ) {
41           int x = event.getX();
42           int y = event.getY();
43           balls[ count ] = new BallThread( x, y );
44
45           balls[ count ].start(); // start ball's bouncing
46           count++;
47        }
48     }
49
50     // called if applet is closed.  by setting ball to null,
51     // threads will be ended.
52     public void stop()
53     {
54        for ( int i = 0; i < count; i++ )
55           balls[ i ] = null;
56
57        count = 0;
58     }
59
60     // draws ball at current position
61     public void paint( Graphics g )
62     {
63        super.paint( g );
64
65        for ( int i = 0; i < count; i++ ) {
66           g.setColor( Color.darkGray );
67           g.fillOval( balls[ i ].getX(), 195, 10, 5 );
```

```
68
69             g.setColor( balls[ i ].getColor() );
70             g.fillOval( balls[ i ].getX(), balls[ i ].getY(), 10, 10 );
71          }
72       }
73
74       // class that represents one ball
75       private class BallThread extends Thread {
76          private Color color;
77          private int x, y;
78          private boolean xUp, yUp;
79          private int xDx, yDy;
80          private final int MAX_X = 200, MAX_Y = 200;
81
82          public BallThread( int xCoord, int yCoord ) {
83             // initialize all the variables
84             xUp = false;
85             yUp = false;
86             xDx = 1;
87             yDy = 1;
88             x = xCoord;
89             y = yCoord;
90
91             // create a random color for the ball
92             color = new Color( (float)Math.random(), (float)Math.random(),
93                                (float)Math.random() );
94          }
95
96          // return the color
97          public Color getColor() {
98             return color;
99          }
100
101         // return the x position
102         public int getX() {
103            return x;
104         }
105
106         // return the y position
107         public int getY() {
108            return y;
109         }
110
111         // action to perform on execution, bounces ball
112         // perpetually until applet is closed.
113         public void run()
114         {
115            while ( true ) {
116
117               // sleep for a random interval
118               try {
119                  this.sleep( 20 );
120               }
121
```

```
122             // process InterruptedException during sleep
123             catch ( InterruptedException exception ) {
124                // applet is closing
125             }
126
127             // determine new x position
128             if ( xUp == true )
129                x += xDx;
130             else
131                x -= xDx;
132
133             // determine new y position
134             if ( yUp == true )
135                y += yDy;
136             else
137                y -= yDy;
138
139             // randomize variables for creating next move
140             if ( y <= 0 ) {
141                yUp = true;
142                yDy = ( int ) ( Math.random() * 5 + 2 );
143             }
144
145             else if ( y >= MAX_Y - 10 ) {
146                yDy = ( int ) ( Math.random() * 5 + 2 );
147                yUp = false;
148             }
149
150             if ( x <= 0 ) {
151                xUp = true;
152                xDx = ( int ) ( Math.random() * 5 + 2 );
153             }
154
155             else if ( x >= MAX_X - 10 ) {
156                xUp = false;
157                xDx = ( int ) ( Math.random() * 5 + 2 );
158             }
159
160             repaint();
161
162          }  // end while
163
164       }  // end method run
165
166    } // end class BallThread
167
168 }  // end class Ball3
```



16.18 Modify the program of Exercise 16.15 or Exercise 16.16 to bounce the balls off each other when they collide.

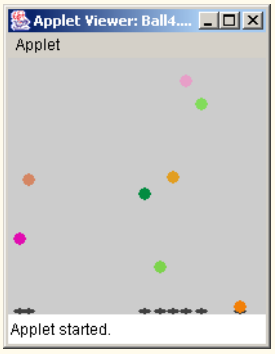
ANS:

```
1  // Exercise 16.18 Solution: Ball4.java
2  // Program bounces a ball around the applet
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Ball4 extends JApplet
8  {
9     private final int DIAMETER = 10;
10    private final int MAX_X = 200, MAX_Y = 200;
11    private final int MAX_BALLS = 20;
12    private BallThread[] balls;
13    private int count;
14
15    public void init()
16    {
17       count = 0;
18
19       balls = new BallThread[ 20 ];
20
21       // let Ball Applet be its own MouseListener
22       addMouseListener(
23
24          new MouseAdapter() { // anonymous inner class
25
26             public void mousePressed( MouseEvent event )
27             {
28                createBall( event ); // delegate call to ball starter
29             }
30
31          } // end anonymous inner class
32
33       ); // end call to addMouseListener
34
```

```
35          setSize( MAX_X, MAX_Y ); // set size of Applet
36       }
37
38       // creates a ball and sets it in motion
39       private void createBall( MouseEvent event )
40       {
41          if ( count < MAX_BALLS ) {
42             int x = event.getX();
43             int y = event.getY();
44             balls[ count ] = new BallThread( x, y );
45
46             balls[ count ].start(); // start ball's bouncing
47             count++;
48          }
49       }
50
51       // called if applet is closed.  by setting ball to null,
52       // threads will be ended.
53       public void stop()
54       {
55          for ( int i = 0; i < count; i++ )
56             balls[ i ] = null;
57
58          count = 0;
59       }
60
61       // draws ball at current position
62       public void paint( Graphics g )
63       {
64          super.paint( g );
65
66          for ( int i = 0; i < count; i++ ) {
67
68             for ( int j = 0; j < i; j ++ )
69                if ( balls[ i ].collide( balls[ j ] ) ) {
70                   balls[ i ].collision();
71                   balls[ j ].collision();
72                }
73
74             g.setColor( Color.darkGray );
75             g.fillOval( balls[ i ].getX(), 195, DIAMETER, DIAMETER / 2 );
76
77             g.setColor( balls[ i ].getColor() );
78             g.fillOval( balls[ i ].getX(), balls[ i ].getY(),
79                DIAMETER, DIAMETER );
80          }
81       }
82
83       // class that represents one ball
84       private class BallThread extends Thread {
85          private Color color;
86          private int x, y;
87          private boolean xUp, yUp;
88          private int xDx, yDy;
```

```
89          private final int MAX_X = 200, MAX_Y = 200;
90
91          public BallThread( int xCoord, int yCoord ) {
92             // initialize all the variables
93             xUp = false;
94             yUp = false;
95             xDx = 1;
96             yDy = 1;
97             x = xCoord;
98             y = yCoord;
99
100            // create a random color for the ball
101            color = new Color( (float)Math.random(), (float)Math.random(),
102               (float)Math.random() );
103         }
104
105         // return the color
106         public Color getColor() {
107            return color;
108         }
109
110         // return the x position
111         public int getX() {
112            return x;
113         }
114
115         // return the y position
116         public int getY() {
117            public y;
118         }
119
120         // action to perform on execution, bounces ball
121         // perpetually until applet is closed.
122         public void run()
123         {
124            while ( true ) {
125
126               // sleep for a random interval
127               try {
128                  this.sleep( 20 );
129               }
130
131               // process InterruptedException during sleep
132               catch ( InterruptedException exception ) {
133                  // applet is closing
134               }
135
136               // determine new x position
137               if ( xUp == true )
138                  x += xDx;
139               else
140                  x -= xDx;
141
```

```
142             // determine new y position
143             if ( yUp == true )
144                y += yDy;
145             else
146                y -= yDy;
147
148             // randomize variables for creating next move
149             if ( y <= 0 ) {
150                yUp = true;
151                yDy = ( int ) ( Math.random() * 5 + 2 );
152             }
153
154             else if ( y >= MAX_Y - DIAMETER ) {
155                yDy = ( int ) ( Math.random() * 5 + 2 );
156                yUp = false;
157             }
158
159             if ( x <= 0 ) {
160                xUp = true;
161                xDx = ( int ) ( Math.random() * 5 + 2 );
162             }
163
164             else if ( x >= MAX_X - DIAMETER ) {
165                xUp = false;
166                xDx = ( int ) ( Math.random() * 5 + 2 );
167             }
168
169             repaint();
170
171          }  // end while
172
173       }  // end method run
174
175       public void collision()
176       {
177          xUp = !xUp;
178          xDx = ( int ) ( Math.random() * 5 + 2 );
179          yUp = !yUp;
180          yDy = ( int ) ( Math.random() * 5 + 2 );
181       }
182
183       public boolean collide( BallThread other )
184       {
185          if ( ( ( x - other.getX() ) * ( x - other.getX() ) +
186                 ( y - other.getY() ) * ( y - other.getY() ) ) < 100 )
187             return true;
188          else
189             return false;
190       }
191
192    } // end class BallThread
193
194 }  // end class Ball4
```



17

Files and Streams

Objectives

- To be able to create, read, write and update files.
- To be able to use class `File`.
- To understand the Java streams class hierarchy.
- To be able to use the `FileInputStream` and `FileOutputStream` classes.
- To be able to use a `JFileChooser` dialog to access files and directories.
- To be able to use the `ObjectInputStream` and `ObjectOutputStream` classes.
- To be able to use class `RandomAccessFile`.
- To become familiar with sequential-access and random-access file processing.

I can only assume that a "Do Not File" document is filed in a "Do Not File" file.
Senator Frank Church
Senate Intelligence Subcommittee Hearing, 1975

Consciousness … does not appear to itself chopped up in bits. … A "river" or a "stream" are the metaphors by which it is most naturally described.
William James

I read part of it all the way through.
Samuel Goldwyn

It is quite a three-pipe problem.
Sir Arthur Conan Doyle

SELF-REVIEW EXERCISES

17.1 Fill in the blanks in each of the following statements:

a) Ultimately, all data items processed by a computer are reduced to combinations of _____ and _____.

ANS: ones, zeros.

b) The smallest data item a computer can process is called a(n) _____.

ANS: bit.

c) A(n) _____ is a group of related records.

ANS: file.

d) Digits, letters and special symbols are referred to as _____.

ANS: characters.

e) A group of related files is called a _____.

ANS: database.

f) Method _____ of the file stream classes `FileOutputStream`, `FileInputStream`, and `RandomAccessFile` closes a file.

ANS: `close`.

g) `RandomAccessFile` method _____ reads an integer from the specified stream.

ANS: `readInt`.

h) `RandomAccessFile` method _____ reads a line of text from the specified stream.

ANS: `readLine`.

i) `RandomAccessFile` method _____ sets the file-position pointer to a specific location in a file for input or output.

ANS: `seek`.

17.2 Determine which of the given statements are *true* and which are *false*. If *false*, explain why.

a) The programmer must explicitly create `System.in`, `System.out` and `System.err`.

ANS: False. These three streams are created automatically for the programmer.

b) If the file-position pointer points to a location in a sequential file other than the beginning of the file, the file must be closed and reopened to read from the beginning of the file.

ANS: True.

c) It is not necessary to search all the records in a random-access file in order to find a record.

ANS: True.

d) Records in random-access files must be of uniform length.

ANS: False. Records in a random-access file are normally of uniform length.

e) Method `seek` must seek relative to the beginning of a file.

ANS: True.

17.3 Complete the following tasks, assume that each applies to the same program:

a) Write a statement that opens file `"oldmast.dat"` for input; use `ObjectInputStream` object `inOldMaster` to wrap a `FileInputStream` object.

ANS:
```
ObjectInputStream inOldMaster = new ObjectInputStream(
      new FileInputStream( "oldmast.dat" ) );
```

b) Write a statement that opens file `"trans.dat"` for input; use `ObjectInputStream` object `inTransaction` to wrap a `FileInputStream` object.

ANS:
```
ObjectInputStream inTransaction = new ObjectInputStream(
   new FileInputStream( "trans.dat" ) );
```

c) Write a statement that opens file `"newmast.dat"` for output (and creation); use `ObjectOutputStream` object `outNewMaster` to wrap a `FileOutputStream`.

ANS:
```
ObjectOutputStream outNewMaster = new ObjectOutputStream(
   new FileOutputStream( "newmast.dat" ) );
```

d) Write a statement that reads a record from the file "oldmast.dat". The record is an object of class AccountRecord; use ObjectInputStream object inOldMaster.

ANS: `accountRecord = ( AccountRecord ) inOldMaster.getObject();`

e) Write a statement that reads a record from the file "trans.dat". The record is an object of class TransactionRecord; use ObjectInputStream object inTransaction.

ANS: `transactionRecord = ( TransactionRecord ) inTransaction.getObject();`

f) Write a statement that outputs a record to the file "newmast.dat". The record is an object of type AccountRecord; use ObjectOutputStream object outNewMaster.

ANS: `outNewMaster.writeObject( newAccountRecord );`

17.4 Find the error in each block of codes and show how to correct it.

a) Assume account, company and amount are declared.

```
ObjectOutputStream outputStream;

outputStream.writeInt( account );
outputStream.writeChars( company );
outputStream.writeDouble( amount );
```

ANS: Error: The file has not been opened before the attempt is made to output data to the stream.

Correction: Create a new ObjectOutputStream object wrapping a FileOutputStream object in order to open the file for output.

b) The given statements should read a record from the file "payables.dat". The ObjectInputStream object inPayable refers to this file, and FileInputStream object inReceivable refers to the file "receivables.dat".

```
account = inReceivable.readInt();
companyID = inReceivable.readLong();
amount = inReceivable.readDouble();
```

ANS: Error: The incorrect FileInputStream object is being used to read from file "payables.dat".

Correction: Use object inPayable to refer to "payables.dat".

EXERCISES

17.5 Fill in the blanks in each of the following statements:

a) Computers store large amounts of data on secondary storage devices as _____.

ANS: files

b) A(n) _____ is composed of several fields.

ANS: record

c) To facilitate the retrieval of specific records from a file, one field in each record is chosen as a(n) _____.

ANS: key

d) The majority of information stored in computer systems is stored in _____ files.

ANS: sequential

e) The standard stream objects are _____, _____ and _____.

ANS: System.in, System.out, System.err

17.6 Determine which of the given statements are *true* and which are *false*. If *false*, explain why.

a) The impressive functions performed by computers essentially involve the manipulation of zeros and ones.

ANS: True.

b) People specify programs and data items as characters; computers then manipulate and process these characters as groups of zeros and ones.

ANS: True.

c) A person's five-digit zip code is an example of a numeric field.

ANS: True. [*Note:* The term "numeric field" was accidentally removed from the chapter. A field of all digits would typically be represented with a `String` or an `int`. This question probable should not be used.]

d) A person's street address is generally considered to be an alphabetic field.

ANS: False. This would be a field consisting of letters, digits and spaces (an alphanumeric field) represented with a `String`. [*Note:* The term "alphabetic" was accidentally removed from the chapter. This question probable should not be used.]

e) Data items represented in computers form a data hierarchy in which data items become larger and more complex as we progress from fields to characters to bits and so on.

ANS: False. Data becomes more complex as we progress from bits to characters to fields, etc.

f) A record key identifies a record as belonging to a particular field.

ANS: False. A record key identifies a record as belonging to a particular person or entity.

g) Companies store all their information in a single file in order to facilitate computer processing of the information. When a program creates a file, the file is automatically retained by the computer for future reference.

ANS: False. Companies typically store information in multiple files.

17.7 Self-Review Exercise 17.3 asks the reader to write a series of single statements. Actually, these statements form the core of an important type of file-processing program, namely, a file-matching program. In commercial data processing, it is common to have several files in each application system. In an accounts receivable system, for example, there is generally a master file containing detailed information about each customer, such as the customer's name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements and possibly a condensed history of recent purchases and cash payments.

a) As transactions occur (i.e., sales are made and payments arrive in the mail), information about them is entered into a file. At the end of each business period (i.e., a month for some companies, a week for others, and a day in some cases), the file of transactions (called `"trans.dat"` in Self-Review Exercise 17.3) is applied to the master file (called `"oldmast.dat"` in Self-Review Exercise 17.3) to update each account's purchase and payment record. During an update, the master file is rewritten as the file `"newmast.dat"`, which is then used at the end of the next business period to begin the updating process again.

b) File-matching programs must deal with certain problems that do not exist in single-file programs. For example, a match does not always occur. A customer on the master file might not have made any purchases or cash payments in the current business period; therefore, no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments could have just moved to this community, and the company might not have had a chance to create a master record for this customer.

c) Use the statements in Self-Review Exercise 17.3 as a basis for writing a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential file with records stored in increasing account-number order.

d) When a match occurs (i.e., records with the same account number appear on both the master file and the transaction file), add the dollar amount on the transaction file to the current balance on the master file, and write the `"newmast.dat"` record. (Assume that purchases are indicated by positive amounts on the transaction file and payments by neg-

ative amounts.) When there is a master record for a particular account, but no corresponding transaction record, merely write the master record to "newmast.dat". When there is a transaction record, but no corresponding master record, print to a log file the message "Unmatched transaction record for account number ..." (fill in the account number from the transaction record).

e) The log file should be a text file named "log.txt". This file should be created using PrintStream object logFile to wrap a FileOutputStream. Error messages can be output to this file, using method println of class PrintStream.

ANS:

```
1  // Exercise 17.7 Solution: AccountRecord.java
2  // A class that represents one record of information.
3  import java.io.Serializable;
4
5  public class AccountRecord implements Serializable {
6     private int account;
7     private String firstName;
8     private String lastName;
9     private double balance;
10
11    // no-argument constructor calls other constructor with default values
12    public AccountRecord()
13    {
14       this( 0, "", "", 0.0 );
15    }
16
17    // initialize a record
18    public AccountRecord( int acct, String first, String last, double bal )
19    {
20       setAccount( acct );
21       setFirstName( first );
22       setLastName( last );
23       setBalance( bal );
24    }
25
26    // add a transaction record to an account record
27    public AccountRecord combine( TransactionRecord transaction ) {
28
29       return new AccountRecord( account, firstName, lastName,
30          balance + transaction.getBalance() );
31    }
32
33    // set account number
34    private void setAccount( int acct )
35    {
36       account = acct;
37    }
38
39    // get account number
40    public int getAccount()
41    {
42       return account;
43    }
```

```
44
45     // set first name
46     private void setFirstName( String first )
47     {
48        firstName = first;
49     }
50
51     // get first name
52     public String getFirstName()
53     {
54        return firstName;
55     }
56
57     // set last name
58     private void setLastName( String last )
59     {
60        lastName = last;
61     }
62
63     // get last name
64     public String getLastName()
65     {
66        return lastName;
67     }
68
69     // set balance
70     private void setBalance( double bal )
71     {
72        balance = bal;
73     }
74
75     // get balance
76     public double getBalance()
77     {
78        return balance;
79     }
80
81  } // end class AccountRecord
```

ANS:

```
1  // Exercise 17.7 Solution: TransactionRecord.java
2  // A class that represents one transaction record.
3  import java.io.Serializable;
4
5  public class TransactionRecord implements Serializable {
6     private int account;
7     private double balance;
8
9     // no-argument constructor calls other constructor with default values
10    public TransactionRecord()
11    {
12       this( 0, 0.0 );
```

```
13     }
14
15     // initialize a record
16     public TransactionRecord( int acct, double bal )
17     {
18        setAccount( acct );
19        setBalance( bal );
20     }
21
22     // set account number
23     private void setAccount( int acct )
24     {
25        account = acct;
26     }
27
28     // get account number
29     public int getAccount()
30     {
31        return account;
32     }
33
34     // set balance
35     private void setBalance( double bal )
36     {
37        balance = bal;
38     }
39
40     // get balance
41     public double getBalance()
42     {
43        return balance;
44     }
45
46  } // end class TransactionRecord
```

ANS:

```
1  // Exercise 17.7 Solution: FileMatch.java
2  // Combine account file and a transactions file into a new account file
3  import java.io.*;
4  import javax.swing.*;
5
6  public class FileMatch {
7     private static ObjectInputStream inOldMaster, inTransaction;
8     private static ObjectOutputStream outNewMaster;
9     private static PrintStream logFile;
10
11    public static void main( String args[] )
12    {
13       try {
14          // file streams for input and output files
15          inOldMaster = new ObjectInputStream(
16             new FileInputStream( "oldmast.dat" ) );
```

```
17             inTransaction = new ObjectInputStream(
18                new FileInputStream( "trans.dat" ) );
19             outNewMaster = new ObjectOutputStream(
20                new FileOutputStream( "newmast.dat" ) );
21             logFile = new PrintStream( new FileOutputStream( "log.txt" ) );
22          }
23
24          catch ( IOException io ) {
25             JOptionPane.showMessageDialog( null, "Error opening the file",
26                "IO Exception", JOptionPane.ERROR_MESSAGE );
27          }
28
29          int transactionNumber, accountNumber;
30          TransactionRecord transaction;
31          AccountRecord account;
32
33          // block for reading/writing all the records
34          read: {
35
36             try {
37
38                // get a transaction record and its account number
39                transaction = getTransactionRecord();
40
41                // if the transaction is null, we are done
42                if ( transaction == null )
43                   break read;
44
45                transactionNumber = transaction.getAccount();
46
47                // get an account record and its account number
48                account = getAccountRecord();
49
50                // if the account is null, we are done
51                if ( account == null )
52                   break read;
53
54                accountNumber = account.getAccount();
55
56                while( true ) {
57
58                   while ( accountNumber < transactionNumber ) {
59
60                      // there is no transaction for this account
61                      outNewMaster.writeObject( account );
62
63                      account = getAccountRecord(); // get a new account
64                      if ( account == null )
65                         break read;
66
67                      accountNumber = account.getAccount();
68                   }
69
```

```
70                // if there is a transaction for this account
71                if ( accountNumber == transactionNumber ) {
72
73                   // combine the records
74                   AccountRecord outputRecord =
75                      account.combine( transaction );
76
77                   // write them to the master file
78                   outNewMaster.writeObject( outputRecord );
79
80                   // get a new transaction
81                   transaction = getTransactionRecord();
82                   if ( transaction == null )
83                      break read;
84
85                   transactionNumber = transaction.getAccount();
86
87                   // get a new account
88                   account = getAccountRecord();
89                   if ( account == null )
90                      break read;
91
92                   accountNumber = account.getAccount();
93                }
94
95                while( transactionNumber < accountNumber ) {
96
97                   // there is no account for this transaction
98                   logFile.println(
99                      "Unmatched transaction record for account number " +
100                     transactionNumber );
101
102                  // get a new transaction
103                  transaction = getTransactionRecord();
104                  if ( transaction == null )
105                     break read;
106
107                  transactionNumber = transaction.getAccount();
108               }
109
110            } // end while
111
112         } // end try
113
114         catch( IOException io ) {
115            JOptionPane.showMessageDialog( null,
116               "Error reading or writing the file", "IO Error",
117               JOptionPane.ERROR_MESSAGE );
118            System.exit( 1 );
119         }
120
121         catch( ClassNotFoundException noClass ) {
122            JOptionPane.showMessageDialog( null, "Error reading the file",
123               "Invalid class name", JOptionPane.ERROR_MESSAGE );
```

```
124             System.exit( 1 );
125          }
126
127       } // end read block
128
129       // close the files
130       try {
131          inTransaction.close();
132          outNewMaster.close();
133          inOldMaster.close();
134          logFile.close();
135       }
136       catch( IOException io ) {
137          JOptionPane.showMessageDialog( null, "Error closing the file",
138             "IO Error", JOptionPane.ERROR_MESSAGE );
139          System.exit( 1 );
140       }
141
142    } // end main
143
144    // get a transaction record
145    private static TransactionRecord getTransactionRecord()
146       throws IOException, ClassNotFoundException {
147       TransactionRecord transaction;
148
149       // try to read the record
150       try {
151          transaction = ( TransactionRecord )inTransaction.readObject();
152       }
153
154       // if we hit the end of the transaction file
155       catch( EOFException eof ) {
156          try {
157
158             // read the remaining records from the old master
159             while( true )
160                outNewMaster.writeObject( inOldMaster.readObject() );
161          }
162
163          // we have hit the end of the old master file
164          catch( EOFException eof2 ) {
165             return null;
166          }
167
168       } // end outer catch
169
170       // return a transaction if we successfully read it
171       return transaction;
172
173    } // end method getTransactionRecord
174
175    // get an account record
176    private static AccountRecord getAccountRecord()
177       throws IOException, ClassNotFoundException {
```

```
178         AccountRecord account;
179
180         // try to read an account record
181         try {
182            account = ( AccountRecord )inOldMaster.readObject();
183         }
184
185         // we hit the end of the old master file
186         catch( EOFException eof ) {
187
188            try {
189               TransactionRecord temp;
190
191               // all of these records are transactions without accounts
192               while( true ) {
193                  temp = ( TransactionRecord )inTransaction.readObject();
194                  logFile.println(
195                     "Unmatched transaction record for account number " +
196                     temp.getAccount() );
197               }
198            }
199
200            // we hit the end of the transaction file
201            catch( EOFException eof2 ) {
202               return null;
203            }
204
205         } // end outer catch
206
207         return account;
208
209      } // end method getAccountRecord
210
211   } // end class FileMatch
```

17.8 After writing the program of Exercise 17.7, write a simple program to create some test data for checking out the program. Use the sample account data in Fig. 17.24 and Fig. 17.25. Run the program of Exercise 17.7, using the files of test data created in this exercise. Print the new master file. Check that the accounts have been updated correctly.

Master file Account number	Name	Balance
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	-14.22

Fig. 17.24 Sample data for master file.

Transaction file Account number	Transaction amount
100	27.14
300	62.11
400	100.56
900	82.17

Fig. 17.25 Sample data for transaction file.

ANS:

```
1  // Exercise 17.8 Solution: CreateData.java
2  // Create data to put into an account file and a transactions file
3  import java.io.*;
4  import javax.swing.*;
5
6  public class CreateData {
7     private static ObjectOutputStream outOldMaster, outTransaction;
8
9     public static void main( String args[] )
10    {
11       try {
12          // file streams for output files
13          outOldMaster = new ObjectOutputStream(
14             new FileOutputStream( "oldmast.dat" ) );
15          outTransaction = new ObjectOutputStream(
16             new FileOutputStream( "trans.dat" ) );
17       }
18
19       catch ( IOException io ) {
20          JOptionPane.showMessageDialog( null, "Error opening the file",
21             "IO Exception", JOptionPane.ERROR_MESSAGE );
22       }
23
24       try {
25
26          outOldMaster.writeObject(
27             new AccountRecord( 100, "Alan", "Jones", 348.17 ) );
28          outOldMaster.writeObject(
29             new AccountRecord( 300, "Mary", "Smith", 27.19 ) );
30          outOldMaster.writeObject(
31             new AccountRecord( 500, "Sam", "Sharp", 0.00 ) );
32          outOldMaster.writeObject(
33             new AccountRecord( 700, "Suzy", "Green", -14.22 ) );
34
35          outTransaction.writeObject(
36             new TransactionRecord( 100, 27.14 ) );
37          outTransaction.writeObject(
38             new TransactionRecord( 300, 62.11 ) );
```

```
39             outTransaction.writeObject(
40                new TransactionRecord( 400, 100.56 ) );
41             outTransaction.writeObject(
42                new TransactionRecord( 900, 82.17 ) );
43
44          } // end try
45
46          catch( IOException io ) {
47             JOptionPane.showMessageDialog( null,
48                "Error reading or writing to the file",
49                "IO Error", JOptionPane.ERROR_MESSAGE );
50             System.exit( 1 );
51          }
52
53          // close the files
54          try {
55             outTransaction.close();
56             outOldMaster.close();
57          }
58
59          catch( IOException io ) {
60             JOptionPane.showMessageDialog( null, "Error closing the file",
61                "IO Error", JOptionPane.ERROR_MESSAGE );
62             System.exit( 1 );
63          }
64
65       } // end main
66
67    } // end class CreateData
```

17.9 It is possible (and actually common) to have several transaction records with the same record key. This situation occurs when a particular customer makes several purchases and cash payments during a business period. Rewrite your accounts receivable file-matching program of Exercise 17.7 to provide for the possibility of handling several transaction records with the same record key. Modify the test data of Exercise 17.8 to include the additional transaction records in Fig. 17.26.

ANS:

```
1  // Exercise 17.9 Solution: FileMatch.java
2  // Combine account file and a transactions file into a new account file
3  import java.io.*;
4  import javax.swing.*;
5
6  public class FileMatch {
7     private static ObjectInputStream inOldMaster, inTransaction;
8     private static ObjectOutputStream outNewMaster;
9     private static PrintStream logFile;
10    private static TransactionRecord transaction;
11    private static AccountRecord account;
12
13    public static void main( String args[] )
14    {
15       try {
```

```
16            // file streams for input and output files
17            inOldMaster = new ObjectInputStream(
18               new FileInputStream( "oldmast.dat" ) );
19            inTransaction = new ObjectInputStream(
20               new FileInputStream( "trans.dat" ) );
21            outNewMaster = new ObjectOutputStream(
22               new FileOutputStream( "newmast.dat" ) );
23            logFile = new PrintStream( new FileOutputStream( "log.txt" ) );
24         }
25
26         catch ( IOException io ) {
27            JOptionPane.showMessageDialog( null, "Error opening the file",
28               "IO Exception", JOptionPane.ERROR_MESSAGE );
29         }
30
31         int transactionNumber, accountNumber;
32
33         // block for reading/writing all the records
34         read: {
35
36            try {
37
38               // get a transaction record and its account number
39               transaction = getTransactionRecord();
40
41               // if the transaction is null, we are done
42               if ( transaction == null )
43                  break read;
44
45               transactionNumber = transaction.getAccount();
46
47               // get an account record and its account number
48               account = getAccountRecord();
49
50               // if the account is null, we are done
51               if ( account == null )
52                  break read;
53
54               accountNumber = account.getAccount();
55
56               while( true ) {
57
58                  while ( accountNumber < transactionNumber ) {
59
60                     // there is no transaction for this account
61                     outNewMaster.writeObject( account );
62
63                     account = getAccountRecord(); // get a new account
64                     if ( account == null )
65                        break read;
66
67                     accountNumber = account.getAccount();
68                  }
69
```

```
70                // if there is a transaction for this account
71                if ( accountNumber == transactionNumber ) {
72                   AccountRecord outputRecord = account;
73
74                   while ( accountNumber == transactionNumber ) {
75
76                      // combine the records
77                      outputRecord = outputRecord.combine( transaction );
78
79                      // get a new transaction
80                      transaction = getTransactionRecord();
81                      if ( transaction == null )
82                         break read;
83
84                      transactionNumber = transaction.getAccount();
85                   }
86
87                   // write them to the master file
88                   outNewMaster.writeObject( outputRecord );
89
90                   // get a new account
91                   account = getAccountRecord();
92                   if ( account == null )
93                      break read;
94
95                   accountNumber = account.getAccount();
96                }
97
98                while( transactionNumber < accountNumber ) {
99
100                  // there is no account for this transaction
101                  logFile.println(
102                     "Unmatched transaction record for account number " +
103                     transactionNumber );
104
105                  // get a new transaction
106                  transaction = getTransactionRecord();
107                  if ( transaction == null )
108                     break read;
109
110                  transactionNumber = transaction.getAccount();
111               }
112
113            } // end while
114
115         } // end try
116
117         catch( IOException io ) {
118            JOptionPane.showMessageDialog( null,
119               "Error reading or writing the file",
120               "IO Error", JOptionPane.ERROR_MESSAGE );
121            System.exit( 1 );
122         }
123
```

```
124          catch( ClassNotFoundException noClass ) {
125             JOptionPane.showMessageDialog( null, "Error reading the file",
126                "Invalid class name", JOptionPane.ERROR_MESSAGE );
127             System.exit( 1 );
128          }
129
130       } // end read block
131
132       // close the files
133       try {
134          inTransaction.close();
135          outNewMaster.close();
136          inOldMaster.close();
137          logFile.close();
138       }
139       catch( IOException io ) {
140          JOptionPane.showMessageDialog( null, "Error closing the file",
141             "IO Error", JOptionPane.ERROR_MESSAGE );
142          System.exit( 1 );
143       }
144
145    } // end main
146
147    // get a transaction record
148    private static TransactionRecord getTransactionRecord()
149       throws IOException, ClassNotFoundException {
150       TransactionRecord transaction;
151
152       // try to read the record
153       try {
154          transaction = ( TransactionRecord )inTransaction.readObject();
155       }
156
157       // if we hit the end of the transaction file
158       catch( EOFException eof ) {
159          try {
160
161             // read the remaining records from the old master
162             while( true )
163                outNewMaster.writeObject( inOldMaster.readObject() );
164          }
165
166          // we have hit the end of the old master file
167          catch( EOFException eof2 ) {
168             return null;
169          }
170
171       } // end outer catch
172
173       // return a transaction if we successfully read it
174       return transaction;
175
176    } // end method getTransactionRecord
177
```

```
178    // get an account record
179    private static AccountRecord getAccountRecord()
180       throws IOException, ClassNotFoundException {
181       AccountRecord account;
182
183       // try to read an account record
184       try {
185          account = (AccountRecord)inOldMaster.readObject();
186       }
187
188       // we hit the end of the old master file
189       catch( EOFException eof ) {
190
191          try {
192
193             // all of these records are transactions without accounts
194             while( true ) {
195                logFile.println(
196                   "Unmatched transaction record for account number " +
197                   transaction.getAccount() );
198                transaction =
199                   ( TransactionRecord )inTransaction.readObject();
200             }
201          }
202
203          // we hit the end of the transaction file
204          catch( EOFException eof2 ) {
205             return null;
206          }
207
208       } // end outer catch
209
210       return account;
211
212    } // end method getAccountRecord
213
214 } // end class FileMatch
```

17.10 You are the owner of a hardware store and need to keep an inventory that can tell you what different tools you have, how many of each you have on hand and the cost of each one. Write a program that initializes the random-access file `"hardware.dat"` to 100 empty records, lets you input the data concerning each tool, enables you to list all your tools, lets you delete a record for a tool that you no longer have and lets you update *any* information in the file. The tool identification number should be the record number. Use the information in Fig. 17.27 to start your file.

ANS:

```
1  // Exercise 17.10 Solution: Record.java
2  // Definition of class Record.
3  import java.io.*;
4
5  public class Record {
6     public final int SIZE = 46;
```

```
 7
 8     private int recordNumber, quantity;
 9     private String toolName;
10     private double cost;
11
12     public Record()
13     {
14        this( 0, "", 0, 0 );
15     }
16
17     public Record( int record, String name, int q, double c )
18     {
19        setRecordNumber( record );
20        setToolName( name );
21        setQuantity( q );
22        setCost( c );
23     }
24
25     // read a record from the specified RandomAccessFile
26     public void read( RandomAccessFile file ) throws IOException
27     {
28        setRecordNumber( file.readInt() );
29        setToolName( padName( file ) );
30        setQuantity( file.readInt() );
31        setCost( file.readDouble() );
32     }
33
34     // ensure that name is proper length
35     private String padName( RandomAccessFile file ) throws IOException
36     {
37        char name[] = new char[ 15 ];
38
39        for ( int i = 0; i < name.length; i++ )
40           name[ i ] = file.readChar();
41
42        return new String( name ).replace( '\0', ' ' );
43     }
44
45     // write a record to the specified RandomAccessFile
46     public void write( RandomAccessFile file ) throws IOException
47     {
48        file.writeInt( getRecordNumber() );
49
50        StringBuffer buffer;
51
52        if ( toolName != null )
53           buffer = new StringBuffer( toolName );
54
55        else
56           buffer = new StringBuffer( 15 );
57
58        buffer.setLength( 15 );
59
60        file.writeChars( buffer.toString() );
```

```
61          file.writeInt( getQuantity() );
62          file.writeDouble( getCost() );
63       }
64
65       // accessor methods
66       public void setRecordNumber( int n )
67       {
68          recordNumber = n;
69       }
70
71       public int getRecordNumber()
72       {
73          return recordNumber;
74       }
75
76       public void setToolName( String t )
77       {
78          toolName = t;
79       }
80
81       public String getToolName()
82       {
83          return toolName;
84       }
85
86       public void setQuantity( int q )
87       {
88          quantity = q;
89       }
90
91       public int getQuantity()
92       {
93          return quantity;
94       }
95
96       public void setCost( double c )
97       {
98          cost = c;
99       }
100
101      public double getCost()
102      {
103         return cost;
104      }
105
106   }  // end class Record
```

ANS:

```
1  // Exercise 17.10 Solution: HardwareStore.java
2  // Program simulates hardware store inventory checking.
3  import java.awt.*;
4  import java.io.*;
```

```
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class HardwareStore extends JFrame {
9     private UpdateDialog updateDialog;
10    private NewDialog newDialog;
11    private DeleteDialog deleteDialog;
12    private InventoryDialog inventoryDialog;
13    private JMenuItem newItem, updateItem, deleteItem,
14       inventoryItem, exitItem;
15    private JDesktopPane desktop;
16    private RandomAccessFile file;
17    private Record records[];
18
19    public HardwareStore()
20    {
21       super( "Hardware Store" );
22
23       records = new Record[ 100 ];
24
25       // open file
26       try {
27          file = new RandomAccessFile( "hardware.dat", "rw" );
28
29          for ( int i = 0; i < 100; i++ ) {
30             records[ i ] = new Record();
31             records[ i ].write( file );
32          }
33       }
34
35       catch ( IOException ioException ) {
36          System.err.println( ioException.toString() );
37          System.exit( 1 );
38       }
39
40       newDialog = new NewDialog( file );
41       updateDialog = new UpdateDialog( file );
42       deleteDialog = new DeleteDialog( file );
43       inventoryDialog = new InventoryDialog( file );
44
45       // set up GUI
46       Container container = getContentPane();
47       desktop = new JDesktopPane();
48       container.add( desktop );
49       desktop.add( newDialog );
50       desktop.add( updateDialog );
51       desktop.add( deleteDialog );
52       desktop.add( inventoryDialog );
53
54       JMenuBar menuBar = new JMenuBar();
55       setJMenuBar( menuBar );
56
57       JMenu fileMenu = new JMenu( "File" );
58       menuBar.add( fileMenu );
```

```
59
60          newItem = new JMenuItem( "New Record" );
61          newItem.addActionListener(
62
63             new ActionListener() { // anonymous inner class
64
65                public void actionPerformed( ActionEvent event )
66                {
67                   newDialog.setVisible( true );
68                }
69
70             } // end anonymous inner class
71
72          ); // end call to addActionListener
73
74          updateItem = new JMenuItem( "Update Record" );
75          updateItem.addActionListener(
76
77             new ActionListener() { // anonymous inner class
78
79                public void actionPerformed( ActionEvent event )
80                {
81                   updateDialog.setVisible( true );
82                }
83
84             } // end anonymous inner class
85
86          ); // end call to addActionListener
87
88          deleteItem = new JMenuItem( "Delete Record" );
89          deleteItem.addActionListener(
90
91             new ActionListener() { // anonymous inner class
92
93                public void actionPerformed( ActionEvent event )
94                {
95                   deleteDialog.setVisible( true );
96                }
97
98             } // end anonymous inner class
99
100         ); // end call to addActionListener
101
102         inventoryItem = new JMenuItem( "Show Inventory" );
103         inventoryItem.addActionListener(
104
105            new ActionListener() { // anonymous inner class
106
107               public void actionPerformed( ActionEvent event )
108               {
109                  inventoryDialog.setVisible( true );
110                  inventoryDialog.display();
111               }
112
113            } // end anonymous inner class
```

```
114
115       ); // end call to addActionListener
116
117       exitItem = new JMenuItem( "Exit" );
118       exitItem.addActionListener(
119
120          new ActionListener() { // anonymous inner class
121
122             public void actionPerformed( ActionEvent event )
123             {
124                try {
125                   file.close();
126                }
127
128                catch ( IOException ioException ) {
129                   System.exit( 1 );
130                }
131
132                System.exit( 0 );
133             }
134
135          } // end anonymous inner class
136
137       ); // end call to addActionListener
138
139       fileMenu.add( newItem );
140       fileMenu.add( updateItem );
141       fileMenu.add( deleteItem );
142       fileMenu.addSeparator();
143       fileMenu.add( inventoryItem );
144       fileMenu.addSeparator();
145       fileMenu.add( exitItem );
146
147       setDefaultCloseOperation( WindowConstants.DO_NOTHING_ON_CLOSE );
148
149       setSize( 370, 330 );
150       setVisible( true );
151
152    }  // end constructor
153
154    public static void main( String args[] )
155    {
156       new HardwareStore();
157    }
158
159 }  // end class HardwareStore
160
161 // class for updating records
162 class UpdateDialog extends JInternalFrame implements ActionListener{
163
164    private RandomAccessFile file;
165    private JTextField recordField, toolField, quantityField, costField;
166    private JLabel recordLabel, toolLabel, quantityLabel, costLabel;
167    private JButton saveButton, cancelButton;
```

```
168    private Record data;
169    private int recordNumber;
170
171    public UpdateDialog( RandomAccessFile updateFile )
172    {
173       super( "Update Record" );
174
175       file = updateFile;
176       data = new Record();
177
178       // set up GUI
179       recordLabel = new JLabel( "Record Number" );
180       recordField= new JTextField( 10 );
181       toolLabel = new JLabel( "Tool Name" );
182       toolField = new JTextField( 10 );
183       toolField.setEditable( false );
184       quantityLabel = new JLabel( "Quantity" );
185       quantityField = new JTextField( 10 );
186       quantityField.setEditable( false );
187       costLabel = new JLabel( "Cost" );
188       costField = new JTextField( 10 );
189       costField.setEditable( false );
190
191       saveButton = new JButton( "Save Changes" );
192       cancelButton = new JButton( "Cancel" );
193
194       recordField.addActionListener( this );
195       saveButton.addActionListener( this );
196       cancelButton.addActionListener( this );
197
198       Container container = getContentPane();
199       container.setLayout( new GridLayout( 5, 2 ) );
200       container.add( recordLabel );
201       container.add( recordField );
202       container.add( toolLabel );
203       container.add( toolField );
204       container.add( quantityLabel );
205       container.add( quantityField );
206       container.add( costLabel );
207       container.add( costField );
208       container.add( saveButton );
209       container.add( cancelButton );
210
211       setSize( 300, 150 );
212       setVisible( false );
213    }
214
215    // process events
216    public void actionPerformed( ActionEvent event )
217    {
218       // text field
219       if ( event.getSource() == recordField )  {
220          recordNumber = Integer.parseInt( recordField.getText() );
221
```

```
222          // check if record number is within range
223          if ( recordNumber < 1 || recordNumber > 100 ) {
224             JOptionPane.showMessageDialog( this, "Invalid record number",
225                "Account Error", JOptionPane.ERROR_MESSAGE );
226             clear();
227             return;
228          }
229
230          // set file-pointer to appropriate location
231          try {
232             file.seek( ( recordNumber - 1 ) * data.SIZE );
233             data.read( file );
234          }
235
236          catch ( IOException ioException ) {
237             JOptionPane.showMessageDialog( this,
238                "Error while reading from file",
239                "Read Error", JOptionPane.ERROR_MESSAGE );
240          }
241
242          // retrieve current record information
243          if ( data.getRecordNumber() != 0 ) {
244             toolField.setText( data.getToolName() );
245             toolField.setEditable( true );
246             quantityField.setText(
247                Integer.toString( data.getQuantity() ) );
248             quantityField.setEditable( true );
249             costField.setText( String.valueOf( data.getCost() ) );
250             costField.setEditable( true );
251          }
252
253          // no tool for that record number
254          else {
255             recordField.setText( recordNumber + " does not exist" );
256             toolField.setText( "" );
257             quantityField.setText( "" );
258             costField.setText( "" );
259          }
260       }
261
262       // save button
263       else if ( event.getSource() == saveButton ) {
264
265          // update record information
266          try {
267             file.seek( ( recordNumber - 1 ) * data.SIZE );
268             data.setRecordNumber( recordNumber );
269             data.setToolName( toolField.getText() );
270             data.setQuantity(
271                Integer.parseInt( quantityField.getText() ) );
272             data.setCost( ( Double.valueOf(
273                costField.getText() ) ).doubleValue() );
274             data.write( file );
275          }
```

```
276
277          catch ( IOException ioException ) {
278             JOptionPane.showMessageDialog( this,
279                "Error while writing to file",
280                "Write Error", JOptionPane.ERROR_MESSAGE );
281             return;
282          }
283
284          setVisible( false );
285          clear();
286       }
287
288       // cancel button
289       else if ( event.getSource() == cancelButton ) {
290          setVisible( false );
291          clear();
292       }
293
294    }  // end method actionPerformed
295
296    // clear text fields
297    private void clear()
298    {
299       recordField.setText( "" );
300       toolField.setText( "" );
301       quantityField.setText( "" );
302       costField.setText( "" );
303    }
304
305 }  // end class UpdateDialog
306
307 // class for creating new records
308 class NewDialog extends JInternalFrame implements ActionListener {
309
310    private RandomAccessFile file;
311    private JTextField recordField, toolField, quantityField, costField;
312    private JLabel recordLabel, toolLabel, quantityLabel, costLabel;
313    private JButton saveButton, cancelButton;
314    private Record data;
315    private int recordNumber;
316
317    public NewDialog( RandomAccessFile newFile )
318    {
319       super( "New Record" );
320
321       file = newFile;
322       data = new Record();
323
324       // set up GUI
325       recordLabel = new JLabel( "Record Number" );
326       recordField= new JTextField( 10 );
327       toolLabel = new JLabel( "Tool Name" );
328       toolField = new JTextField( 10 );
329       quantityLabel = new JLabel( "Quantity" );
```

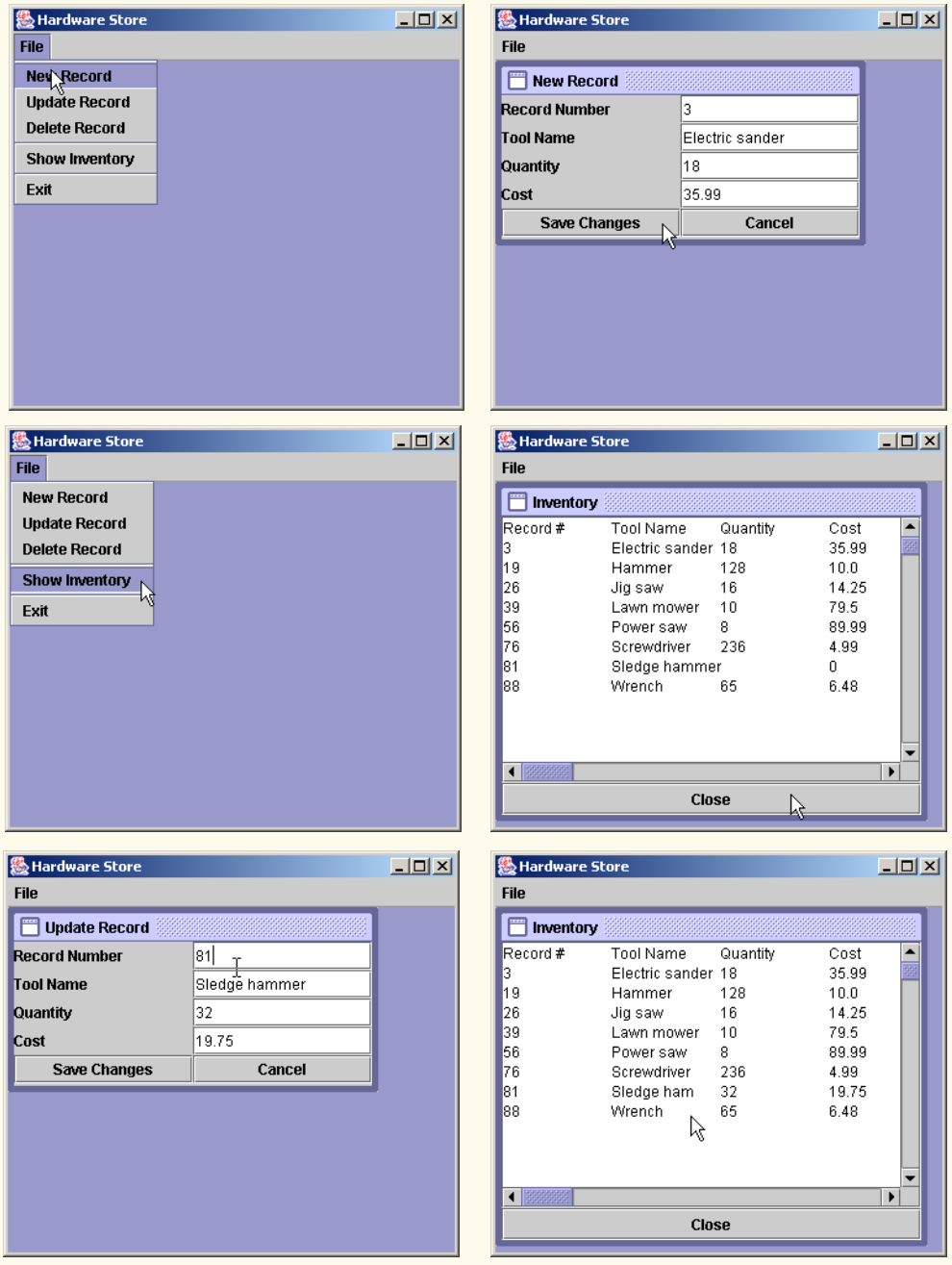
```
330         quantityField = new JTextField( 10 );
331         costLabel = new JLabel( "Cost" );
332         costField = new JTextField( 10 );
333
334         saveButton = new JButton( "Save Changes" );
335         cancelButton = new JButton( "Cancel" );
336
337         saveButton.addActionListener( this );
338         cancelButton.addActionListener( this );
339
340         Container container = getContentPane();
341         container.setLayout( new GridLayout( 5, 2 ) );
342         container.add( recordLabel );
343         container.add( recordField );
344         container.add( toolLabel );
345         container.add( toolField );
346         container.add( quantityLabel );
347         container.add( quantityField );
348         container.add( costLabel );
349         container.add( costField );
350         container.add( saveButton );
351         container.add( cancelButton );
352
353         setSize( 300, 150 );
354         setVisible( false );
355      }
356
357      // process events
358      public void actionPerformed( ActionEvent event )
359      {
360         // save button
361         if ( event.getSource() == saveButton ) {
362
363            recordNumber = Integer.parseInt( recordField.getText() );
364
365            // check if record number is within range
366            if ( recordNumber < 1 || recordNumber > 100 ) {
367               JOptionPane.showMessageDialog( this, "Invalid record number",
368                  "Account Error", JOptionPane.ERROR_MESSAGE );
369               clear();
370               return;
371            }
372
373            // set file-pointer to appropriate location
374            try {
375               file.seek( ( recordNumber - 1 ) * data.SIZE );
376               data.read( file );
377            }
378
379            catch ( IOException ioException ) {
380               JOptionPane.showMessageDialog( this,
381                  "Error while reading from file",
382                  "Read Error", JOptionPane.ERROR_MESSAGE );
383            }
```

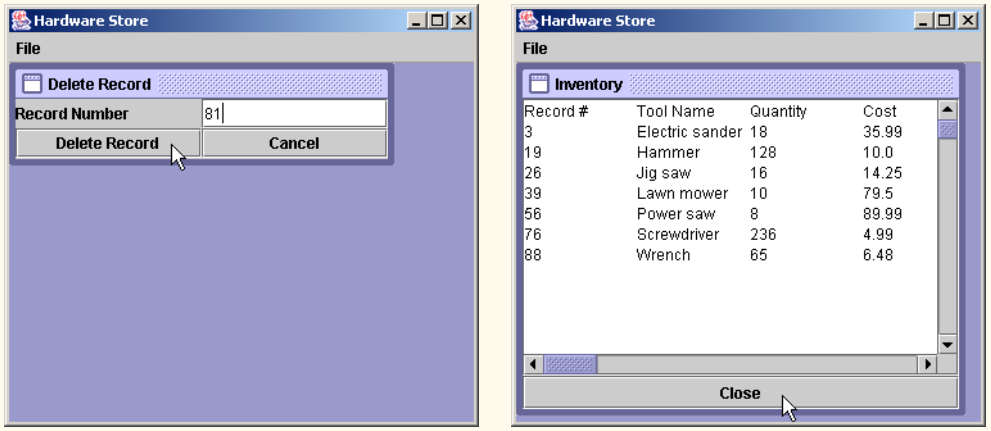
```
384
385          // record number already exists
386          if ( data.getRecordNumber() != 0 ) {
387             JOptionPane.showMessageDialog( this,
388                "Record number already exists",
389                "Duplicate record number", JOptionPane.ERROR_MESSAGE );
390             clear();
391             return;
392          }
393
394          // set values of new record and write to file
395          try {
396             data.setRecordNumber( recordNumber );
397             data.setToolName( toolField.getText() );
398             data.setQuantity(
399                Integer.parseInt( quantityField.getText() ) );
400             data.setCost( ( new Double(
401                costField.getText() ) ).doubleValue() );
402             file.seek( ( recordNumber - 1 ) * data.SIZE );
403             data.write( file );
404          }
405
406          catch ( IOException ioException ) {
407             JOptionPane.showMessageDialog( this,
408                "Error while writing to file",
409                "Write Error", JOptionPane.ERROR_MESSAGE );
410             return;
411          }
412
413          setVisible( false );
414          clear();
415       }
416
417       // cancel button
418       else if ( event.getSource() == cancelButton ) {
419          setVisible( false );
420          clear();
421       }
422
423    }  // end method actionPerformed
424
425    // clear text fields
426    private void clear()
427    {
428       recordField.setText( "" );
429       toolField.setText( "" );
430       quantityField.setText( "" );
431       costField.setText( "" );
432    }
433
434 }  // end class NewDialog
435
436 // class for deleting records
437 class DeleteDialog extends JInternalFrame implements ActionListener {
```

```
438
439    private RandomAccessFile file;
440    private JTextField recordField;
441    private JLabel recordLabel;
442    private JButton cancelButton, deleteButton;
443    private Record data;
444    private int recordNumber;
445
446    public DeleteDialog( RandomAccessFile deleteFile )
447    {
448       super( "Delete Record" );
449
450       file = deleteFile;
451       data = new Record();
452
453       recordLabel = new JLabel( "Record Number" );
454       recordField = new JTextField( 10 );
455       deleteButton = new JButton( "Delete Record" );
456       cancelButton = new JButton( "Cancel" );
457
458       cancelButton.addActionListener( this );
459       deleteButton.addActionListener( this );
460
461       Container container = getContentPane();
462       container.setLayout( new GridLayout( 2, 2 ) );
463       container.add( recordLabel );
464       container.add( recordField );
465       container.add( deleteButton );
466       container.add( cancelButton );
467
468       setSize( 300, 80 );
469       setVisible( false );
470    }
471
472    // process events
473    public void actionPerformed( ActionEvent event )
474    {
475       // delete button
476       if ( event.getSource() == deleteButton ) {
477          recordNumber =
478             Integer.parseInt( recordField.getText() );
479
480          if ( recordNumber < 1 || recordNumber > 100 ) {
481             recordField.setText( "Invalid part number" );
482             return;
483          }
484
485          try {
486             file.seek( ( recordNumber - 1 ) * data.SIZE );
487             data.read( file );
488          }
489
490          catch ( IOException ioException ) {
491             recordField.setText( "Error reading file" );
492          }
```

```
493
494          // no such record
495          if ( data.getRecordNumber() == 0 ) {
496             recordField.setText( recordNumber + " does not exist" );
497             return;
498          }
499
500          try {
501             file.seek( ( recordNumber - 1 ) * data.SIZE );
502             data.setRecordNumber( 0 );
503             data.setToolName( "" );
504             data.setQuantity( 0 );
505             data.setCost( 0.0 );
506             data.write( file );
507          }
508
509          catch ( IOException ioException ) {
510             JOptionPane.showMessageDialog( this,
511                "Error while writing to file",
512                "Write Error", JOptionPane.ERROR_MESSAGE );
513             return;
514          }
515
516          setVisible( false );
517          recordField.setText( "" );
518       }
519
520       // cancel button
521       else if ( event.getSource() == cancelButton ) {
522          setVisible( false );
523          recordField.setText( "" );
524       }
525
526    }  // end method actionPerformed
527
528 }  // end class DeleteDialog
529
530 // class for displaying records
531 class InventoryDialog extends JInternalFrame {
532
533    private RandomAccessFile file;
534    private JTextArea displayArea;
535    private JButton closeButton;
536    private JScrollPane scroller;
537    private Record data;
538
539    public InventoryDialog( RandomAccessFile deleteFile )
540    {
541       super( "Inventory" );
542
543       file = deleteFile;
544       data = new Record();
545
546       displayArea = new JTextArea( 300, 200 );
```

```
547         displayArea.setEditable( false );
548         scroller = new JScrollPane( displayArea );
549
550         closeButton = new JButton( "Close" );
551         closeButton.addActionListener(
552
553            new ActionListener() { // anonymous inner class
554
555               public void actionPerformed( ActionEvent event )
556               {
557                  setVisible( false );
558               }
559
560            } // end anonymous inner class
561
562         ); // end call to addActionListener
563
564         Container container = getContentPane();
565         container.setLayout( new BorderLayout() );
566         container.add( scroller, BorderLayout.CENTER );
567         container.add( closeButton, BorderLayout.SOUTH );
568
569         setSize( 350, 275 );
570         setVisible( false );
571      }
572
573      // display tool inventory
574      public void display()
575      {
576         displayArea.setText( "Record #\tTool Name\tQuantity\tCost\n" );
577
578         try {
579            for ( int i = 0; i < 100; i++ ) {
580
581               file.seek( ( i ) * data.SIZE );
582               data.read( file );
583
584               if ( data.getRecordNumber() != 0 )
585                  displayArea.append( data.getRecordNumber() + "\t" +
586                     data.getToolName() + "\t" + data.getQuantity() + "\t" +
587                     data.getCost() + "\n" );
588            }
589         }
590
591         // error reading from file
592         catch ( IOException ioException ) {
593            JOptionPane.showMessageDialog( this,
594               "Error during read from file",
595               "Read Error", JOptionPane.ERROR_MESSAGE );
596         }
597      }
598
599   }  // end class InventoryDialog
```





17.11 *(Telephone-Number Word Generator)* Standard telephone keypads contain the digits zero through nine. The numbers two through nine each have three letters associated with them. (See Fig. 17.28.) Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2377 might use the correspondence indicated in Fig. 17.28 to develop the seven-letter word "NUMBERS." Each seven-letter word corresponds to exactly one seven-digit telephone number. The restaurant wishing to increase its takeout business could surely do so with the number 825-3688 (i.e., "TAKEOUT").

Each seven-letter phone number corresponds to many separate seven-letter words. Unfortunately, most of these words represent unrecognizable juxtapositions of letters. It is possible, however, that the owner of a barbershop would be pleased to know that the shop's telephone number, 424-7288, corresponds to "HAIRCUT." The owner of a liquor store would, no doubt, be delighted to find that the

Account number	Dollar amount
300	83.89
700	80.78
700	1.53

Fig. 17.26 Additional transaction records.

Record #	Tool name	Quantity	Cost
3	Electric sander	18	35.99
19	Hammer	128	10.00
26	Jigsaw	16	14.25

Fig. 17.27 Data for Exercise 17.10. (Part 1 of 2.)

Record #	Tool name	Quantity	Cost
39	Lawn mower	10	79.50
56	Power saw	8	89.99
76	Screwdriver	236	4.99
81	Sledgehammer	32	19.75
88	Wrench	65	6.48

Fig. 17.27 Data for Exercise 17.10. (Part 2 of 2.)

Digit	Letters
2	A B C
3	D E F
4	G H I
5	J K L
6	M N O
7	P R S
8	T U V
9	W X Y

Fig. 17.28 Telephone keypad digits and letters.

store's number, 233-7226, corresponds to "BEERCAN." A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters "PETCARE." An automotive dealership would be pleased to know that the dealership number, 639-2277, corresponds to "NEWCARS."

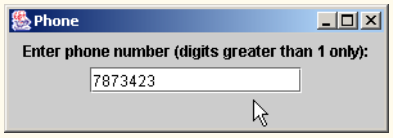
Write a program that, given a seven-digit number, uses a `PrintStream` object to write to a file every possible seven-letter word combination corresponding to that number. There are 2187 (3^7) such combinations. Avoid phone numbers with the digits 0 and 1.

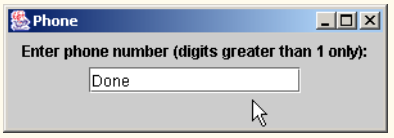
ANS:

```
1  // Exercise 17.11 Solution: Phone.java
2  // Note: phone number must be input in the form #######.
3  // Only the digits 2 thru 9 are recognized.
4  import java.io.*;
5  import java.awt.*;
6  import java.awt.event.*;
7  import javax.swing.*;
8
9  public class Phone extends JFrame
10 {
11    private int phoneNumber[];
12    private JTextField input;
13    private JLabel prompt;
14
```

```
15    public Phone()
16    {
17       super( "Phone" );
18       input = new JTextField( 15 );
19       input.addActionListener(
20
21          new ActionListener() { // anonymous inner class
22
23             public void actionPerformed( ActionEvent event )
24             {
25                calculate(); // calculate character sequences
26             }
27
28           } // end anonymous inner class
29
30       ); // end call to addActionListener
31
32       prompt = new JLabel(
33          "Enter phone number (digits greater than 1 only):" );
34
35       Container container = getContentPane();
36       container.setLayout( new FlowLayout() );
37       container.add( prompt );
38       container.add( input );
39
40       setSize( 300, 100 );
41       setVisible( true );
42    }
43
44    // output letter combinations to file
45    private void calculate()
46    {
47       String letters[][] = { { "" },
48          { "" },            { "A", "B", "C" }, { "D", "E", "F" },
49          { "G", "H", "I" }, { "J", "K", "L" }, { "M", "N", "O" },
50          { "P", "R", "S" }, { "T", "U", "V" }, { "W", "X", "Y" } };
51
52       long phoneNumber = Long.parseLong( input.getText() );
53       int digits[] = new int[ 7 ];
54       for ( int i = 0; i < 7; i++ ) {
55          digits[i] = ( int )(phoneNumber % 10);
56          phoneNumber /= 10;
57       }
58
59       PrintStream output = null;
60
61       try {
62          output = new PrintStream( new FileOutputStream( "phone.dat" ) );
63       }
64
65       catch( IOException exception ) {
66          JOptionPane.showMessageDialog( null, exception.toString(),
67             "Exception", JOptionPane.ERROR_MESSAGE );
68
```

```
69             System.exit( 1 );
70          }
71
72          input.setText( "Please wait..." );
73
74          int loop[] = new int[ 7 ];
75
76          // output all possible combinations
77          for ( loop[ 0 ] = 0; loop[ 0 ] <= 2; loop[ 0 ]++ )
78             for ( loop[ 1 ] = 0; loop[ 1 ] <= 2; loop[ 1 ]++ )
79                for ( loop[ 2 ] = 0; loop[ 2 ] <= 2; loop[ 2 ]++ )
80                   for ( loop[ 3 ] = 0; loop[ 3 ] <= 2; loop[ 3 ]++ )
81                      for ( loop[ 4 ] = 0; loop[ 4 ] <= 2; loop[ 4 ]++ )
82                         for ( loop[ 5 ] = 0; loop[ 5 ] <= 2; loop[ 5 ]++ )
83                            for ( loop[ 6 ] = 0; loop[ 6 ] <= 2; loop[6]++ ) {
84                               for ( int i = 6; i >= 0; i-- )
85                                  output.print(
86                                     letters[ digits[ i ] ][ loop[ i ] ] );
87
88                               output.println();
89                            }
90
91          input.setText( "Done" );
92
93          output.close(); // close output stream
94
95       } // end method actionPerformed
96
97       public static void main( String args[] )
98       {
99          Phone application = new Phone();
100         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
101      }
102
103   }  // end class Phone
```





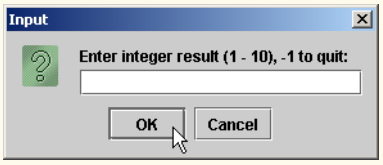

```
PTPDGAD
RTPDGAD
STPDGAD
PUPDGAD
RUPDGAD
SUPDGAD
PVPDGAD
...
```

17.12 Figure 7.8 contains an array of survey responses that is hard coded into the program. Suppose we wish to process survey results that are stored in a file. This exercise requires two separate programs. First, create an application that prompts the user for survey responses and outputs each response to a file. Use a `DataOutputStream` wrapping a `FileOutputStream` to create a file called `numbers.dat`. Each integer should be written using `DataOutputStream` method `writeInt`. Then modify the program of Fig. 7.8 to read the survey responses from `numbers.dat`. The responses should be read from the file by using a `DataInputStream` wrapping a `FileInputStream`. Method `readInt` from class `DataInputStream` should be used to input one integer from the file at a time. The program should continue to read responses until it reaches the end of file, at which point method `readInt` throws an `EOFException`. The results should be output to a text file and displayed in a window. Use a `PrintWriter` object wrapping a `FileWriter` to write to the file `"output.txt"`. Use method `print` from class `PrintWriter` to output a string to the file.

ANS:

```
1  // Exercise 17.12 Solution: CreateResults.java
2  // Create poll results and output them to a file.
3  import javax.swing.*;
4  import java.io.*;
5
6  public class CreateResults {
7
8     private static int getValue()
9     {
10       // prompt the user for input
11       String input = JOptionPane.showInputDialog(
12          "Enter integer result (1 - 10), -1 to quit:" );
13
14       try {
15          return Integer.parseInt( input ); // try to get an integer
16       }
17
18       // if its not an integer
19       catch( NumberFormatException format ) {
20          return 11; // return an invalid value
21       }
22    }
23
24    public static void main( String args[] ) {
25       try {
26          // create the output stream
27          DataOutputStream pollNumbers =
28             new DataOutputStream( new FileOutputStream( "numbers.dat" ) );
29
30          int pollValue = getValue(); // get a number from the user
31
32          // test for the sentinel value
33          while( pollValue != -1 ) {
34
35             // if the number is valid
36             if ( pollValue > 0 && pollValue < 11 )
37                pollNumbers.writeInt( pollValue ); // write the value
38
39             pollValue = getValue(); // get another value
```

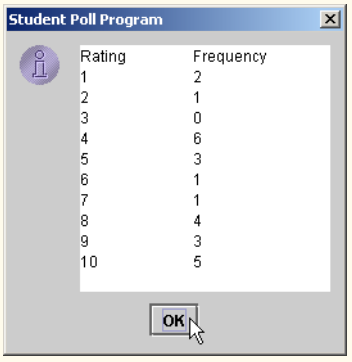
```
40          }
41
42          pollNumbers.close(); // close the file
43       }
44
45       catch( IOException ioException ) {
46          JOptionPane.showMessageDialog( null, "Error with the file",
47             "IOError", JOptionPane.ERROR_MESSAGE );
48       }
49
50       System.exit( 0 );
51
52    } // end main
53
54 } // end class CreateResults
```



ANS:

```
1  // Exercise 17.12 Solution: StudentPoll.java
2  // Read poll results from a file and output ratings.
3  import javax.swing.*;
4  import java.io.*;
5
6  public class StudentPoll {
7
8     public static void main( String args[] )
9     {
10       int frequency[] = new int[ 11 ];
11
12       try {
13          DataInputStream pollNumbers =
14             new DataInputStream( new FileInputStream( "numbers.dat" ) );
15
16          try {
17             // for each answer, use that value as subscript to
18             // determine element to increment
19             while( true )
20                ++frequency[ pollNumbers.readInt() ];
21          }
22
23          catch( EOFException eof ) {
24          }
25
26          String output = "Rating\tFrequency\r\n";
27
```

```
28            // append frequencies to String output
29            for ( int rating = 1; rating < frequency.length; rating++ )
30               output += rating + "\t" + frequency[ rating ] + "\r\n";
31
32            JTextArea outputArea = new JTextArea();
33            outputArea.setText( output );
34
35            PrintWriter writer =
36               new PrintWriter( new FileWriter( "output.txt" ) );
37            writer.print( output );
38            writer.close();
39
40            JOptionPane.showMessageDialog( null, outputArea,
41               "Student Poll Program", JOptionPane.INFORMATION_MESSAGE );
42
43            pollNumbers.close();
44
45            System.exit( 0 );
46
47         }
48
49         catch( IOException io ) {
50            JOptionPane.showMessageDialog( null, "File Error", "IO Error",
51               JOptionPane.ERROR_MESSAGE );
52            System.exit( 1 );
53         }
54
55      } // end main
56
57   } // end class StudentPoll
```



17.13 Modify Exercise 14.17 to allow the user to output the array of `MyShape` objects to a file and to read in a file. Use an `ObjectOutputStream` wrapping a `FileOutputStream` to write to the file and an `ObjectInputStream` wrapping a `FileInputStream` to read from the file. Output the number of shapes currently stored in the array with method `writeInt`. Then output the array of shapes. The integer will be used to keep track of the next available position in the array so that more shapes can be added when the array is read back from the file. When reading from the file, read the integer with method `readInt`. Note that the integer and array must be read in the same order as they are written.

ANS:

```
1  // Exercise 17.13 Solution: DrawFrame.java
2  // This class creates a frame that allows the user to draw shapes.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7  import java.io.*;
8
9  public class DrawFrame extends JInternalFrame {
10    private DrawPanel3 drawPanel;
11    private JComboBox choose;
12    private DrawProgram parent;
13    private String name;
14
15    public DrawFrame( String title, DrawProgram caller ) {
16       super( title, true, true, true, true );
17
18       parent = caller;
19       name = title;
20
21       MyColorChooser4 colorChooser = new MyColorChooser4();
22
23       // when the red slider is altered, update the shape
24       final JSlider red = colorChooser.getRedSlider();
25       red.addChangeListener(
26
27          new ChangeListener() { // anonymous inner class
28
29             // handle slider event
30             public void stateChanged( ChangeEvent event )
31             {
32                drawPanel.setRed( red.getValue() );
33             }
34
35          }  // end anonymous inner class
36
37       ); // end call to addChangeListener
38
39       // when the red text field is altered, update the shape
40       final JTextField redField = colorChooser.getRedDisplay();
41       redField.addActionListener(
42
43          new ActionListener() { // anonymous inner class
44
45             // handle text field event
46             public void actionPerformed( ActionEvent event )
47             {
48                drawPanel.setRed(
49                   Integer.parseInt( redField.getText() ) );
50             }
51
52          }  // end anonymous inner class
```

```
53
54          ); // end call to addActionListener
55
56          // when the green slider is altered, update the shape
57          final JSlider green = colorChooser.getGreenSlider();
58          green.addChangeListener(
59
60             new ChangeListener() { // anonymous inner class
61
62                // handle slider event
63                public void stateChanged( ChangeEvent event )
64                {
65                   drawPanel.setGreen( green.getValue() );
66                }
67
68             }  // end anonymous inner class
69
70          ); // end call to addChangeListener
71
72          // when the green text field is altered, update the shape
73          final JTextField greenField = colorChooser.getGreenDisplay();
74          greenField.addActionListener(
75
76             new ActionListener() { // anonymous inner class
77
78                // handle text field event
79                public void actionPerformed( ActionEvent event )
80                {
81                   drawPanel.setGreen(
82                      Integer.parseInt( greenField.getText() ) );
83                }
84
85             }  // end anonymous inner class
86
87          ); // end call to addActionListener
88
89          // when the blue slider is altered, update the shape
90          final JSlider blue = colorChooser.getBlueSlider();
91          blue.addChangeListener(
92
93             new ChangeListener() { // anonymous inner class
94
95                // handle slider event
96                public void stateChanged( ChangeEvent event )
97                {
98                   drawPanel.setBlue( blue.getValue() );
99                }
100
101            }  // end anonymous inner class
102
103         ); // end call to addChangeListener
104
105         // when the blue text field is altered, update the shape
106         final JTextField blueField = colorChooser.getBlueDisplay();
```

```
107       blueField.addActionListener(
108
109          new ActionListener() { // anonymous inner class
110
111             // handle text field event
112             public void actionPerformed( ActionEvent event )
113             {
114                drawPanel.setBlue(
115                   Integer.parseInt( blueField.getText() ) );
116             }
117
118          }  // end anonymous inner class
119
120       ); // end call to addActionListener
121
122       drawPanel = new DrawPanel3();
123       drawPanel.addMouseListener(
124
125          new MouseAdapter() { // anonymous inner class
126
127             // create the chosen shape
128             public void mousePressed( MouseEvent event )
129             {
130                   drawPanel.createShape( event.getX(), event.getY(),
131                      ( String )choose.getSelectedItem() );
132             }
133
134             // finish creating the shape
135             public void mouseReleased( MouseEvent event )
136             {
137                   drawPanel.finishShape();
138             }
139
140          } // end anonymous inner class
141
142       ); // end call to addMouseListener
143
144       drawPanel.addMouseMotionListener(
145
146          new MouseMotionAdapter() { // anonymous inner class
147
148             // resize the shape
149             public void mouseDragged( MouseEvent event )
150             {
151                   drawPanel.resizeShape( event.getX(), event.getY() );
152             }
153
154          } // end anonymous inner class
155
156       ); // end call to addMouseMotionListener
157
158       ShapePanel shapePanel = new ShapePanel();
159       JCheckBox fill = shapePanel.getFill();
160       fill.addItemListener(
```

```
161
162          new ItemListener() { // anonymous inner class
163
164             // toggle the fill field in the drawing panel
165             public void itemStateChanged( ItemEvent event ) {
166                if ( event.getStateChange() == ItemEvent.SELECTED )
167                   drawPanel.setFill( true );
168                else
169                   drawPanel.setFill( false );
170             }
171
172          } // end anonymous inner class
173
174       ); // end call to addItemListener
175
176       choose = shapePanel.getChooser();
177       choose.addItemListener(
178
179          new ItemListener() { // anonymous inner class
180
181             // change the shape that is displayed
182             public void itemStateChanged( ItemEvent event )
183             {
184                drawPanel.setShape( (String)choose.getSelectedItem() );
185             }
186
187          } // end anonymous inner class
188
189       ); // end call to addItemListener
190
191       JMenuBar menuBar = new JMenuBar();
192       JMenu fileMenu = new JMenu( "File" );
193       menuBar.add( fileMenu );
194       JMenuItem saveItem = new JMenuItem( "Save" );
195       fileMenu.add( saveItem );
196       JMenuItem loadItem = new JMenuItem( "Load" );
197       fileMenu.add( loadItem );
198       JMenuItem exitItem = new JMenuItem( "Exit" );
199       fileMenu.add( exitItem );
200
201       saveItem.addActionListener(
202
203          new ActionListener() { // anonymous inner class
204
205             public void actionPerformed( ActionEvent event )
206             {
207                JFileChooser fileChooser = new JFileChooser();
208                fileChooser.setFileSelectionMode(
209                   JFileChooser.FILES_ONLY );
210                int result = fileChooser.showSaveDialog( DrawFrame.this );
211
212                if ( result == JFileChooser.CANCEL_OPTION )
213                   return;
214
```

```
215                File fileName = fileChooser.getSelectedFile();
216
217                try {
218                   ObjectOutputStream output = new ObjectOutputStream(
219                      new FileOutputStream( fileName ) );
220                   output.writeInt( drawPanel.getCount() );
221                   output.writeObject( drawPanel.getShapes() );
222                   output.close();
223                }
224
225                catch( IOException io ) {
226                   JOptionPane.showMessageDialog( null,
227                      "Error with the file", "IO Error",
228                      JOptionPane.ERROR_MESSAGE );
229                   System.exit( 1 );
230                }
231             }
232
233          } // end anonymous inner class
234
235       ); // end call to addActionListener
236
237       loadItem.addActionListener(
238
239          new ActionListener() { // anonymous inner class
240
241             public void actionPerformed( ActionEvent event )
242             {
243                JFileChooser fileChooser = new JFileChooser();
244                fileChooser.setFileSelectionMode(
245                   JFileChooser.FILES_ONLY );
246                int result = fileChooser.showOpenDialog( DrawFrame.this );
247
248                if ( result == JFileChooser.CANCEL_OPTION )
249                   return;
250
251                File fileName = fileChooser.getSelectedFile();
252
253                try {
254                   ObjectInputStream input = new ObjectInputStream(
255                      new FileInputStream( fileName ) );
256                   drawPanel.setCount( input.readInt() );
257                   drawPanel.setShapes( ( MyShape[] )input.readObject() );
258                   input.close();
259                }
260                catch( Exception e ) {
261                   JOptionPane.showMessageDialog( null,
262                      "Error with the file", "IO Error",
263                      JOptionPane.ERROR_MESSAGE );
264                   System.exit( 1 );
265                }
266             }
267
268          } // end anonymous inner class
```

```
269
270       ); // end call to addActionListener
271
272       exitItem.addActionListener(
273
274          new ActionListener() { // anonymous inner class
275
276             public void actionPerformed( ActionEvent event )
277             {
278                try {
279                   DrawFrame.this.setClosed( true );
280                }
281
282                catch( Exception e ) {
283                }
284             }
285
286          } // end anonymous inner class
287
288       ); // end call to addActionListener
289
290       setJMenuBar( menuBar );
291
292       Container container = getContentPane();
293       container.setLayout( new BorderLayout() );
294       container.add( shapePanel, BorderLayout.NORTH );
295       container.add( drawPanel, BorderLayout.CENTER );
296       container.add( colorChooser, BorderLayout.SOUTH );
297
298       addInternalFrameListener(
299
300          new InternalFrameAdapter() { // anonymous inner class
301
302             // when the frame is closed, alert the DrawProgram
303             public void internalFrameClosed( InternalFrameEvent e )
304             {
305                parent.frameClosed( name );
306             }
307          } // end anonymous inner class
308
309       ); // end call to addInternalFrameListener
310
311       setSize( 300, 300 );
312
313    } // end constructor
314
315 } // end class DrawFrame
```

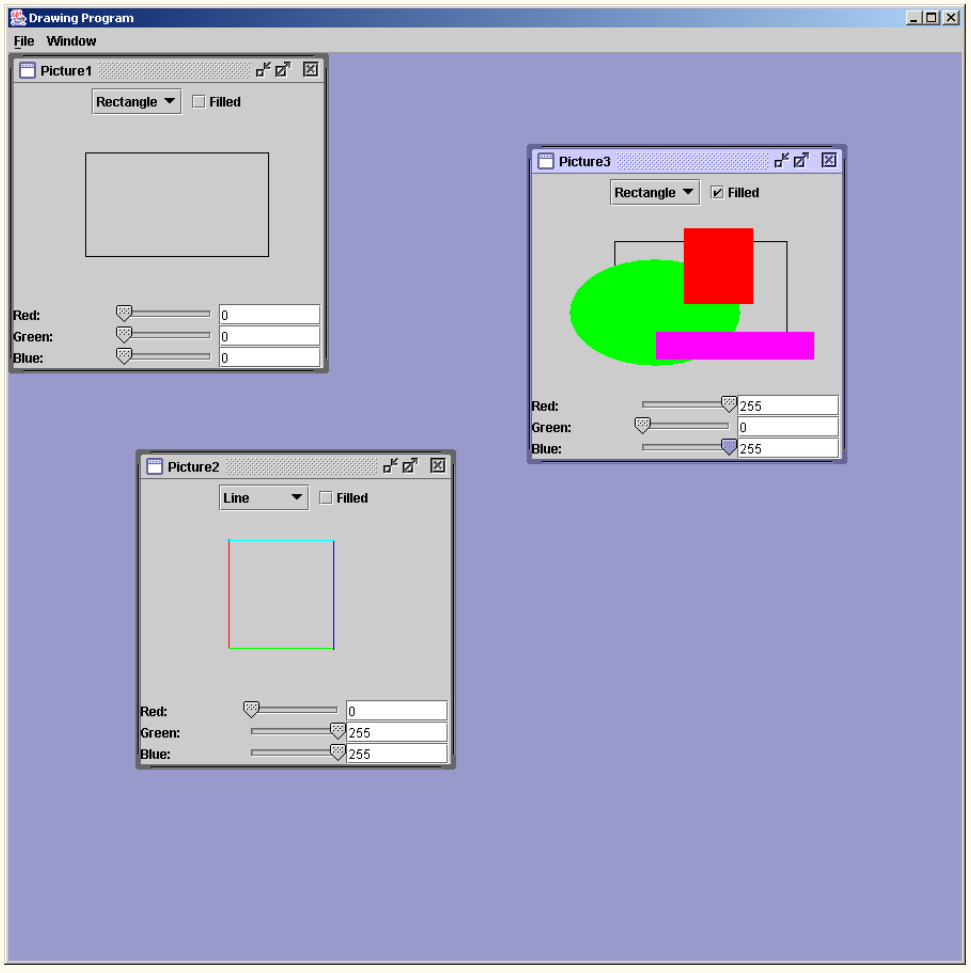
ANS:

```
1  // Exercise 17.13 Solution: DrawProgram.java
2  // This class creates the desktop, manages menu items and creates frames.
3  import java.awt.*;
```

```
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class DrawProgram extends JFrame {
9     private static final int MAX_WINDOWS = 20;
10    private JInternalFrame[] frames;
11    private JMenuItem[] frameItems;
12    private int count = 0;
13    private JDesktopPane desktop;
14    private JMenuItem newItem;
15    private JMenu windowMenu;
16
17    public DrawProgram()
18    {
19       super( "Drawing Program" );
20
21       desktop = new JDesktopPane();
22       getContentPane().add( desktop );
23
24       frames = new JInternalFrame[ MAX_WINDOWS ];
25       frameItems = new JMenuItem[ MAX_WINDOWS ];
26
27       JMenuBar menuBar = new JMenuBar();
28       JMenu fileMenu = new JMenu( "File" );
29       fileMenu.setMnemonic( 'f' );
30
31       // create a new frame
32       newItem = new JMenuItem( "New" );
33       newItem.setMnemonic( 'n' );
34       fileMenu.add( newItem );
35       newItem.addActionListener(
36
37          new ActionListener() { // anonymous inner class
38
39             public void actionPerformed( ActionEvent event ) {
40
41                // the title of the window
42                String title = "Picture" + ( count + 1 );
43
44                // create the frame
45                frames[ count ] = new DrawFrame( title, DrawProgram.this );
46                desktop.add( frames[ count ] );
47                frames[ count ].setVisible( true );
48
49                // create the menu item
50                frameItems[ count ] = new JMenuItem( title );
51                windowMenu.add( frameItems[ count ] );
52                frameItems[ count ].addActionListener(
53
54                   new ActionListener() { // anonymous inner class
55
56                      public void actionPerformed( ActionEvent event ) {
57                         String caller = event.getActionCommand();
```

```
58                          int number =
59                             Integer.parseInt( caller.substring( 7 ) );
60
61                          // set the selected frame to the front
62                          frames[ number - 1 ].toFront();
63                       }
64
65                    } // end anonymous inner class
66
67                 ); // end call to addActionListener
68
69                 // increment the count
70                 count++;
71
72                 // if we hit the maximum number of windows,
73                 // disable the new menu item
74                 if ( count == MAX_WINDOWS )
75                    newItem.setEnabled( false );
76
77              }
78
79           } // end anonymous inner class
80
81        ); // end call to addActionListener
82
83        // exit the application
84        JMenuItem exitItem = new JMenuItem( "Exit" );
85        exitItem.setMnemonic( 'e' );
86        fileMenu.add( exitItem );
87        menuBar.add( fileMenu );
88        exitItem.addActionListener(
89
90           new ActionListener() { // anonymous inner class
91
92              public void actionPerformed( ActionEvent event ) {
93                 System.exit( 0 );
94              }
95
96           } // end anonymous inner class
97
98        ); // end call to addActionListener
99
100       // this will bring the selected window to the front
101       windowMenu = new JMenu( "Window" );
102       menuBar.add( windowMenu );
103
104       setJMenuBar( menuBar );
105
106       setSize( 900, 900 ); // set the window's size
107       setVisible( true );
108
109    } // end constructor
110
```

```
111    // called when a child frame is closed
112    public void frameClosed( String window )
113    {
114       int number = Integer.parseInt( window.substring( 7 ) );
115
116       // remove the menu item and set the reference to null
117       windowMenu.remove( frameItems[ number - 1 ] );
118       frameItems[ number - 1 ] = null;
119    }
120
121    public static void main( String args[] )
122    {
123       DrawProgram application = new DrawProgram();
124       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
125    }
126
127 } // end class DrawProgram
```



18

Networking

Objectives

- To understand Java networking with URLs, sockets and datagrams.
- To implement Java networking applications by using sockets and datagrams.
- To understand how to implement Java clients and servers that communicate with one another.
- To understand how to implement network-based collaborative applications.
- To construct a multithreaded server.

If the presence of electricity can be made visible in any part of a circuit, I see no reason why intelligence may not be transmitted instantaneously by electricity.
Samuel F. B. Morse

Mr. Watson, come here, I want to see you.
Alexander Graham Bell

What networks of railroads, highways and canals were in another age, the networks of telecommunications, information and computerization … are today.
Bruno Kreisky

Science may never come up with a better office-communication system than the coffee break.
Earl Wilson

It's currently a problem of access to gigabits through punybaud.
J. C. R. Licklider

SELF-REVIEW EXERCISES

18.1 Fill in the blanks in each of the following statements:

a) Exception _____ occurs when an input/output error occurs when closing a socket.
ANS: `IOException`

b) Exception _____ occurs when a host name indicated by a client cannot be resolved to an address.
ANS: `UnknownHostException`

c) If a `DatagramSocket` constructor fails to set up a `DatagramSocket` properly, an exception of type _____ occurs.
ANS: `SocketException`

d) Many of Java's networking classes are contained in package _____.
ANS: `java.net`

e) Class _____ binds the application to a port for datagram transmission.
ANS: `DatagramSocket`

f) An object of class _____ contains an IP address.
ANS: `InetAddress`

g) The two types of sockets we discussed in this chapter are _____ and _____.
ANS: stream sockets, datagram sockets

h) The acronym URL stands for _____.
ANS: Uniform Resource Locator

i) The acronym URI stands for _____.
ANS: Universal Resource Identifier

j) The key protocol that forms the basis of the World Wide Web is _____.
ANS: HTTP

k) `AppletContext` method _____ receives a `URL` object as an argument and displays in a browser the World Wide Web resource associated with that `URL`.
ANS: `showDocument`

l) Method `getLocalHost` returns a(n) _____ object containing the local host name of the computer on which the program is executing.
ANS: `InetAddress`

m) `MulticastSocket` method _____ subscribes a `MulticastSocket` to a multicast group.
ANS: `joinGroup`

n) The `URL` constructor determines whether its string argument is a valid URL. If so, the `URL` object is initialized with that location; otherwise, a(n) _____ exception occurs.
ANS: `MalformedURLException`

18.2 State whether each of the following is *true or false*. *If* false, explain why.

a) Multicast broadcasts `DatagramPackets` to every host on the Internet.
ANS: False; multicast sends `DatagramPackets` only to hosts that have joined the multicast group.

b) UDP is a connection-oriented protocol.
ANS: False; UDP is a connectionless protocol and TCP is a connection-oriented protocol.

c) With stream sockets a process establishes a connection to another process.
ANS: True

d) A server waits at a port for connections from a client.
ANS: True

e) Datagram packet transmission over a network is reliable—packets are guaranteed to arrive in sequence.
ANS: False; packets could be lost and packets can arrive out of order.

f) For security reasons, many Web browsers such as Netscape allow Java applets to do file processing only on the machines on which they execute.

ANS: False; most browsers prevent applets from doing file processing on the client machine.

g) Web browsers often restrict an applet so that it can only communicate with the machine from which it was originally downloaded.

ANS: True

h) IP addresses from `224.0.0.0` to `239.255.255.255` are reserved for multicast.

ANS: True

EXERCISES

18.3 Distinguish between connection-oriented and connectionless network services.

ANS: Connection-oriented services maintain a connection while data is being transferred. Connectionless services do not maintain a connection. Connection-oriented services are generally slower but more reliable.

18.4 How does a client determine the host name of the client computer?

ANS: `InetAddress.getLocalHost().getHostName()`.

18.5 Under what circumstances would a `SocketException` be thrown?

ANS: If a `DatagramSocket` cannot be constructed properly, a `SocketException` is thrown.

18.6 How can a client get a line of text from a server?

ANS: Through the `Socket` using a stream object (e.g., such as `ObjectInputStream`).

18.7 Describe how a client connects to a server.

ANS: A client can read a file through a `URL` connection by creating a `URL` object and issuing an `openStream` method call on the `URL` object. The `openStream` call returns an `InputStream` object that can be used to read bytes from the file. Also, a `DataInputStream` object can be chained to the `InputStream` returned from `openStream` to allow other data types to be read from the file on the server.

18.8 Describe how a server sends data to a client.

ANS: A client connects to the server by creating a socket using the `Socket` class constructor. The name of the server and the port to connect to are passed to the `Socket` constructor. Information can be exchanged between the client and server using the socket's `InputStream` and `OutputStream`.

18.9 Describe how to prepare a server to receive a stream-based connection request from a single client.

ANS: First a `ServerSocket` object must be created and associated with a port on the server computer. If the `ServerSocket` is created properly, a call to the `accept` method can be issued on the `ServerSocket` object. This call will wait for a client to connect. When a client connects, a `Socket` object is returned from the `accept` call. The `Socket` object is used to get the `InputStream` and `OutputStream` objects for communication with the client.

18.10 Describe how to prepare a server to receive connection requests from multiple clients if each client that connects should be processed concurrently with all other connected clients.

ANS: First a `ServerSocketChannel` object must be created and associated with a port on the server computer, configure the channel so that it works in nonblocking mode. Then register a `Selector` (the interest set contains OP_ACCEPT operation) to the channel. If the `ServerSocketChannel` is created properly and ready to accept new connections, a call to

the `accept` method can be issued on the `ServerSocketChannel`, a `SocketChannel` object is returned from the `accept` call. The `SocketChannel` object is used to communicate with the client.

18.11 How does a server listen for connections at a port?

ANS: The `ServerSocket` `accept` method (or the `ServerSocketChannel` `accept` method) is used.

18.12 What determines how many connect requests from clients can wait in a queue to connect to a server?

ANS: When the `ServerSocket` object is created on the server, the second argument to the `ServerSocket` constructor specifies the "queue length" (the number of clients that can wait to be processed by the server).

18.13 As described in the text, what reasons might cause a server to refuse a connection request from a client?

ANS: A server usually has a capacity of the number of clients that can wait for a connection and be processed by the server. If the queue of clients is full, client connections are refused.

18.14 Use a socket connection to allow a client to specify a file name and have the server send the contents of the file or indicate that the file does not exist.

ANS:

```
1  // Exercise 18.14 Solution: Server.java
2  // Program receives file name from client and sends
3  // contents of file back to the client if file exists.
4  import java.net.*;
5  import java.io.*;
6
7  public class Server {
8     private ServerSocket server;
9     private Socket connection;
10    private BufferedReader input;
11    private BufferedWriter output;
12
13    // constructor
14    public Server()
15    {
16       // create ServerSocket
17       try {
18          server = new ServerSocket( 5000, 10 );
19       }
20
21       // process problems communicating with server
22       catch( IOException exception ) {
23          exception.printStackTrace();
24          System.exit( 0 );
25       }
26    }
27
28    // run server
29    public void runServer()
30    {
31       // wait for connection, get streams, read file
32       try {
```

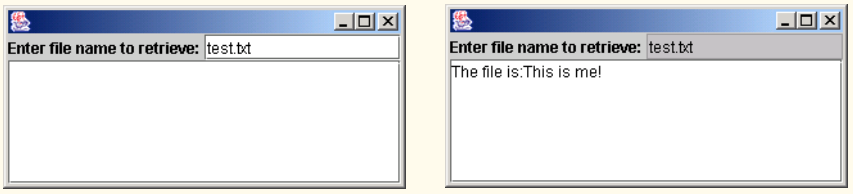
```
33
34          // allow server to accept connection
35          connection = server.accept();
36
37          // set up output stream
38          output = new BufferedWriter( new OutputStreamWriter(
39             connection.getOutputStream() ) );
40
41          // flush output buffer to send header information
42          output.flush();
43
44          // set up input stream
45          input = new BufferedReader( new InputStreamReader(
46             connection.getInputStream() ) );
47
48          // receive file name from client
49          File file = new File( input.readLine() );
50
51          String result;
52
53          // send file to client
54          if ( file.exists() ) {
55             BufferedReader fileInput = new BufferedReader(
56                new InputStreamReader( new FileInputStream( file ) ) );
57
58             // write first 13 characters
59             output.write( "The file is:\n", 0, 13 );
60             output.flush();
61
62             // read first line of file
63             result = fileInput.readLine();
64
65             while ( result != null ) {
66                output.write( result + '\n', 0, result.length() + 1 );
67                output.flush();
68                result = fileInput.readLine();
69             }
70          }
71
72          // file does not exist
73          else {
74             result = file.getName() + " does not exist\n";
75             output.write( result, 0, result.length() );
76             output.flush();
77          }
78
79          // close streams and socket
80          output.close();
81          input.close();
82          connection.close();
83       }
84
85       // process problems communicating with server
86       catch( IOException ioException ) {
```

```
87             System.err.println( "IOException has occurred!" );
88             ioException.printStackTrace();
89
90             System.exit( 0 );
91          }
92
93       }  // end method runServer
94
95       public static void main( String args[] )
96       {
97          Server application = new Server();
98          application.runServer();
99       }
100
101 }  // end class Server
```

```
1  // Exercise 18.14 Solution: Client.java
2  // Program receives a file name from user, sends file name
3  // to server and displays file contents if file exists.
4  import java.awt.*;
5  import java.net.*;
6  import java.io.*;
7  import java.awt.event.*;
8  import javax.swing.*;
9
10 public class Client extends JFrame implements ActionListener {
11    private JTextField fileField;
12    private JTextArea contents;
13    private BufferedReader bufferInput;
14    private BufferedWriter bufferOutput;
15    private Socket connection;
16    private JPanel panel;
17    private JLabel label;
18    private JScrollPane scroller;
19
20    // set up GUI, connect to server, get streams
21    public Client()
22    {
23       // set up GUI
24       label = new JLabel( "Enter file name to retrieve:" );
25
26       panel = new JPanel();
27       panel.setLayout( new GridLayout( 1, 2, 0, 0 ) );
28       panel.add( label );
29
30       fileField = new JTextField();
31       fileField.addActionListener( this );
32       panel.add( fileField );
33
34       contents = new JTextArea();
35       scroller = new JScrollPane( contents );
36
37       Container container = getContentPane();
```

```
38          container.setLayout( new BorderLayout() );
39          container.add( panel, BorderLayout.NORTH );
40          container.add( scroller, BorderLayout.CENTER );
41
42          // connect to server, get streams
43          try {
44
45             // create Socket to make connection to server
46             connection = new Socket( InetAddress.getLocalHost(), 5000 );
47
48             // set up output stream
49             bufferOutput = new BufferedWriter( new OutputStreamWriter(
50                connection.getOutputStream() ) );
51
52             // flush output buffer to send header information
53             bufferOutput.flush();
54
55             // set up input stream
56             bufferInput = new BufferedReader( new InputStreamReader(
57                connection.getInputStream() ) );
58          }
59
60          // process problems communicating with server
61          catch( IOException ioException ) {
62             ioException.printStackTrace();
63          }
64
65          setSize( 500, 500 );
66          setVisible( true );
67       }
68
69       // process file name entered by user
70       public void actionPerformed( ActionEvent event )
71       {
72          // display contents of file
73          try {
74             String fileName = event.getActionCommand() + "\n";
75             bufferOutput.write( fileName, 0, fileName.length() );
76             bufferOutput.flush();
77             String output = bufferInput.readLine();
78
79             contents.setText( output );
80
81             // if file exists, dislay file contents
82             if ( output.equals( "The file is:" ) ) {
83                output = bufferInput.readLine();
84
85                while ( output != null ) {
86                   contents.append( output + "\n" );
87                   output = bufferInput.readLine();
88                }
89             }
90
91             fileField.setEditable( false );
```

```
92             fileField.setBackground( Color.lightGray );
93             fileField.removeActionListener( this );
94
95             // close streams and socket
96             bufferOutput.close();
97             bufferInput.close();
98             connection.close();
99          }
100
101         // end of file
102         catch ( EOFException eofException ) {
103            System.out.println( "End of file" );
104         }
105
106         // process problems communicating with server
107         catch ( IOException ioException ) {
108            ioException.printStackTrace();
109         }
110      }
111
112      public static void main( String args[] )
113      {
114         Client application = new Client();
115         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
116      }
117
118   }  // end class Client
```



18.15 Modify Exercise 18.14 to allow the client to modify the contents of the file and send the file back to the server for storage. The user can edit the file in a `JTextArea`, then click a *save changes* button to send the file back to the server.

ANS:

```
1  // Exercise 18.15 Solution: Server.java
2  // Program receives file name from client, allows the user to modify the
3  // file and sends contents of file back to the client if file exists.
4  import java.net.*;
5  import java.io.*;
6
7  public class Server {
8     private ServerSocket server;
9     private Socket connection;
10    private BufferedReader input;
11    private BufferedWriter output;
```

```
12    private File file;
13
14    // constructor
15    public Server()
16    {
17       // create ServerSocket
18       try {
19          server = new ServerSocket( 5000, 10 );
20       }
21
22       // process problems communicating with server
23       catch( IOException exception ) {
24          exception.printStackTrace();
25          System.exit( 0 );
26       }
27    }
28
29    // run server
30    public void runServer()
31    {
32       // wait for connection, get streams, read file
33       try {
34
35          // allow server to accept connection
36          connection = server.accept();
37
38          // set up output stream
39          output = new BufferedWriter( new OutputStreamWriter(
40             connection.getOutputStream() ) );
41
42          // flush output buffer to send header information
43          output.flush();
44
45          // set up input stream
46          input = new BufferedReader( new InputStreamReader(
47             connection.getInputStream() ) );
48
49          // receive file name from client
50          file = new File( input.readLine() );
51
52          String result;
53
54          // send file to client
55          if ( file.exists() ) {
56             BufferedReader fileInput = new BufferedReader(
57                new InputStreamReader( new FileInputStream( file ) ) );
58
59             // read file
60             result = fileInput.readLine();
61
62             while ( result != null ) {
63                output.write( result + '\n', 0, result.length() + 1 );
64                output.flush();
65                result = fileInput.readLine();
66             }
```

```
67            }
68
69            // file does not exist
70            else {
71               result = file.getName() + " does not exist\n";
72               output.write( result, 0, result.length() );
73               output.flush();
74            }
75
76            // close streams and socket
77            output.close();
78            input.close();
79            connection.close();
80
81            // allow server to accept connection
82            connection = server.accept();
83
84            // receive changed file from client set up input stream
85            input = new BufferedReader( new InputStreamReader(
86               connection.getInputStream() ) );
87
88            BufferedWriter writer = new BufferedWriter(
89               new FileWriter( file ) );
90
91            // write file
92            result = input.readLine();
93
94            while ( result != null ) {
95               writer.write( result );
96               writer.flush();
97               writer.newLine();
98
99               result = input.readLine();
100           }
101
102           input.close();
103           connection.close();
104        }
105
106        // process problems communicating with server
107        catch( IOException ioException ) {
108           System.err.println( "IOException has occurred!" );
109           ioException.printStackTrace();
110
111           System.exit( 0 );
112        }
113
114     }  // end method runServer
115
116     public static void main( String args[] )
117     {
118        Server application = new Server();
119        application.runServer();
120     }
```

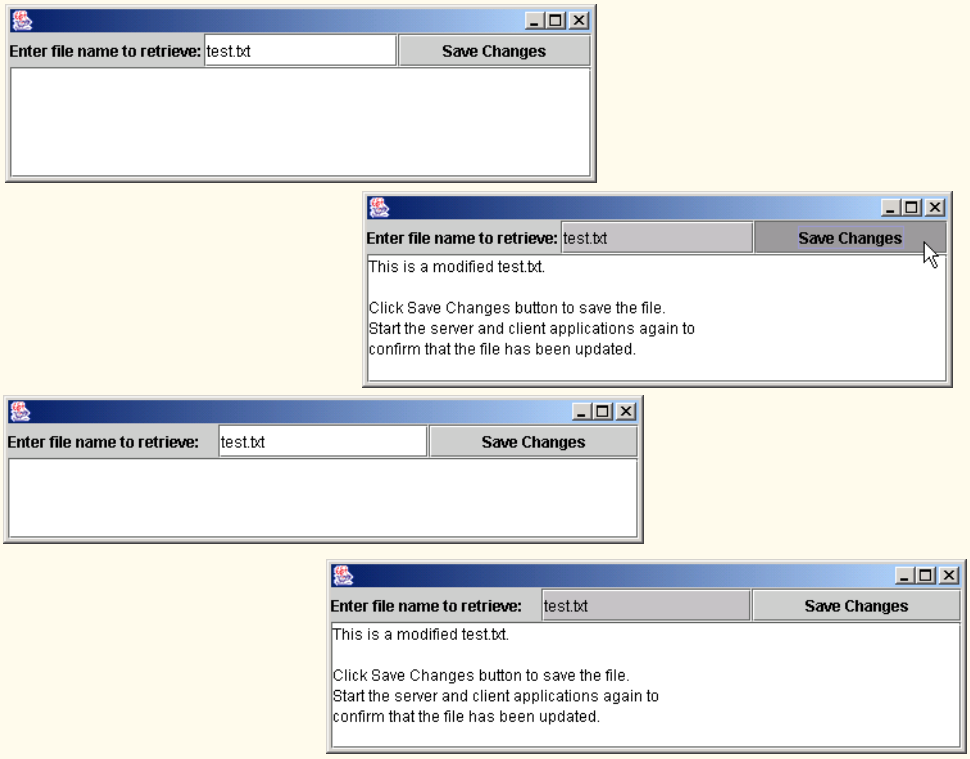
```

} // end class Server
```

```
// Exercise 18.15 Solution: Client.java
// Program receives a file name from user, sends file name
// to server and displays file contents if file exists. It
// also allows the user to modify the files.
import java.awt.*;
import java.net.*;
import java.io.*;
import java.awt.event.*;
import javax.swing.*;

public class Client extends JFrame implements ActionListener {
   private JTextField fileField;
   private JTextArea contents;
   private BufferedReader bufferInput;
   private BufferedWriter bufferOutput;
   private Socket connection;
   private JPanel panel;
   private JLabel label;
   private JScrollPane scroller;
   private JButton saveButton;

   // set up GUI, connect to server, get streams
   public Client()
   {
      // set up GUI
      label = new JLabel( "Enter file name to retrieve:" );
      fileField = new JTextField();
      fileField.addActionListener( this );
      saveButton = new JButton( "Save Changes" );

      // allow user to send changes back to server
      saveButton.addActionListener(

         new ActionListener() {

            public void actionPerformed( ActionEvent event ) {
               saveFile();
            }
         }
      );

      contents = new JTextArea();
      scroller = new JScrollPane( contents );

      // add components to container
      panel = new JPanel();
      panel.setLayout( new GridLayout( 1, 3, 0, 0 ) );
      panel.add( label );
      panel.add( fileField );
      panel.add( saveButton );
```

```
51
52          Container container = getContentPane();
53          container.setLayout( new BorderLayout() );
54          container.add( panel, BorderLayout.NORTH );
55          container.add( scroller, BorderLayout.CENTER );
56
57          // connect to server, get streams
58          try {
59
60             // create Socket to make connection to server
61             connection = new Socket( InetAddress.getLocalHost(), 5000 );
62
63             // set up output stream
64             bufferOutput = new BufferedWriter(
65                new OutputStreamWriter( connection.getOutputStream() ) );
66
67             // flush output buffer to send header information
68             bufferOutput.flush();
69
70             // set up input stream
71             bufferInput = new BufferedReader(
72                new InputStreamReader( connection.getInputStream() ) );
73          }
74
75          // process problems communicating with server
76          catch( IOException ioException ) {
77             ioException.printStackTrace();
78          }
79
80          setSize( 500, 500 );
81          setVisible( true );
82
83       } // end constructor Client
84
85       // send the new version of the file to the server
86       public void saveFile() {
87
88          // connect to server, get streams
89          try {
90
91             // create Socket to make connection to server
92             connection = new Socket( InetAddress.getLocalHost(), 5000 );
93
94             // set up output stream
95             bufferOutput = new BufferedWriter(
96                new OutputStreamWriter( connection.getOutputStream() ) );
97
98             String file = contents.getText();
99
100            // add text of file to output stream
101            bufferOutput.write( file, 0, file.length() );
102            bufferOutput.flush();
103
104            // close streams and socket
105            bufferOutput.close();
```

```
106          bufferInput.close();
107          connection.close();
108       }
109
110       // end of file
111       catch ( EOFException eofException ) {
112          System.out.println( "End of file" );
113       }
114
115       // process problems communicating with server
116       catch ( IOException ioException ) {
117          ioException.printStackTrace();
118       }
119
120    } // end method saveFile
121
122    // process file name entered by user
123    public void actionPerformed( ActionEvent event )
124    {
125       // display contents of file
126       try {
127          String fileName = event.getActionCommand() + "\n";
128          bufferOutput.write( fileName, 0, fileName.length() );
129          bufferOutput.flush();
130          String output = bufferInput.readLine();
131
132          while ( output != null ) {
133             contents.append( output + "\n" );
134             output = bufferInput.readLine();
135          }
136
137          fileField.setEditable( true );
138          fileField.setBackground( Color.lightGray );
139
140          // close streams and socket
141          bufferOutput.close();
142          bufferInput.close();
143          connection.close();
144       }
145
146       // end of file
147       catch ( EOFException eofException ) {
148          System.out.println( "End of file" );
149       }
150
151       // process problems communicating with server
152       catch ( IOException ioException ) {
153          ioException.printStackTrace();
154       }
155    }
156
157    public static void main( String args[] )
158    {
159       Client application = new Client();
```

```
160         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
161      }
162
163   }  // end class Client
```



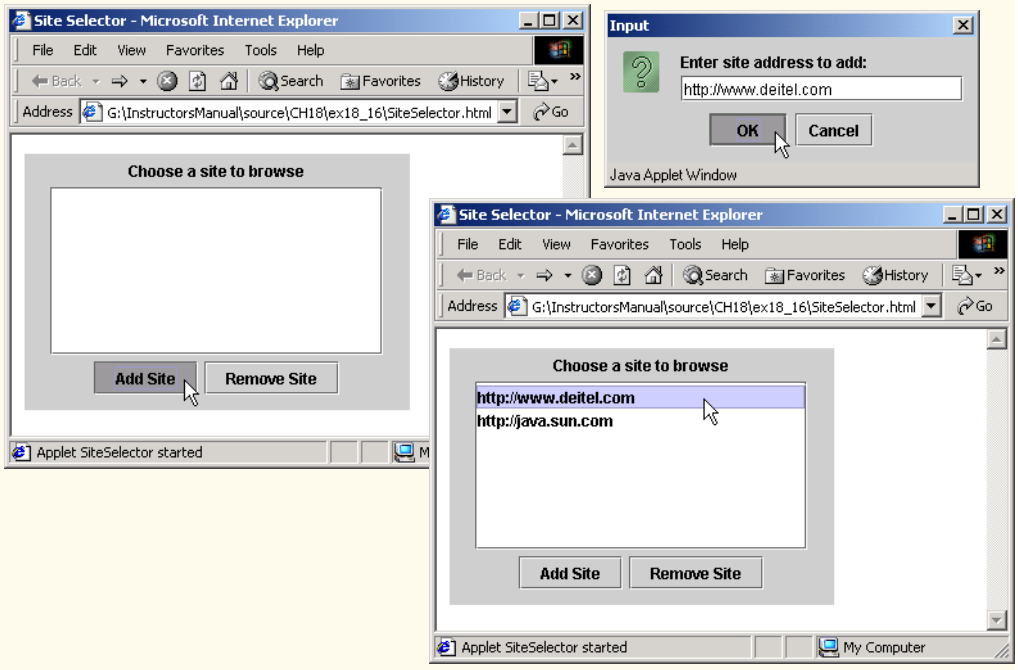
18.16 Modify program of Fig. 18.2 to allow users to add their own sites to the list and remove sites from the list.

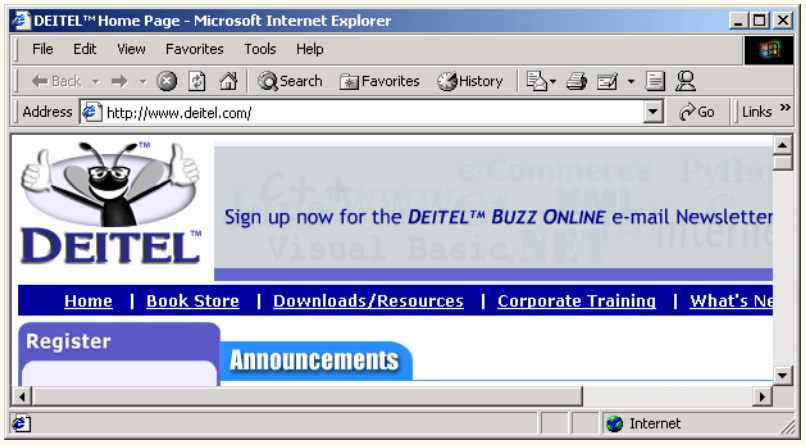
ANS:

```
1   // Exercise 18.16 solution: SiteSelector.java
2   // Program that allows user to add/remove/view web sites.
3   import java.net.*;
4   import java.util.*;
5   import java.awt.*;
6   import java.awt.event.*;
7   import java.applet.AppletContext;
8   import javax.swing.*;
9   import javax.swing.event.*;
10
11  public class SiteSelector extends JApplet {
12     private Container container;  // applet container
13     private static Vector siteNames;   // site names
14     private JList siteChooser;  // list of sites to choose from
```

```
15    private JButton addButton, removeButton;  // add/remove buttons
16
17    // read HTML parameters and set up GUI
18    public void init()
19    {
20       // create Hashtable and Vector
21       siteNames = new Vector();
22
23       // create GUI components and layout interface
24       container = getContentPane();
25       container.setLayout( new FlowLayout() );
26       container.add( new JLabel( "Choose a site to browse" ),
27          BorderLayout.NORTH );
28
29       siteChooser = new JList( siteNames );
30       siteChooser.addListSelectionListener(
31
32          new ListSelectionListener() {
33
34             public void valueChanged( ListSelectionEvent event )
35             {
36                // go to site user selected
37                try {
38
39                   // get selected site address
40                   Object object = siteChooser.getSelectedValue();
41                   URL newDocument = new URL( object.toString() );
42
43                   // get reference to applet container
44                   AppletContext browser = getAppletContext();
45
46                   // tell applet container to change pages
47                   browser.showDocument( newDocument, "_blank" );
48                }
49
50                catch( Exception exception ) {
51                   exception.printStackTrace();
52                }
53
54             } // end method valueChanged
55
56          } // end anonymous inner class
57
58       ); // end call to addListSelectionListener
59
60       container.add( new JScrollPane( siteChooser ) );
61
62       addButton = new JButton( "Add Site" );
63       addButton.addActionListener( new ButtonHandler() );
64       container.add( addButton );
65
66       removeButton = new JButton( "Remove Site" );
67       removeButton.addActionListener( new ButtonHandler() );
68       container.add( removeButton );
```

```
69
70     } // end method init
71
72     private class ButtonHandler implements ActionListener {
73
74        public void actionPerformed( ActionEvent event )
75        {
76           String siteAddress;
77
78           if ( event.getSource() == addButton ) {
79              siteAddress = JOptionPane.showInputDialog(
80                 "Enter site address to add:" );
81              siteNames.add( siteAddress );
82              siteChooser.setListData( siteNames );
83           }
84           else {
85              siteAddress = JOptionPane.showInputDialog(
86                 "Enter site address to remove:" );
87              siteNames.remove( siteAddress );
88              siteChooser.setListData( siteNames );
89           }
90
91           container.repaint();
92        }
93
94     } // end inner class ButtonHandler
95
96  } // end class SiteSelector
```





18.17 Multithreaded servers are quite popular today, especially because of the increasing use of multiprocessing servers. Modify the simple server application presented in Section 18.6 to be a multithreaded server. Then use several client applications and have each of them connect to the server simultaneously. Use a `Vector` to store the client threads. `Vector` provides several methods of use in this exercise. Method `size` determines the number of elements in a `Vector`. Method `get` returns the element (as an `Object` reference) in the location specified by its argument. Method `add` places its argument at the end of the `Vector`. Method `remove` deletes its argument from the `Vector`. Method `lastElement` returns an `Object` reference to the last object you inserted in the `Vector`.

ANS:

```
1  // Exercise 18.17 Solution: Server2.java
2  // Program sets up a Server that receives connections from clients, sends
3  // strings to the clients and receives string from the clients.
4  import java.io.*;
5  import java.net.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.util.*;
9  import javax.swing.*;
10
11 public class Server2 extends JFrame {
12    private JTextField enterField;
13    private JTextArea display;
14    private JScrollPane scroller;
15    private Vector clients;
16    private int numberOfClients;
17
18    // set up GUI
19    public Server2()
20    {
21       super( "Server" );
22
23       numberOfClients = 0;
24
```

```
25          // create enterField and register listener
26          enterField = new JTextField();
27          enterField.setEnabled( false );
28          enterField.addActionListener(
29
30             new ActionListener() {  // anonymous inner class
31
32                // send message to clients
33                public void actionPerformed( ActionEvent event )
34                {
35                   for ( int i = 0; i < clients.size(); i++ )
36                      ( ( ClientThread ) clients.elementAt( i ) ).sendData(
37                         event.getActionCommand() );
38
39                   enterField.setText( "" );
40                }
41             }
42          );
43
44          display = new JTextArea();
45          display.setEnabled( false );
46          scroller = new JScrollPane( display );
47
48          Container container = getContentPane();
49          container.add( enterField, BorderLayout.NORTH );
50          container.add( scroller, BorderLayout.CENTER );
51
52          setSize( 300, 150 );
53          setVisible( true );
54
55       } // end constructor Server2
56
57       // set up and run server
58       public void runServer()
59       {
60          // set up server and process connections
61          try {
62
63             // create ServerSocket
64             ServerSocket server = new ServerSocket( 5558, 100 );
65
66             clients = new Vector();
67
68             // accept connections and add ClientThreads to Vector
69             while ( true ) {
70                display.append( "Waiting for connection\n" );
71                numberOfClients++;
72                clients.add( new ClientThread( server.accept(),
73                   display, numberOfClients ) );
74                ( ( ClientThread ) clients.lastElement() ).start();
75
76                enterField.setEnabled( true );
77             }
78          }
```

```
79
80          // process problems with I/O
81          catch ( IOException ioException ) {
82             ioException.printStackTrace();
83          }
84
85       } // end method runServer
86
87       public static void main( String args[] )
88       {
89          Server2 application = new Server2();
90          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
91          application.runServer();
92       }
93
94       // private inner class ClientThread manages each Client as a thread
95       private class ClientThread extends Thread {
96          private int clientNumber;
97          private Socket connection;
98          private ObjectOutputStream output;
99          private ObjectInputStream input;
100         private JTextArea display;
101
102         // set up a Client thread
103         public ClientThread( Socket socket, JTextArea display, int number )
104         {
105            this.display = display;
106            clientNumber = number;
107            connection = socket;
108
109            // obtain streams from Socket
110            try {
111               output = new ObjectOutputStream(
112                  connection.getOutputStream() );
113               output.flush();
114
115               input = new ObjectInputStream( connection.getInputStream() );
116
117               sendData( "Connection successful" );
118
119               this.display.append( "\nConnection " +
120                  clientNumber + " received from: " +
121                  connection.getInetAddress().getHostName() + "\n" );
122            }
123
124            // process problems with IO
125            catch ( IOException ioException ) {
126               ioException.printStackTrace();
127            }
128
129         } // end constructor ClientThread
130
131         // send message to client
132         public void sendData( String message )
133         {
```

```
134          // send object to client
135          try {
136             output.writeObject( "SERVER>>> " + message );
137             output.flush();
138             display.append( "\nSERVER>>>" + message );
139          }
140
141          // process problems sending object
142          catch ( IOException ioException ) {
143             display.append( "\nError writing object" );
144          }
145       }
146
147       // control thread's execution
148       public void run()
149       {
150          String message = null;
151
152          // process connection
153          try {
154
155             // read message from client
156             do {
157
158                try {
159                   message = ( String ) input.readObject();
160                   display.append( "\n" + message );
161                   display.setCaretPosition( display.getText().length() );
162                }
163
164                // process problems reading from client
165                catch ( ClassNotFoundException classNotFoundException ) {
166                   display.append( "\nUnknown object type received" );
167                }
168
169             } while ( !message.equals( "CLIENT>>> TERMINATE" ) );
170
171             display.append( "\nClient terminated connection" );
172             display = null;
173          }
174
175          // process problems with I/O
176          catch ( IOException ioException ) {
177             System.out.println( "Client terminated connection" );
178          }
179
180          // close streams and socket
181          finally {
182
183             try {
184                output.close();
185                input.close();
186                connection.close();
187             }
```

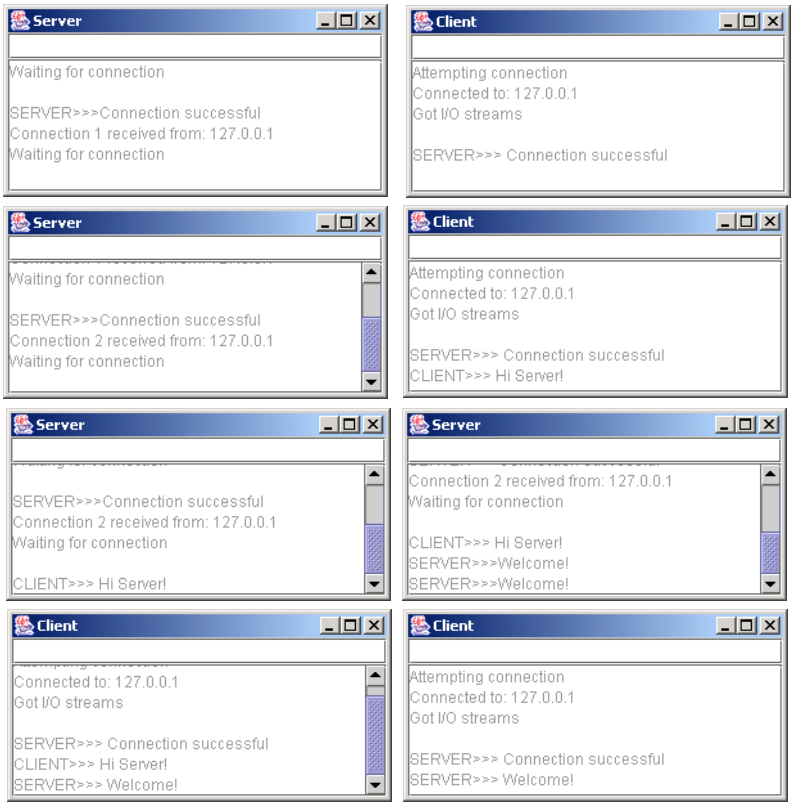
```
188
189             // process problems with I/O
190             catch ( IOException ioException ) {
191                ioException.printStackTrace();
192             }
193
194             clients.remove( this );
195          }
196
197       }  // end method run
198
199    }  // end class ClientThread
200
201 }  // end class Server2
```

```
1  // Exercise 18.17 Solution: Client2.java
2  // Program sets up a Client that will read information
3  // sent from a Server and display the information.
4  import java.io.*;
5  import java.net.*;
6  import java.awt.*;
7  import java.awt.event.*;
8  import java.util.*;
9  import javax.swing.*;
10
11 public class Client2 extends JFrame {
12    private JTextField enterField;
13    private JTextArea displayArea;
14    private JScrollPane scroller;
15    ObjectOutputStream output;
16    ObjectInputStream input;
17    String message = "";
18
19    // set up GUI
20    public Client2()
21    {
22       super( "Client" );
23
24       addWindowListener(  // set closing operation
25
26          new WindowAdapter() {
27
28             public void windowClosing( WindowEvent e ) {
29
30                sendData( "TERMINATE" );
31                System.exit( 0 );
32             }
33          }
34       );
35
36       // create enterField and register listener
37       enterField = new JTextField();
38       enterField.setEnabled( false );
```

```
39          enterField.addActionListener(
40
41             new ActionListener() {  // anonymous inner class
42
43                // send message to server
44                public void actionPerformed( ActionEvent event )
45                {
46                   sendData( event.getActionCommand() );
47                   enterField.setText( "" );
48                }
49
50             }  // end anonymous inner class
51
52          ); // end call to addActionListener
53
54          displayArea = new JTextArea();
55          displayArea.setEnabled( false );
56          scroller = new JScrollPane( displayArea );
57
58          Container container = getContentPane();
59          container.add( enterField, BorderLayout.NORTH );
60          container.add( scroller, BorderLayout.CENTER );
61
62          setSize( 300, 150 );
63          setVisible( true );
64
65       } // end constructor Client2
66
67       // connect to server, get streams, process connection
68       public void runClient()
69       {
70          Socket client;
71
72          // connect to server, get streams, process connection
73          try {
74             displayArea.setText( "Attempting connection\n" );
75
76             // create Socket to make connection to server
77            client = new Socket( InetAddress.getByName( "127.0.0.1" ), 5558 );
78
79             // display connection information
80             displayArea.append( "Connected to: " +
81                client.getInetAddress().getHostName() );
82
83             // set up output stream for objects
84             output = new ObjectOutputStream( client.getOutputStream() );
85
86             // flush output buffer to send header information
87             output.flush();
88
89             // set up input stream for objects
90             input = new ObjectInputStream( client.getInputStream() );
91
92             displayArea.append( "\nGot I/O streams\n" );
```

```
93
94          // enable enterField so client user can send messages
95          enterField.setEnabled( true );
96
97          // process messages sent from server
98          do {
99
100            // read message and display it
101            try {
102               message = ( String ) input.readObject();
103               displayArea.append( "\n" + message );
104               displayArea.setCaretPosition(
105                  displayArea.getText().length() );
106            }
107
108            // catch problems reading from server
109            catch ( ClassNotFoundException classNotFoundException ) {
110               displayArea.append( "\nUnknown object type received" );
111            }
112
113         } while ( !message.equals( "SERVER>>> TERMINATE" ) );
114
115         displayArea.append( "\nClosing connection.\n" );
116
117         // close streams and socket
118         output.close();
119         input.close();
120         client.close();
121
122         displayArea.append( "Connection closed." );
123
124      }  // end try
125
126      // server closed connection
127      catch ( EOFException eofException ) {
128         System.err.println( "Server terminated connection" );
129      }
130
131      // process problems communicating with server
132      catch ( IOException ioException ) {
133         ioException.printStackTrace();
134      }
135
136   } // end method runClient
137
138   // send message to server
139   private void sendData( String string )
140   {
141      // send object to client
142      try {
143         message = string;
144         output.writeObject( "CLIENT>>> " + string );
145         output.flush();
146         displayArea.append( "\nCLIENT>>> " + string );
147      }
```

```
148
149         // process problems sending object
150         catch ( IOException ioException ) {
151            displayArea.append( "\nError writing object" );
152            ioException.printStackTrace();
153         }
154      }
155
156      public static void main( String args[] )
157      {
158         final Client2 application = new Client2();
159         application.runClient();
160      }
161
162   }  // end class Client2
```



18.18 In the text, we presented a tic-tac-toe program controlled by a multithreaded server. Develop a checkers program modeled after the tic-tac-toe program. The two users should alternate making moves. Your program should mediate the players' moves, determining whose turn it is and allowing only valid moves. The players themselves will determine when the game is over.

ANS:

```
// Exercise 18.18 solution: CheckerGameServer.java
// This class maintains check game for two client applets.
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import javax.swing.*;

public class CheckerGameServer extends JFrame {
   private byte board[], kingBoard[];
   private JTextArea outputArea;
   private Player players[];
   private ServerSocket server;
   private int currentPlayer;
   private int currentCell = -1, takeOver1 = -1, takeOver2 = -1;

   // set up check game server and GUI that displays messages
   public CheckerGameServer()
   {
      super( "Check Game Server" );

      board = new byte[ 64 ];
      kingBoard = new byte[ 64 ];

      // initialize board
      for ( int row = 0; row < 8; row++ ) {
         for ( int column = 0; column < 8; column++ ) {
            if ( row < 3 && row % 2 == column % 2 )
               board[ row * 8 + column ] = ( byte ) 'W';
            else if ( row > 4 && row % 2 == column % 2 )
               board[ row * 8 + column ] = ( byte ) 'Y';
         }
      }

      players = new Player[ 2 ];
      currentPlayer = 0;

      // set up ServerSocket
      try {
         server = new ServerSocket( 12345, 2 );
      }

      // process problems creating ServerSocket
      catch( IOException ioException ) {
         ioException.printStackTrace();
         System.exit( 1 );
      }

      // set up JTextArea to display messages during execution
      outputArea = new JTextArea();
      getContentPane().add(
         new JScrollPane( outputArea ), BorderLayout.CENTER );
```

```
53          outputArea.setText( "Server awaiting connections\n" );
54
55          setSize( 300, 300 );
56          setVisible( true );
57
58       } // end constructor TicTacToeServer
59
60       // wait for two connections so game can be played
61       public void execute()
62       {
63          // wait for each client to connect
64          for ( int i = 0; i < players.length; i++ ) {
65
66             // wait for connection, create Player, start thread
67             try {
68                players[ i ] = new Player( server.accept(), i );
69                players[ i ].start();
70             }
71
72             // process problems receiving connection from client
73             catch( IOException ioException ) {
74                ioException.printStackTrace();
75                System.exit( 1 );
76             }
77          }
78
79          // Player W is suspended until Player Y connects.
80          // Resume player W now.
81          synchronized ( players[ 0 ] ) {
82             players[ 0 ].setSuspended( false );
83             players[ 0 ].notify();
84          }
85
86       }  // end method execute
87
88       // display a message in outputArea
89       public void display( String message )
90       {
91          outputArea.append( message + "\n" );
92       }
93
94       // Determine if a move is valid. This method is synchronized because
95       // only one move can be made at a time.
96       public synchronized boolean validMove(
97          int oldCell, int newCell, int player )
98       {
99          boolean moveDone = false;
100
101         // while not current player, must wait for turn
102         while ( player != currentPlayer ) {
103
104            // wait for turn
105            try {
106               wait();
107            }
```

```
108
109          // catch wait interruptions
110          catch( InterruptedException interruptedException ) {
111             interruptedException.printStackTrace();
112          }
113       }
114
115       // if a diagonal move and location not occupied, make move
116       if ( !isOccupied( newCell ) && isDiagonal( oldCell, newCell ) &&
117          isForwardMove( oldCell, newCell, player ) ) {
118
119          if ( currentPlayer == 0 && board[ oldCell ] != 'W' )
120             return false;
121          if ( currentPlayer == 1 && board[ oldCell ] != 'Y' )
122             return false;
123
124          // set move in board array
125          board[ newCell ] = ( byte ) ( currentPlayer == 0 ? 'W' : 'Y' );
126          board[ oldCell ] = ' ';
127          currentCell = newCell;
128
129          int take1 = -1, take2 = -1;
130          int newRow = newCell / 8;
131          int oldRow = oldCell / 8;
132          int takeRow = newRow > oldRow ? newRow - 1 : oldRow - 1;
133          int newColumn = newCell % 8;
134          int oldColumn = oldCell % 8;
135          int takeColumn = newColumn > oldColumn ?
136              newColumn - 1 : oldColumn - 1;
137
138          // if take occurs and next take available, continue takes
139          if ( currentPlayer == 0 ) {
140             int takeCell = takeRow * 8 + takeColumn;
141
142             if ( board[ takeCell ] == 'Y' ) {
143                board[ takeCell ] = ' ';
144                take1 = takeCell;
145                take2 = nextTake( newCell, currentPlayer );
146                takeOver1 = take1;
147                takeOver2 = take2;
148             }
149          }
150          else {  // currentPlayer == 1
151             int takeCell = takeRow * 8 + takeColumn;
152
153             if ( board[ takeCell ] == 'W' ) {
154                board[ takeCell ] = ' ';
155                take1 = takeCell;
156                take2 = nextTake( newCell, currentPlayer );
157                takeOver1 = take1;
158                takeOver2 = take2;
159             }
160          }
161
```

```
162          // check whether become king
163          if ( currentPlayer == 0 ) {
164             if ( currentCell / 8 == 7 )
165                kingBoard[ currentCell ] = ( byte ) 'K';
166          }
167          else {
168             if ( currentCell / 8 == 0 )
169                kingBoard[ currentCell ] = ( byte ) 'K';
170          }
171
172          // update king board
173          if ( kingBoard[ oldCell ] == 'K' )
174             kingBoard[ currentCell ] = 'K';
175
176          // change current player
177          currentPlayer = ( currentPlayer + 1 ) % 2;
178
179          // let new current player know that move occurred
180          players[ currentPlayer ].otherPlayerMoved(
181             oldCell, currentCell, take1, take2 );
182
183          // tell waiting player to continue
184          notify();
185
186          // tell player that made move that the move was valid
187          return true;
188       }
189
190       // tell player that made move that the move was not valid
191       else
192          return false;
193
194    } // end method validMove
195
196    // determine whether location is occupied
197    public boolean isOccupied( int location )
198    {
199       if ( board[ location ] == 'W' || board [ location ] == 'Y' )
200           return true;
201       else
202           return false;
203    }
204
205    // determine diagonal move
206    public boolean isDiagonal( int oldLocation, int newLocation )
207    {
208       int oldRow = oldLocation / 8;
209       int oldColumn = oldLocation % 8;
210
211       // check diagonal up to two levels
212       for ( int i = 1; i < 3; i++ ) {
213          if ( newLocation == ( oldRow - i ) * 8 + oldColumn - i )
214             return true;
215          else if ( newLocation == ( oldRow - i ) * 8 + oldColumn + i )
216             return true;
```

```
217          else if ( newLocation == ( oldRow + i ) * 8 + oldColumn + i )
218             return true;
219          else if ( newLocation == ( oldRow + i ) * 8 + oldColumn - i )
220             return true;
221       }
222
223       return false;
224
225    } // end method isDiagonal
226
227    // determine forward move
228    public boolean isForwardMove( int oldCell, int newCell, int player )
229    {
230       if ( kingBoard[ oldCell ] == 'K' )  // king can move backword
231          return true;
232       else {  // non-king can only move forward
233          if ( player == 0 )
234             return newCell > oldCell ? true : false;
235          else
236             return newCell < oldCell ? true : false;
237       }
238    }
239
240    // next take if available
241    public int nextTake( int newCell, int player )
242    {
243       int row = newCell / 8;
244       int column = newCell % 8;
245       int take2 = -1;
246
247       switch ( player ) {
248
249          case 0:  // player White
250
251             // check left take
252             if ( row + 2 <= 7 && column - 2 >= 0) {
253
254                if ( ( board[ ( row + 1 ) * 8 + column - 1 ] == 'Y' ) &&
255                   ( board[ ( row + 2 ) * 8 + column - 2 ] == ' ' ) ) {
256                   board[ newCell ] = ' ';
257                   board[ ( row + 1 ) * 8 + column - 1 ] = ' ';
258                   board[ ( row + 2 ) * 8 + column - 2 ] = 'W';
259                   take2 = ( row + 1 ) * 8 + column - 1;
260                   currentCell = ( row + 2 ) * 8 + column - 2;
261                }
262             }
263
264             // check right take
265             else if ( row + 2 <= 7 && column + 2 <= 7 ) {
266
267                if ( ( board[ ( row + 1 ) * 8 + column + 1 ] == 'Y' ) &&
268                   ( board[ ( row + 2 ) * 8 + column + 2 ] == ' ' ) ) {
269                   board[ newCell ] = ' ';
270                   board[ ( row + 1 ) * 8 + column + 1 ] = ' ';
```

```
271                   board[ ( row + 2 ) * 8 + column + 2 ] = 'W';
272                   take2 = ( row + 1 ) * 8 + column + 1;
273                   currentCell = ( row + 2 ) * 8 + column + 2;
274                }
275             }
276             break;
277
278          case 1:  // player Yellow
279
280             // check left take
281             if ( row - 2 >= 0 && column - 2 >= 0 ) {
282
283                if ( ( board[ ( row - 1 ) * 8 + column - 1 ] == 'W' ) &&
284                   ( board[ ( row - 2 ) * 8 + column - 2 ] == ' ' ) ) {
285                   board[ newCell ] = ' ';
286                   board[ ( row - 1 ) * 8 + column - 1 ] = ' ';
287                   board[ ( row - 2 ) * 8 + column - 2 ] = 'Y';
288                   take2 = ( row - 1 ) * 8 + column - 1;
289                   currentCell = ( row - 2 ) * 8 + column - 2;
290                }
291             }
292
293             // check right take
294             if ( row - 2 >= 0 && column + 2 <= 7 ) {
295
296                if ( ( board[ ( row - 1 ) * 8 + column + 1 ] == 'W' ) &&
297                   ( board[ ( row - 2 ) * 8 + column + 2 ] == ' ' ) ) {
298                   board[ newCell ] = ' ';
299                   board[ ( row - 1 ) * 8 + column + 1 ] = ' ';
300                   board[ ( row - 2 ) * 8 + column + 2 ] = 'Y';
301                   take2 = ( row - 1 ) * 8 + column + 1;
302                   currentCell = ( row - 2 ) * 8 + column + 2;
303                }
304             }
305             break;
306
307       } // end switch
308
309       return take2;
310
311    } // end method nextTake
312
313    // place code in this method to determine whether game over
314    public boolean gameOver()
315    {
316       return false;  // leave this as an exercise
317    }
318
319    public static void main( String args[] )
320    {
321       CheckerGameServer application = new CheckerGameServer();
322       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
323       application.execute();
324    }
```

```
325
326    // private inner class Player manages each Player as a thread
327    private class Player extends Thread {
328       private Socket connection;
329       private DataInputStream input;
330       private DataOutputStream output;
331       private int playerNumber;
332       private char mark;
333       protected boolean suspended = true;
334
335       // set up Player thread
336       public Player( Socket socket, int number )
337       {
338          playerNumber = number;
339
340          // specify player's mark
341          mark = ( playerNumber == 0 ? 'W' : 'Y' );
342
343          connection = socket;
344
345          // obtain streams from Socket
346          try {
347             input = new DataInputStream( connection.getInputStream() );
348             output = new DataOutputStream( connection.getOutputStream() );
349          }
350
351          // process problems getting streams
352          catch( IOException ioException ) {
353             ioException.printStackTrace();
354             System.exit( 1 );
355          }
356
357       } // end constructor Player
358
359       // send message that other player moved
360       public void otherPlayerMoved(
361          int oldCell, int newCell, int take1, int take2 )
362       {
363          // send message indicating move
364          try {
365             output.writeUTF( "Opponent moved" );
366             output.writeUTF(
367                oldCell + " " + newCell + " " + take1 + " " + take2 );
368          }
369
370          // process problems sending message
371          catch ( IOException ioException ) {
372             ioException.printStackTrace();
373          }
374       }
375
376       // control thread's execution
377       public void run()
378       {
```

```
379          // send client message indicating its mark (W or Y),
380          // process messages from client
381          try {
382             display( "Player " + ( playerNumber == 0 ? 'W' : 'Y' ) +
383                " connected" );
384
385             // send player's mark
386             output.writeChar( mark );
387
388             // send message indicating connection
389             output.writeUTF( "Player " + ( playerNumber == 0 ?
390                "W connected\n" : "Y connected, please wait\n" ) );
391
392             // if player W, wait for another player to arrive
393             if ( mark == 'W' ) {
394                output.writeUTF( "Waiting for another player" );
395
396                // wait for player Y
397                try {
398                   synchronized( this ) {
399                      while ( suspended )
400                         wait();
401                   }
402                }
403
404                // process interruptions while waiting
405                catch ( InterruptedException exception ) {
406                   exception.printStackTrace();
407                }
408
409                // send message that other player connected and
410                // player W can make a move
411                output.writeUTF( "Other player connected. Your move." );
412             }
413
414             // while game not over
415             while ( ! gameOver() ) {
416
417                // get move location from client
418                String locations = input.readUTF();
419                int splitPoint = locations.indexOf( " " );
420                int oldLocation = Integer.parseInt(
421                   locations.substring( 0, splitPoint ) );
422                int newLocation = Integer.parseInt( locations.substring(
423                   splitPoint + 1, locations.length() ) );
424
425                // check for valid move
426               if ( validMove( oldLocation, newLocation, playerNumber ) ) {
427                   display( "Player" + playerNumber + ": [ " +
428                      oldLocation / 8 + " ][ " + oldLocation % 8 +
429                      " ] --> [ " + newLocation / 8 + " ][ " +
430                      newLocation % 8 + " ]" );
431                   output.writeUTF( "Valid move." );
432                   output.writeUTF(
```

```
433                      currentCell + " " + takeOver1 + " " + takeOver2 );
434
435                   takeOver1 = -1;  // reset takeOver1
436                   takeOver2 = -1;  // reset takeOver2
437                }
438                else
439                   output.writeUTF( "Invalid move, try again" );
440             }
441
442             // close connection to client
443             connection.close();
444
445          } // end try
446
447          // process problems communicating with client
448          catch( IOException ioException ) {
449             ioException.printStackTrace();
450             System.exit( 1 );
451          }
452
453       } // end method run
454
455       // set whether or not thread is suspended
456       public void setSuspended( boolean status )
457       {
458          suspended = status;
459       }
460
461    } // end class Player
462
463 } // end class CheckerGameServer
```

```
1  // Exercise 18.18 solution: CheckerGameClient.java
2  // Client that let a user play checker game with another across a network.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.net.*;
6  import java.io.*;
7  import java.util.*;
8  import javax.swing.*;
9
10 public class CheckerGameClient extends JApplet implements Runnable {
11    private JTextField idField;
12    private JTextArea displayArea;
13    private JPanel boardPanel, panel2;
14    private Square board[][], currentSquare, lastSquare;
15    private Socket connection;
16    private DataInputStream input;
17    private DataOutputStream output;
18    private Thread outputThread;
19    private char myMark;
20    private boolean myTurn;
21
```

```
22     // Set up user-interface and board
23     public void init()
24     {
25        Container container = getContentPane();
26
27        // set up JTextArea to display messages to user
28        displayArea = new JTextArea( 4, 30 );
29        displayArea.setEditable( false );
30        container.add( new JScrollPane( displayArea ), BorderLayout.SOUTH );
31
32        // set up panel for squares in board
33        boardPanel = new JPanel();
34        boardPanel.setLayout( new GridLayout( 8, 8, 0, 0 ) );
35
36        // create board
37        board = new Square[ 8 ][ 8 ];
38
39        // initialize board
40        for ( int row = 0; row < board.length; row++ ) {
41
42           for ( int column = 0; column < board[ row ].length; column++ ) {
43
44              // create Square
45              if ( row < 3 && row % 2 == column % 2 )
46                 board[ row ][ column ] = new Square( 'W', row, column );
47              else if ( row > 4 && row % 2 == column % 2 )
48                 board[ row ][ column ] = new Square( 'Y', row, column );
49              else
50                 board[ row ][ column ] = new Square( ' ', row, column );
51              boardPanel.add( board[ row ][ column ] );
52           }
53        }
54
55        // textfield to display player's mark
56        idField = new JTextField();
57        idField.setEditable( false );
58        container.add( idField, BorderLayout.NORTH );
59
60        // set up panel to contain boardPanel (for layout purposes)
61        panel2 = new JPanel();
62        panel2.add( boardPanel, BorderLayout.CENTER );
63        container.add( panel2, BorderLayout.CENTER );
64
65     } // end method init
66
67     // Make connection to server and get associated streams.
68     // Start separate thread to allow this applet to
69     // continually update its output in text area display.
70     public void start()
71     {
72        // connect to server, get streams and start outputThread
73        try {
74
```

```
75            // make connection
76            connection = new Socket(
77               InetAddress.getByName( "127.0.0.1" ), 12345 );
78
79            // get streams
80            input = new DataInputStream( connection.getInputStream() );
81            output = new DataOutputStream( connection.getOutputStream() );
82         }
83
84         // catch problems setting up connection and streams
85         catch ( IOException ioException ) {
86            ioException.printStackTrace();
87         }
88
89         // create and start output thread
90         outputThread = new Thread( this );
91         outputThread.start();
92
93      } // end method start
94
95      // control thread that allows continuous update of displayArea
96      public void run()
97      {
98         // get player's mark (W or Y)
99         try {
100           myMark = input.readChar();
101           idField.setText( "You are player \"" + myMark + "\"" );
102           myTurn = ( myMark == 'W' ? true : false );
103        }
104
105        // process problems communicating with server
106        catch ( IOException ioException ) {
107           ioException.printStackTrace();
108        }
109
110        // receive messages sent to client and output them
111        while ( true ) {
112
113           // read message from server and process message
114           try {
115              String message = input.readUTF();
116              processMessage( message );
117           }
118
119           // process problems communicating with server
120           catch ( IOException ioException ) {
121              ioException.printStackTrace();
122           }
123        }
124
125     }  // end method run
126
127     // process messages received by client
128     private void processMessage( String message )
129     {
```

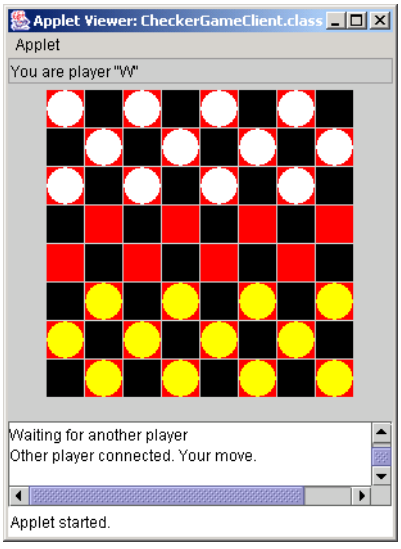
```
130         // valid move occurred
131         if ( message.equals( "Valid move." ) ) {
132            displayArea.append( "Valid move, please wait.\n" );
133
134            // update board
135            try {
136               StringTokenizer tokens =
137                  new StringTokenizer( input.readUTF() );
138               final int currentCell =
139                  Integer.parseInt( tokens.nextToken() );
140               final int take1 = Integer.parseInt( tokens.nextToken() );
141               final int take2 = Integer.parseInt( tokens.nextToken() );
142
143               // set mark in square from event-dispatch thread
144               SwingUtilities.invokeLater(
145
146                  new Runnable() {
147
148                     public void run()
149                     {
150                        int currentRow = currentCell / 8;
151                        int currentCol = currentCell % 8;
152                        currentSquare = board[ currentRow ][ currentCol ];
153                        currentSquare.setMark( myMark );
154                        lastSquare.setMark( ' ' );
155
156                        resetBoard( take1 );
157                        resetBoard( take2 );
158                     }
159                  }
160               );
161
162            } // end try
163
164   catch ( Exception exception ) {
165               exception.printStackTrace();
166            }
167
168         } // end if
169
170         // invalid move occurred
171         else if ( message.equals( "Invalid move, try again" ) ) {
172            displayArea.append( message + "\n" );
173            myTurn = true;
174         }
175
176         // opponent moved
177         else if ( message.equals( "Opponent moved" ) ) {
178
179            // get move location and update board
180            try {
181               StringTokenizer tokens =
182                  new StringTokenizer( input.readUTF() );
183               final int oldCell = Integer.parseInt( tokens.nextToken() );
```

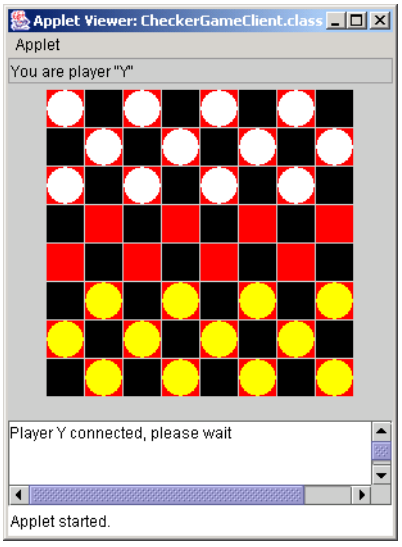
```
184             final int newCell = Integer.parseInt( tokens.nextToken() );
185             final int take1 = Integer.parseInt( tokens.nextToken() );
186             final int take2 = Integer.parseInt( tokens.nextToken() );
187
188             // set mark in square from event-dispatch thread
189             SwingUtilities.invokeLater(
190
191                new Runnable() {
192
193                   public void run()
194                   {
195                      int oldCellRow = oldCell / 8;
196                      int oldCellColumn = oldCell % 8;
197                      int newCellRow = newCell / 8;
198                      int newCellColumn = newCell % 8;
199
200                      board[ oldCellRow ][ oldCellColumn ].setMark( ' ' );
201                      board[ newCellRow ][ newCellColumn ].setMark(
202                         ( myMark == 'W' ? 'Y' : 'W' ) );
203
204                      resetBoard( take1 );
205                      resetBoard( take2 );
206
207                      displayArea.append( "Opponent moved. Your turn.\n" );
208
209                   } // end method run
210
211                } // end inner class Runnable
212
213             ); // end invokeLater
214
215             myTurn = true;
216
217          } // end try
218
219          // process problems communicating with server
220          catch ( IOException ioException ) {
221             ioException.printStackTrace();
222          }
223
224       } // end else if
225
226       // simply display message
227       else
228          displayArea.append( message + "\n" );
229
230       displayArea.setCaretPosition( displayArea.getText().length() );
231
232    }  // end method processMessage
233
234    public void resetBoard( int takeCell )
235    {
236       if ( takeCell != -1 ) {  // take opponent away
237          int takeRow = takeCell / 8;
```

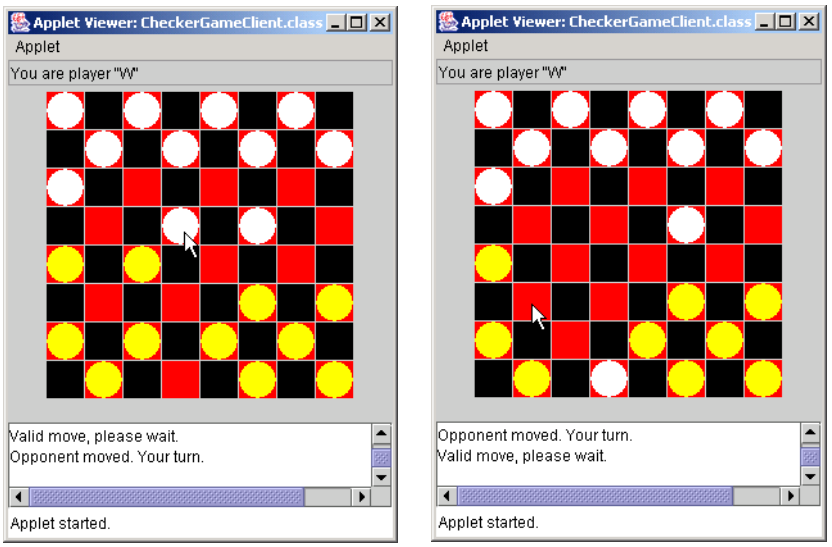
```
238          int takeColumn = takeCell % 8;
239          board[ takeRow ][ takeColumn ].setMark( ' ' );
240       }
241    }
242
243    // send message to server indicating clicked square
244    public void sendClickedSquare()
245    {
246       int lastLocation = lastSquare.getSquareLocation();
247       int newLocation = currentSquare.getSquareLocation();
248
249       if ( myTurn ) {
250
251          // send location to server
252          try {
253             output.writeUTF( lastLocation + " " + newLocation );
254             myTurn = false;
255          }
256
257          // process problems communicating with server
258          catch ( IOException ioException ) {
259             ioException.printStackTrace();
260          }
261       }
262
263    } // end method sendClickedSquare
264
265    // set current Square
266    public void setCurrentSquare( Square square )
267    {
268       currentSquare = square;
269    }
270
271    // set last Square
272    public void setLastSquare( Square square )
273    {
274       lastSquare = square;
275    }
276
277    // private class for the squares on the board
278    private class Square extends JPanel {
279       private char mark;
280       private int row, column;
281
282       public Square( char squareMark, int r, int c )
283       {
284          mark = squareMark;
285          row = r;
286          column = c;
287
288          addMouseListener(
289
290             new MouseAdapter() {
291
```

```
292                public void mouseClicked( MouseEvent e )
293                {
294                   if ( e.getButton() == MouseEvent.BUTTON1 )
295                      setLastSquare( Square.this );
296                   else {
297                      setCurrentSquare( Square.this );
298                      sendClickedSquare();
299                   }
300                }
301             }
302          );
303
304       }  // end Square constructor
305
306       // return preferred size of Square
307       public Dimension getPreferredSize()
308       {
309          return new Dimension( 30, 30 );
310       }
311
312       // return minimum size of Square
313       public Dimension getMinimumSize()
314       {
315          return getPreferredSize();
316       }
317
318       // set mark for Square
319       public void setMark( char newMark )
320       {
321          mark = newMark;
322          repaint();
323       }
324
325       // return Square location
326       public int getSquareLocation()
327       {
328          return row * 8 + column;
329       }
330
331       // draw Square
332       public void paintComponent( Graphics g )
333       {
334          super.paintComponent( g );
335
336          if ( row % 2 == column % 2 ) {
337             g.setColor( Color.RED );
338             g.fillRect( 0, 0, 29, 29 );
339             g.setColor( Color.BLACK );
340          }
341          else
342             g.fillRect( 0, 0, 29, 29 );
343
344          if ( mark == 'W' ) {
345             g.setColor( Color.WHITE );
```

```
346             g.fillOval( 0, 0, 29, 29 );
347          }
348          else if ( mark == 'Y' ) {
349             g.setColor( Color.YELLOW );
350             g.fillOval( 0, 0, 29, 29 );
351          }
352          else {  // mark == ' ', reset the board
353             if ( row % 2 == column % 2 ) {
354                g.setColor( Color.RED );
355                g.fillRect( 0, 0, 29, 29 );
356                g.setColor( Color.BLACK );
357             }
358             else
359                g.fillRect( 0, 0, 29, 29 );
360       }
361
362        } // end method paintComponent
363
364     } // end class Square
365
366  } // end class CheckerGameClient
```







18.19 Develop a chess-playing program modeled after the checkers program in the Exercise 18.18.

18.20 Develop a Blackjack card game program in which the server application deals cards to each of the client applets. The server should deal additional cards (as per the rules of the game) to each player as requested.

18.21 Develop a Poker card game in which the server application deals cards to each of the client applets. The server should deal additional cards (as per the rules of the game) to each player as requested.

18.22 *(Modifications to the Multithreaded Tic-Tac-Toe Program)* The programs of Fig. 18.8 and Fig. 18.9 implemented a multithreaded, client/server version of the game Tic-Tac-Toe. Our goal in developing this game was to demonstrate a multithreaded server that could process multiple connections from clients at the same time. The server in the example is really a mediator between the two client applets—it makes sure that each move is valid and that each client moves in the proper order. The server does not determine who won or lost or if there was a draw. Also, there is no capability to allow a new game to be played or to terminate an existing game.

The following is a list of suggested modifications to Fig. 18.8 and Fig. 18.9:

a) Modify the `TicTacToeServer` class to test for a win, loss or draw on each move in the game. Send a message to each client applet that indicates the result of the game when the game is over.

b) Modify the `TicTacToeClient` class to display a button that when clicked allows the client to play another game. The button should be enabled only when a game completes. Note that both class `TicTacToeClient` and class `TicTacToeServer` must be modified to reset the board and all state information. Also, the other `TicTacToeClient` should be notified that a new game is about to begin so its board and state can be reset.

c) Modify the `TicTacToeClient` class to provide a button that allows a client to terminate the program at any time. When the user clicks the button, the server and the other client should be notified. The server should then wait for a connection from another client so a new game can begin.

d) Modify the `TicTacToeClient` class and the `TicTacToeServer` class so the winner of a game can choose game piece X or O for the next game. Remember: X always goes first.

e) If you would like to be ambitious, allow a client to play against the server while the server waits for a connection from another client.

18.23 *(3-D Multithreaded Tic-Tac-Toe)* Modify the multithreaded, client/server Tic-Tac-Toe program to implement a three-dimensional 4-by-4-by-4 version of the game. Implement the server application to mediate between the two clients. Display the three-dimensional board as four boards containing four rows and four columns each. If you would like to be ambitious, try the following modifications:

a) Draw the board in a three-dimensional manner.
b) Allow the server to test for a win, loss or draw. Beware! There are many possible ways to win on a 4-by-4-by-4 board!

18.24 *(Networked Morse Code)* Modify your solution to Exercise 11.27 to enable two applets to send Morse Code messages to each other through a multithreaded server application. Each applet should allow the user to type normal characters in `JTextArea`s, translate the characters into Morse Code and send the coded message through the server to the other client. When messages are received, they should be decoded and displayed as normal characters and as Morse Code. The applet should have two `JTextArea`s: one for displaying the other client's messages and one for typing.

ANS:

```
1  // Exercise 18.24 solution: MorseCodeServer.java
2  // This class maintains morse code for two client applets.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.net.*;
6  import java.io.*;
7  import javax.swing.*;
8
9  public class MorseCodeServer {
10    private Player players[];
11    private ServerSocket server;
12
13    // set up morse code server and GUI that displays messages
14    public MorseCodeServer()
15    {
16       players = new Player[ 2 ];
17
18       // set up ServerSocket
19       try {
20          server = new ServerSocket( 12345, 2 );
21       }
22
23       // process problems creating ServerSocket
24       catch( IOException ioException ) {
25          ioException.printStackTrace();
26          System.exit( 1 );
27       }
28
29    } // end constructor MorseCodeServer
30
31    // wait for two connections so communication can be started
32    public void execute()
33    {
```

```
34          // wait for each client to connect
35          for ( int i = 0; i < players.length; i++ ) {
36
37             // wait for connection, create Player, start thread
38             try {
39                players[ i ] = new Player( server.accept(), i );
40                players[ i ].start();
41             }
42
43             // process problems receiving connection from client
44             catch( IOException ioException ) {
45                ioException.printStackTrace();
46                System.exit( 1 );
47             }
48          }
49
50          // Player 0 is suspended until Player 1 connects.
51          // Resume player 0 now.
52          synchronized ( players[ 0 ] ) {
53             players[ 0 ].setSuspended( false );
54             players[ 0 ].notify();
55          }
56
57       }  // end method execute
58
59       // This method is synchronized because only one message can be encoded.
60       public synchronized void translate( String phrase, int playerNumber )
61       {
62          // the numbers from 0 to 9
63          String[] numbers = { "-----", ".----", "..---", "...--", "....-",
64             ".....", "-....", "--...", "---..", "----." };
65
66          // the letters from a to z
67          String[] letters = { ".-", "-...", "-.-.", "-..", ".", "..-.",
68             "--.", "....", "..", ".---", "-.-", ".-..", "--", "-.", "---",
69             ".--.", "--.-", ".-.", "...", "-", "..-", "...-",
70             ".--", "-..-", "-.--", "--.." };
71
72          String morseCode = "";  // morseCode of the phrase
73
74          // loop through the string
75          for ( int i = 0; i < phrase.length(); i++ ) {
76             char alpha = phrase.charAt( i );
77
78             // if the character is a number, access the number array
79             if ( Character.isDigit( alpha ) )
80                morseCode += numbers[ alpha - 48 ] + " ";
81
82             // if the character is a letter, access the letter array
83             if ( Character.isLetter( alpha ) )
84                morseCode +=
85                   letters[ Character.toUpperCase( alpha ) - 65 ] + " ";
86
```

```
87             // if the character is a space, output two extra spaces
88             if ( alpha == ' ' )
89                morseCode += "   ";
90          }
91
92          // let other player know the morseCode
93          if ( playerNumber == 0 )
94             players[ 1 ].showMorseCode( morseCode );
95          else
96             players[ 0 ].showMorseCode( morseCode );
97
98          // tell waiting player to continue
99          notify();
100
101       } // end method translate
102
103       public static void main( String args[] )
104       {
105          MorseCodeServer application = new MorseCodeServer();
106          application.execute();
107       }
108
109       // private inner class Player manages each Player as a thread
110       private class Player extends Thread {
111          private Socket connection;
112          private DataInputStream input;
113          private DataOutputStream output;
114          private int playerNumber;
115          protected boolean suspended = true;
116
117          // set up Player thread
118          public Player( Socket socket, int number )
119          {
120             playerNumber = number;
121
122             connection = socket;
123
124             // obtain streams from Socket
125             try {
126                input = new DataInputStream( connection.getInputStream() );
127                output = new DataOutputStream( connection.getOutputStream() );
128             }
129
130             // process problems getting streams
131             catch( IOException ioException ) {
132                ioException.printStackTrace();
133                System.exit( 1 );
134             }
135
136          } // end constructor Player
137
138          // send message that other player write phrase
139          public void showMorseCode( String morseCode )
140          {
```

```
141          // send message showing morseCode
142          try {
143             output.writeUTF( morseCode );
144          }
145
146          // process problems sending message
147          catch ( IOException ioException ) {
148             ioException.printStackTrace();
149          }
150       }
151
152       // control thread's execution
153       public void run()
154       {
155          // send client message indicating its number,
156          // process messages from client
157          try {
158
159             // if player 0, wait for another player to arrive
160             if ( playerNumber == 0 ) {
161
162                // wait for player 0
163                try {
164                   synchronized( this ) {
165                      while ( suspended )
166                         wait();
167                   }
168                }
169
170                // process interruptions while waiting
171                catch ( InterruptedException exception ) {
172                   exception.printStackTrace();
173                }
174
175             } // end if
176
177             // keep running
178             while ( true ) {
179
180                // get input phrase from client
181                String phrase = input.readUTF();
182
183                // translate phrase
184                translate( phrase, playerNumber );
185             }
186
187          } // end try
188
189          // process problems communicating with client
190          catch( IOException ioException ) {
191             ioException.printStackTrace();
192             System.exit( 1 );
193          }
194
```

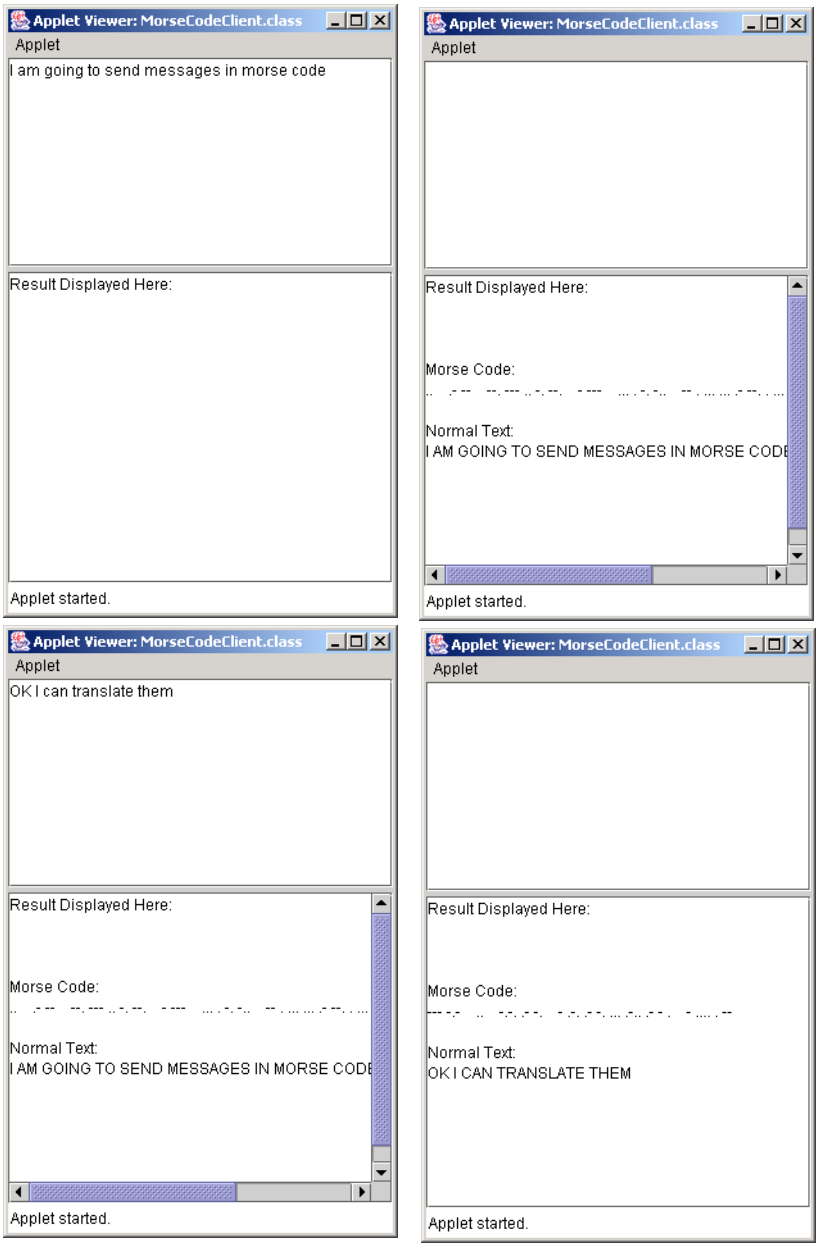
```
195       } // end method run
196
197       // set whether or not thread is suspended
198       public void setSuspended( boolean status )
199       {
200          suspended = status;
201       }
202
203    } // end class Player
204
205 } // end class MorseCodeServer
```

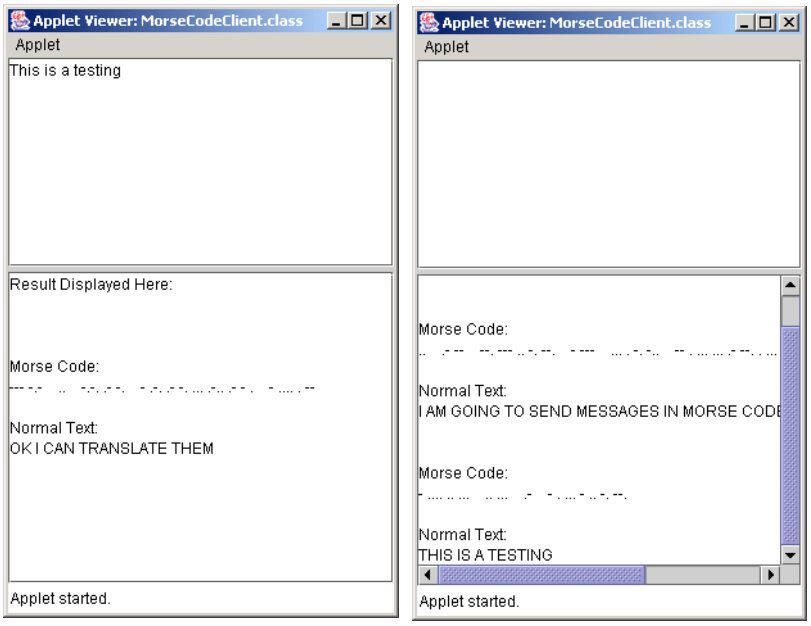
```
1  // Exercise 18.24 solution: MorseCodeClient.java
2  // Client that let a user communicate with another across a network.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.net.*;
6  import java.io.*;
7  import java.util.*;
8  import javax.swing.*;
9
10 public class MorseCodeClient extends JApplet implements Runnable {
11    private JTextArea inputArea, displayArea;
12    private Socket connection;
13    private DataInputStream input;
14    private DataOutputStream output;
15    private Thread outputThread;
16
17    // Set up user-interface and board
18    public void init()
19    {
20       Container container = getContentPane();
21
22       // set up JTextArea to input and display messages
23       inputArea = new JTextArea( 10, 10 );
24       container.add( new JScrollPane( inputArea ), BorderLayout.NORTH );
25       inputArea.addKeyListener(
26
27          new KeyAdapter() {
28
29             public void keyPressed( KeyEvent event )
30             {
31                try {
32
33                   if ( event.getKeyCode() == KeyEvent.VK_ENTER ) {
34                      output.writeUTF( inputArea.getText() );
35                      inputArea.setText( " " );
36                   }
37                }
38                catch ( IOException exception ) {
39                   exception.printStackTrace();
40                }
41             }
```

```
42
43          } // end inner class KeyAdapter
44
45       ); // end addKeyListener
46
47       displayArea = new JTextArea( 15, 15 );
48       displayArea.setEditable( false );
49       displayArea.setText( "Result Displayed Here: \n" );
50       container.add( new JScrollPane( displayArea ), BorderLayout.SOUTH );
51
52    } // end method init
53
54    // Make connection to server and get associated streams.
55    // Start separate thread to allow this applet to
56    // continually update its output in text area display.
57    public void start()
58    {
59       // connect to server, get streams and start outputThread
60       try {
61
62          // make connection
63          connection = new Socket(
64             InetAddress.getByName( "127.0.0.1" ), 12345 );
65
66          // get streams
67          input = new DataInputStream( connection.getInputStream() );
68          output = new DataOutputStream( connection.getOutputStream() );
69       }
70
71       // catch problems setting up connection and streams
72       catch ( IOException ioException ) {
73          ioException.printStackTrace();
74       }
75
76       // create and start output thread
77       outputThread = new Thread( this );
78       outputThread.start();
79
80    } // end method start
81
82    // control thread that allows continuous update of displayArea
83    public void run()
84    {
85       // receive messages sent to client and output them
86       while ( true ) {
87
88          // read message from server and process message
89          try {
90             String message = input.readUTF();
91             processMessage( message );
92          }
93
94          // process problems communicating with server
95          catch ( IOException ioException ) {
```

```
96                ioException.printStackTrace();
97             }
98          }
99
100      }  // end method run
101
102      // process morseCode received by client
103      private void processMessage( String morseCode )
104      {
105         final String input = morseCode;
106
107         // translate morse code to normal characters
108         final String message = translate( morseCode );
109
110         // set mark in square from event-dispatch thread
111         SwingUtilities.invokeLater(
112
113            new Runnable() {
114
115               public void run()
116               {
117                  displayArea.append( "\n\n\nMorse Code: \n" + input +
118                     "\n\nNormal Text: \n" + message );
119               }
120            }
121         );
122
123      }  // end method processMessage
124
125      // translate morse code phrase to normal text
126      private String translate( String morseCode )
127      {
128         String result = "";
129         int start = 0, length = 0;
130         int threeSpaces = morseCode.indexOf( "   " );
131         String word;
132
133         // while not reach the end of morse code
134         while ( length < morseCode.length() ) {
135
136            if ( threeSpaces != -1 ) {
137               word = morseCode.substring( start, threeSpaces );
138               length = threeSpaces;
139            }
140            else {
141               word = morseCode.substring( start, morseCode.length() );
142               length = morseCode.length();
143            }
144
145            StringTokenizer letters = new StringTokenizer( word );
146
147            // decode letter
148            while ( letters.hasMoreTokens() )
149               result += decode( letters.nextToken() );
```

```
150
151            result += " ";
152            start = threeSpaces + 3;
153            threeSpaces = morseCode.indexOf( "   ", start );
154
155         } // end while
156
157         return result;
158
159      } // end method translate
160
161      // decode morse code letter
162      private String decode( String morseCode )
163      {
164         // morse code numbers and letters
165         String[] morseCharacters = { "-----", ".----", "..---", "...--",
166            "....-", ".....", "-....", "--...", "---..", "----.", ".-",
167            "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---",
168            "-.-", ".-..", "--", "-.", "---", ".--.", "--.-", ".-.", "...",
169            "-", "..-", "...-", ".--", "-..-", "-.--", "--.." };
170
171         // normal English characters
172         String[] normalCharacters = { "0", "1", "2", "3", "4", "5", "6",
173            "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J",
174            "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W",
175            "X", "Y", "Z" };
176
177         for ( int i = 0; i < morseCharacters.length; i++ )
178
179            if ( morseCode.equals( morseCharacters[ i ] ) )
180               return normalCharacters[ i ];
181
182         return "";
183
184      } // end method decode
185
186   } // end class MorseCodeClient
```





19

Multimedia: Images, Animation, Audio and Video

Objectives

- To understand how to get and display images.
- To create animations from sequences of images.
- To create image maps.
- To be able to get, play, loop and stop sounds, using an `AudioClip`.

The wheel that squeaks the loudest … gets the grease.
John Billings (Henry Wheeler Shaw)

We'll use a signal I have tried and found far-reaching and easy to yell. Waa-hoo!
Zane Grey

There is a natural hootchy-kootchy motion to a goldfish.
Walt Disney

Between the motion and the act falls the shadow.
Thomas Stearns Eliot

SELF-REVIEW EXERCISES

19.1 Fill in the blanks in each of the following statements:

a) `Applet` method _____ loads an image into an applet.

ANS: `getImage`

b) `Applet` method _____ returns, as an object of class `URL`, the location on the Internet of the HTML file that invoked the applet.

ANS: `getDocumentBase`

c) `Graphics` method _____ displays an image on an applet.

ANS: `drawImage`

d) Java provides two mechanisms for playing sounds in an applet—the `Applet`'s `play` method and the `play` method from the _____ interface.

ANS: `AudioClip`

e) A(n) _____ is an image that has *hot areas* that the user can click to accomplish a task such as loading a different Web page.

ANS: image map

f) Method _____ of class `ImageIcon` displays the `ImageIcon`'s image.

ANS: `paintIcon`

g) Java supports several image formats, including _____, _____ and _____.

ANS: Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG) Portable Network Graphics (PNG)

19.2 Determine whether each of the following statements is true or false. If false, explain why.

a) A sound will be garbage collected as soon as it has finished playing.

ANS: False. The sound will be eligible for garbage collection (if it is not referenced by an `AudioClip`) and will be garbage collected when the garbage collector is able to run.

b) Class `ImageIcon` provides constructors that allow an `ImageIcon` object to be initialized only with an image from the local computer.

ANS: False. `ImageIcon` can load images from the Internet as well.

EXERCISES

19.3 Describe how to make an animation "browser friendly."

ANS: Begin the animation in the `start` method and suspend/terminate the animation in the `stop` method.

19.4 Describe the Java methods for playing and manipulating audio clips.

ANS: Class `Applet` method `play` and the `AudioClip` interface method `play` both load the sound and play it once. `AudioClip` method `loop` continuously loops the audio clip in the background. `AudioClip` method `stop` terminates an audio clip that is currently playing.

19.5 Explain how image maps are used. List 10 examples in which image maps are used.

ANS: Image maps are used to create interactive Web pages: i) load different Web paged based on user choice. ii) show descriptive message for images within the image maps. iii) define regions for an image, when mouse enters the image region, display the image. iv) when mouse clicks the image region, link the user to the URL associated with this image.

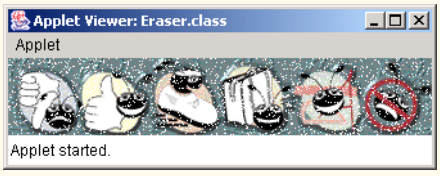
19.6 *(Randomly Erasing an Image)* Suppose an image is displayed in a rectangular screen area. One way to erase the image is simply to set every pixel to the same color immediately, but this is a dull visual effect. Write a Java program that displays an image and then erases it by using random-number generation to select individual pixels to erase. After most of the image is erased, erase all of the remaining pixels at once. You can draw individual pixels as a line that starts and ends at the same

coordinates. You might try several variants of this problem. For example, you might display lines randomly or display shapes randomly to erase regions of the screen.

ANS:

```
1  // Exercise 19.6 Solution: Eraser.java
2  // Program slowly erases an image.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Eraser extends JApplet implements ActionListener {
8     private ImageIcon image;
9     private int imageWidth, imageHeight, count, numberOfTimes;
10    private boolean showImage;
11    private Timer timer;
12
13    // initialize variables, start timer
14    public void init()
15    {
16       showImage = true;
17       count = 0;
18
19       image = new ImageIcon( "icons2.gif" );
20       timer = new Timer( 1, this );
21       timer.start();
22
23       imageWidth = image.getIconWidth();
24       imageHeight = image.getIconHeight();
25       numberOfTimes = imageWidth * imageHeight;
26    }
27
28    // draw on JApplet
29    public void paint( Graphics g )
30    {
31       // draw image only once
32       if ( showImage == true ) {
33          image.paintIcon( this, g, 0, 0 );
34          showImage = false;
35       }
36
37       g.setColor( getBackground() );
38
39       // loop to increase speed
40       for ( int reps = 0; reps < 30; reps++ ) {
41
42          // generate random coordinates within image
43          int x = ( int ) ( Math.random() * imageWidth );
44          int y = ( int ) ( Math.random() * imageHeight );
45
46          // erase random pixels
47          g.drawLine( x, y, x, y );
48       }
49
```

```
50          // erase remaining pixels when most of image has been erased
51          if ( count > numberOfTimes * .95 ) {
52             g.fillRect( 0, 0, imageWidth, imageHeight );
53          }
54
55          count += 30;
56
57       } // end method paint
58
59       // respond to Timer's events
60       public void actionPerformed( ActionEvent event )
61       {
62          repaint();
63       }
64
65    }  // end class Eraser
```



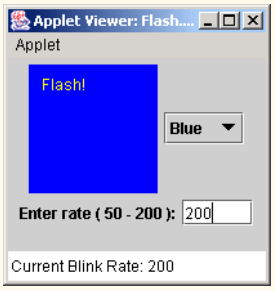
19.7 *(Text Flasher)* Create a Java program that repeatedly flashes text on the screen. Do this by alternating the text with a plain background-color image. Allow the user to control the "blink speed" and the background color or pattern.

ANS:

```
1  // Exercise 19.7 Solution: Flash.java
2  // Program flashes text.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Flash extends JApplet {
8     private MyCanvas theCanvas;
9     private JComboBox colorSelect;
10    private JLabel prompt;
11    private JTextField input;
12
13    // initialize values and set up GUI
14    public void init()
15    {
16       // colors to set background to flash
17       String items[] = { "Black", "Red", "Blue", "Green" };
18
19       // create components
20       prompt = new JLabel( "Enter rate ( 50 - 200 ):" );
21       input = new JTextField( 5 );
22       input.addActionListener(
23
```

```
24          new ActionListener() {
25
26             public void actionPerformed( ActionEvent event )
27             {
28
29                // changes the rate at which the canvas flashes
30                theCanvas.setSleepTime(
31                   Integer.parseInt( input.getText() ) );
32                showStatus(
33                   "Current Blink Rate: " + theCanvas.getSleepTime() );
34             }
35          }
36       );
37
38       theCanvas = new MyCanvas();
39       colorSelect = new JComboBox( items );
40       colorSelect.addItemListener(
41
42          new ItemListener() {
43
44             // changes the background color based on selection
45             public void itemStateChanged( ItemEvent event )
46             {
47                Color color;
48
49                if ( event.getItem().equals( "Black" ) )
50                   color = Color.black;
51                else if ( event.getItem().equals( "Red" ) )
52                   color = Color.red;
53                else if ( event.getItem().equals( "Blue" ) )
54                   color = Color.blue;
55                else
56                   color = Color.green;
57
58                theCanvas.setBackground( color );
59             }
60          }
61       );
62
63       // add components to container
64       Container container = getContentPane();
65       container.setLayout( new FlowLayout() );
66
67       container.add( theCanvas );
68       container.add( colorSelect );
69       container.add( prompt );
70       container.add( input );
71
72    } // end method init
73
74 }  // end class Flash
75
76 // creates the screen on which color and text flash
77 class MyCanvas extends JPanel implements ActionListener {
```

```
78       private String text;
79       private Timer timer;
80       private Color color = Color.black;
81       boolean flash = true;
82
83       // constructor
84       public MyCanvas()
85       {
86          setBackground( Color.black );
87          timer = new Timer( 150, this );
88          timer.start();
89          text = "Flash!";
90          setSize( 100, 100 );
91          setOpaque( true );
92       }
93
94       // draw text
95       public synchronized void paintComponent( Graphics g )
96       {
97          super.paintComponent( g );
98
99          if ( flash ) {
100            g.setColor( Color.yellow );
101            g.drawString( text, 10, 20 );
102         }
103      }
104
105      // changes whether or not the text is drawn
106      public synchronized void actionPerformed( ActionEvent event )
107      {
108         flash = !flash;
109         repaint();
110      }
111
112      // sets the flash rate
113      public void setSleepTime( int time )
114      {
115         timer.setDelay( time >= 50 && time <= 200 ? time : 150 );
116      }
117
118      // returns the current flash rate
119      public int getSleepTime()
120      {
121         return timer.getDelay();
122      }
123
124      // used to size the window
125      public Dimension getPreferredSize()
126      {
127         return new Dimension( 100, 100 );
128      }
129
130   } // end class MyCanvas
```



19.8 *(Image Flasher)* Create a Java program that repeatedly flashes an image on the screen. Do this by alternating the image with a plain background-color image.

ANS:

```
1  // Exercise 19.8 Solution: Flash2.java
2  // Program flashes an image.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Flash2 extends JPanel implements ActionListener {
8     private ImageIcon image;
9     private int imageWidth, imageHeight;
10    private boolean flash;
11    private Timer timer;
12
13    // initialize variables, set background color, start timer
14    public void init()
15    {
16       flash = true;
17
18       setBackground( Color.blue );
19
20       image = new ImageIcon( "icons2.gif" );
21       imageWidth = image.getIconWidth();
22       imageHeight = image.getIconHeight();
23
24       // create and start Timer
25       timer = new Timer( 500, this );
26       timer.start();
27    }
28
29    // draw on JApplet
30    public void paint( Graphics g )
31    {
32       g.setColor( getBackground() );
33
34       // draw image
35       if ( flash == true )
36          image.paintIcon( this, g, 20, 20 );
37
```

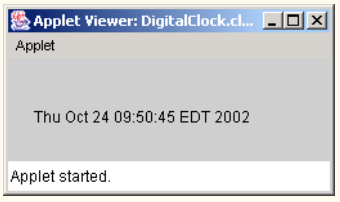
```
38          // draw rectangle over image
39          else
40             g.fillRect( 20, 20, imageWidth, imageHeight );
41       }
42
43       // respond to Timer's events
44       public void actionPerformed( ActionEvent event )
45       {
46          flash = !flash;
47          repaint();
48       }
49
50       public static void main( String args[] )
51       {
52          Flash2 flash2 = new Flash2();
53          JFrame window = new JFrame( "Flashing Images" );
54          window.getContentPane().add( flash2 );
55          window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
56          window.pack();
57          window.setSize( 380, 120 );
58          window.setVisible( true );
59       }
60
61    }  // end class Flash2
```

19.9 *(Digital Clock)* Implement a program that displays a digital clock on the screen. You might add options to scale the clock; display the day, month and year; issue an alarm; play certain audios at designated times and the like.

ANS:

```
1  // Exercise 19.9 Solution: DigitalClock.java
2  // Program creates a digital clock.
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.util.*;
7
8  public class DigitalClock extends JApplet implements ActionListener {
9     private String theTime;
10    private javax.swing.Timer t;
11
12    public void init()
13    {
14       theTime = "";
15       t = new javax.swing.Timer( 1000, this );
16    }
```

```
17
18    public void paint( Graphics g )
19    {
20       super.paint( g );  // clears the background
21
22       g.drawString( theTime, 20, 50 );
23    }
24
25    public void start()
26    {
27       t.start();
28    }
29
30    public void stop()
31    {
32       t.stop();
33    }
34
35    public void actionPerformed( ActionEvent e )
36    {
37       theTime = new Date().toString();
38       repaint();
39    }
40
41 } // end class DigitalClock
```



19.10 *(Calling Attention to an Image)* If you want to emphasize an image, you might place a row of simulated light bulbs around your image. You can let the light bulbs flash in unison, or you can let them fire on and off in sequence one after the other.

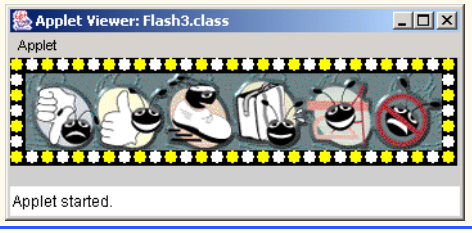
ANS:

```
1  // Exercise 19.10 Solution: Flash3.java
2  // Program highlights an image.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Flash3 extends JApplet {
8     private MyCanvas theCanvas;
9
10    // set up GUI to contain a MyCanvas
11    public void init()
12    {
13       // create an image icon
14       ImageIcon image1 = new ImageIcon( "icons2.gif" );
```

```
15
16          int width = image1.getIconWidth();
17          int height = image1.getIconHeight();
18
19          int wide = 12;
20          int high = 12;
21          Image image2 = createImage( wide, high );
22          Image image3 = createImage( wide, high );
23
24          // create canvas and add to GUI
25          theCanvas = new MyCanvas( image1.getImage(),
26             image2, image3, width, height, wide, high );
27          getContentPane().add( theCanvas, BorderLayout.CENTER );
28       }
29
30    }  // end class Flash3
31
32    // MyCanvas displays a flashing border
33    class MyCanvas extends JPanel implements ActionListener {
34       private Image image1, image2, image3;
35       private Graphics graph2, graph3;
36       private Image[] lights;
37       private int numLights, rows, columns,
38          count, wide, high, width, height;
39       private Timer timer;
40
41       // constructor
42       public MyCanvas( Image i1, Image i2, Image i3,
43          int w, int h, int w2, int h2 )
44       {
45          int width = w;
46          int height = h;
47          wide = w2;
48          high = h2;
49          image1 = i1;
50          image2 = i2;
51          image3 = i3;
52
53          // create and start a timer
54          timer = new Timer( 300, this );
55          timer.start();
56
57          // create a yellow light
58          graph2 = image2.getGraphics();
59          graph2.setColor( Color.black );
60          graph2.fillRect( 0, 0, wide, high );
61          graph2.setColor( Color.yellow );
62          graph2.fillOval( 0, 0, 10, 10 );
63
64          // create a white light
65          graph3 = image3.getGraphics();
66          graph3.setColor( Color.black );
67          graph3.fillRect( 0, 0, wide, high );
68          graph3.setColor( Color.white );
```

```
69          graph3.fillOval( 0, 0, 10, 10 );
70
71          // initialize values and create array
72          rows = width / wide + 2;
73          columns = height / high;
74          numLights = 2 * rows + 2 * columns;
75          lights = new Image[ numLights ];
76
77          // used to make the lights blink
78          int count = 0;
79
80          setSize( wide * rows, high * ( columns + 2 ) );
81
82          repaint();
83       }
84
85       // override to produce blinking lights effect
86       public void paintComponent( Graphics g )
87       {
88          super.paintComponent( g );
89
90          // determine whether light is "on" or "off"
91          for ( int on = 0; on < numLights; on += 2 ) {
92
93             if ( count == 0 ) {
94                lights[ on ] = image2;
95                lights[ on + 1 ] = image3;
96             }
97
98             else {
99                lights[ on ] = image3;
100               lights[ on + 1 ] = image2;
101            }
102         }
103
104         // actually set position of light and draw
105         for ( int on = 0; on < numLights; on++ ) {
106
107            // reposition all lights
108            // top side, left to right
109            if ( on < rows )
110               g.drawImage( lights[ on ], on * wide, 0, this );
111
112            // right side, top to bottom
113            else if ( on >= rows && on < ( rows + columns ) )
114               g.drawImage( lights[ on ], (rows - 1) * wide,
115                  ( on - rows + 1 ) * high, this );
116
117            // bottom side, right to left
118            else if ( on >= ( rows + columns ) &&
119               on < ( rows * 2 + columns ) ) {
120
121               int xValue = rows - ( on - (rows + columns) ) - 1;
122               g.drawImage( lights[ on ],  xValue * wide,
```

```
123                ( columns + 1 ) * high, this );
124          }
125
126          // left side, bottom to top
127          else {
128             int yValue =
129                columns - ( on - ( 2 * rows ) - columns );
130             g.drawImage( lights[ on ], 0, yValue * high, this );
131          }
132
133       } // end for loop
134
135       g.drawImage( image1, wide, high, wide * (rows - 2 ),
136          high * columns, this );
137
138       // Change ++count to y to get the blinking effect
139       count = ++count % 2;
140
141    }  // end method paintComponent
142
143    // action to perform when timer indicates
144    public void actionPerformed( ActionEvent event )
145    {
146       repaint();
147    }
148
149 }  // end class MyCanvas
```

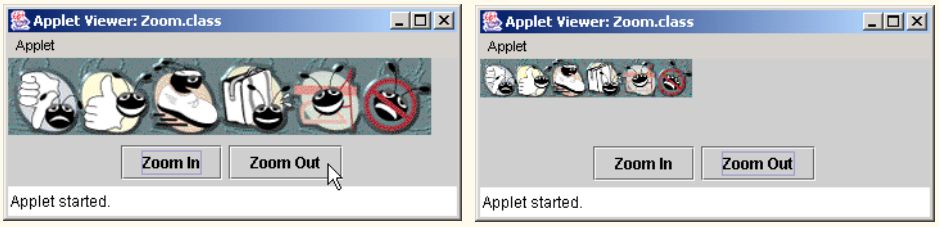


19.11 *(Image Zooming)* Create a program that enables you to zoom in on or away from an image.
ANS:

```
1  // Exercise 19.11 Solution: Zoom.java
2  // Program zooms an image.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class Zoom extends JApplet
8  {
9     private ImageIcon image;
10    private JButton zoomIn, zoomOut;
11    private JPanel drawingPanel, buttonPanel;
12    private int imageWidth, imageHeight;
13
```

```
14    // set up GUI, initialize variables
15    public void init()
16    {
17       image = new ImageIcon( "icons2.gif" );
18
19       buttonPanel = new JPanel();
20       zoomIn = new JButton( "Zoom In" );
21       zoomIn.addActionListener(
22
23          new ActionListener() {
24
25             public void actionPerformed( ActionEvent event )
26             {
27                // zoom in
28                imageWidth *= 2;
29                imageHeight *= 2;
30
31                // refresh image
32                repaint();
33             }
34          }
35       );
36       buttonPanel.add( zoomIn );
37
38       zoomOut = new JButton( "Zoom Out" );
39       zoomOut.addActionListener(
40
41          new ActionListener() {
42
43             public void actionPerformed( ActionEvent event )
44             {
45                // zoom out
46                imageWidth /= 2;
47                imageHeight /= 2;
48
49                // refresh image
50                repaint();
51             }
52          }
53       );
54       buttonPanel.add( zoomOut );
55
56       imageWidth = image.getIconWidth();
57       imageHeight = image.getIconHeight();
58
59       drawingPanel = new JPanel();
60       drawingPanel.setSize( 800, 200 );
61
62       // add components to content pane
63       Container container = getContentPane();
64       container.add( drawingPanel, BorderLayout.CENTER );
65       container.add( buttonPanel, BorderLayout.SOUTH );
66
67    } // end method init
```

```
68
69     // draw image with appropriate dimensions
70     public void paint( Graphics g )
71     {
72        super.paint( g );
73
74        g.drawImage( image.getImage(), 0, 0, imageWidth,
75           imageHeight, drawingPanel );
76
77        buttonPanel.repaint();
78     }
79
80  }  // end class Zoom
```



SPECIAL SECTION: CHALLENGING MULTIMEDIA PROJECTS

The preceding exercises are keyed to the text and designed to test the reader's understanding of fundamental multimedia concepts. This section includes a collection of advanced multimedia projects. The reader should find these problems challenging, yet entertaining. The problems vary considerably in difficulty. Some require an hour or two of program writing and implementation. Others are useful for lab assignments that might require two or three weeks of study and implementation. Some are challenging term projects. [*Note*: Solutions are not provided for these exercises.].

19.12 *(Animation)* Create a a general purpose Java animation program. Your program should allow the user to specify the sequence of frames to be displayed, the speed at which the images are displayed, audios that should be played while the animation is running and so on.

19.13 *(Limericks)* Modify the limerick-writing program you wrote in Exercise 10.10 to sing the limericks your program creates.

19.14 *(Random Inter-Image Transition)* This provides a nice visual effect. If you are displaying one image in a given area on the screen and you would like to transition to another image in the same screen area, store the new screen image in an off-screen buffer and randomly copy pixels from the new image to the display area, overlaying the previous pixels at those locations. When the vast majority of the pixels have been copied, copy the entire new image to the display area to be sure you are displaying the complete new image. To implement this program, you may need to use the `PixelGrabber` and `MemoryImageSource` classes (see the Java API documentation for descriptions of these classes). You might try several variants of this problem. For example, try selecting all the pixels in a randomly selected straight line or shape in the new image, and overlay those pixels above the corresponding positions of the old image.

19.15 *(Background Audio)* Add background audio to one of your favorite applications by using the `loop` method of class `AudioClip` to play the sound in the background while you interact with your application in the normal way.

19.16 *(Scrolling Marquee Sign)* Create a Java program that scrolls dotted characters from right to left (or from left to right if that is appropriate for your language) across a Marquee-like display sign. As an option, display the text in a continuous loop, so that after the text disappears at one end, it reappears at the other end.

19.17 *(Scrolling Image Marquee)* Create a Java program that scrolls an image across a Marquee screen.

19.18 *(Analog Clock)* Create a Java program that displays an analog clock with hour, minute and second hands that move appropriately as the time changes.

19.19 *(Dynamic Audio and Graphical Kaleidoscope)* Write a kaleidoscope program that displays reflected graphics to simulate the popular children's toy. Incorporate audio effects that "mirror" your program's dynamically changing graphics.

19.20 *(Automatic Jigsaw Puzzle Generator)* Create a Java jigsaw puzzle generator and manipulator. The user specifies an image. Your program loads and displays the image. Your program then breaks the image into randomly selected shapes and shuffles the shapes. The user then uses the mouse to move the pieces around to solve the puzzle. Add appropriate audio sounds as the pieces are being moved around and snapped back into place. You might keep tabs on each piece and where it really belongs; then use audio effects to help the user get the pieces into the correct positions.

19.21 *(Maze Generator and Walker)* Develop a multimedia-based maze generator and traverser program based on the maze programs you wrote in Exercise 7.40–Exercise 7.42. Let the user customize the maze by specifying the number of rows and columns and by indicating the level of difficulty. Have an animated mouse walk the maze. Use audio to dramatize the movement of your mouse character.

19.22 *(One-Armed Bandit)* Develop a multimedia simulation of a one-armed bandit. Have three spinning wheels. Place various fruits and symbols on each wheel. Use true random-number generation to simulate the spinning of each wheel and the stopping of each wheel on a symbol.

19.23 *(Horse Race)* Create a Java simulation of a horse race. Have multiple contenders. Use audios for a race announcer. Play the appropriate audios to indicate the correct status of each of the contenders throughout the race. Use audios to announce the final results. You might try to simulate the kinds of horse-racing games that are often played at carnivals. The players get turns at the mouse and have to perform some skill-oriented manipulation with the mouse to advance their horses.

19.24 *(Shuffleboard)* Develop a multimedia-based simulation of the game of shuffleboard. Use appropriate audio and visual effects.

19.25 *(Game of Pool)* Create a multimedia-based simulation of the game of pool. Each player takes turns using the mouse to position a pool stick and to hit the stick against the ball at the appropriate angle to try to get the pool balls to fall into the pockets. Your program should keep score.

19.26 *(Artist)* Design a Java art program that will give an artist a great variety of capabilities to draw, use images, use animations, etc., to create a dynamic multimedia art display.

19.27 *(Fireworks Designer)* Create a Java program that someone might use to create a fireworks display. Create a variety of fireworks demonstrations. Then orchestrate the firing of the fireworks for maximum effect.

19.28 *(Floor Planner)* Develop a Java program that will help someone arrange furniture in his or her home. Add features that enable the person to achieve the best possible arrangement.

19.29 *(Crossword)* Crossword puzzles are among the most popular pastimes. Develop a multimedia-based crossword-puzzle program. Your program should enable the player to place and erase words easily. Tie your program to a large computerized dictionary. Your program also should be able to suggest words on which letters have already been filled in. Provide other features that will make the crossword-puzzle enthusiast's job easier.

19.30 *(15 Puzzle)* Write a multimedia-based Java program that enables the user to play the game of 15. The game is played on is a 4-by-4 board for a total of 16 slots. One of the slots is empty. The other slots are occupied by 15 tiles numbered 1 through 15. Any tile next to the currently empty slot can be moved into that slot by clicking on the tile. Your program should create the board with the tiles out of order. The goal is to arrange the tiles into sequential order, row by row.

19.31 *(Reaction Time/Reaction Precision Tester)* Create a Java program that moves a randomly created shape around the screen. The user moves the mouse to catch and click on the shape. The shape's speed and size can be varied. Keep statistics on how much time the user typically takes to catch a shape of a given size. The user will probably have more difficulty catching faster moving, smaller shapes.

19.32 *(Calendar/Tickler File)* Using both audio and images create a general purpose calendar and "tickler" file. For example, the program should sing "Happy Birthday" when you use it on your birthday. Have the program display images and play audios associated with important events. Also, have the program remind you in advance of these important events. It would be nice, for example, to have the program give you a week's notice so you can pick up an appropriate greeting card for that special person.

19.33 *(Rotating Images)* Create a Java program that lets you rotate an image through some number of degrees (out of a maximum of 360 degrees). The program should let you specify that you want to spin the image continuously. The program should let you adjust the spin speed dynamically.

19.34 *(Coloring Black and White Photographs and Images)* Create a Java program that lets you paint a black and white photograph with color. Provide a color palette for selecting colors. Your program should let you apply different colors to different regions of the image.

19.35 *(Multimedia-Based Simpletron Simulator)* Modify the Simpletron simulator that you developed in the exercises in the previous chapters to include multimedia features. Add computer-like sounds to indicate that the Simpletron is executing instructions. Add a breaking-glass sound when a fatal error occurs. Use flashing lights to indicate which cells of memory or which registers are currently being manipulated. Use other multimedia techniques, as appropriate, to make your Simpletron simulator more valuable to its users as an educational tool.

20

Data Structures

Objectives

- To be able to form linked data structures using references, self-referential classes and recursion.
- To be able to create and manipulate dynamic data structures, such as linked lists, queues, stacks and binary trees.
- To understand various important applications of linked data structures.
- To understand how to create reusable data structures with classes, inheritance and composition.

Much that I bound, I could not free;
Much that I freed returned to me.
Lee Wilson Dodd

'Will you walk a little faster?' said a whiting to a snail,
'There's a porpoise close behind us, and he's treading on my tail.'
Lewis Carroll

There is always room at the top.
Daniel Webster

Push on—keep moving.
Thomas Morton

I think that I shall never see
A poem lovely as a tree.
Joyce Kilmer

SELF-REVIEW EXERCISES

20.1 Fill in the blanks in each of the following statements:

a) A self-_____ class is used to form dynamic data structures that can grow and shrink at execution time.

ANS: referential

b) A(n) _____ is a constrained version of a linked list in which nodes can be inserted and deleted only from the start of the list.

ANS: stack

c) A method that does not alter a linked list, but simply looks at the list to determine whether it is empty is referred to as a(n) _____ method.

ANS: predicate

d) A queue is referred to as a(n) _____ data structure because the first nodes inserted are the first nodes removed.

ANS: first-in, first-out (FIFO)

e) The reference to the next node in a linked list is referred to as a(n) _____.

ANS: link

f) Automatically reclaiming dynamically allocated memory in Java is called _____.

ANS: garbage collection

g) A(n) _____ is a constrained version of a linked list in which nodes can be inserted only at the end of the list and deleted only from the start of the list.

ANS: queue

h) A(n) _____ is a nonlinear, two-dimensional data structure that contains nodes with two or more links.

ANS: tree

i) A stack is referred to as a(n) _____ data structure because the last node inserted is the first node removed.

ANS: last-in, first-out (LIFO)

j) The nodes of a(n) _____ tree contain two link members.

ANS: binary

k) The first node of a tree is the _____ node.

ANS: root

l) Each link in a tree node refers to a(n) _____ or _____ of that node.

ANS: child or subtree

m) A tree node that has no children is called a(n) _____ node.

ANS: leaf

n) The three traversal algorithms we mentioned in the text for binary search trees are _____, _____ and _____.

ANS: inorder, preorder, postorder

20.2 What are the differences between a linked list and a stack?

ANS: It is possible to insert a node anywhere in a linked list and remove a node from anywhere in a linked list. Nodes in a stack may only be inserted at the top of the stack and removed from the top of a stack.

20.3 What are the differences between a stack and a queue?

ANS: A queue is a FIFO data structure that has references to both its head and its tail so that nodes may be inserted at the tail and deleted from the head. A stack is a LIFO data structure that has a single reference to the top of the stack where both insertion and deletion of nodes are performed.

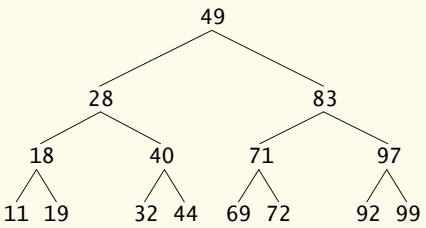


Fig. 20.20 Binary search tree with 15 nodes.

20.4 Perhaps a more appropriate title for this chapter would have been "Reusable Data Structures." Comment on how each of the following entities or concepts contributes to the reusability of data structures:

a) classes

ANS: Classes allow us to instantiate as many data structure objects of a certain type (i.e., class) as we wish.

b) inheritance

ANS: Inheritance enables a subclass to reuse the functionality from a superclass. Public methods of a superclass can be accesed through a subclass to eliminate duplicate logic.

c) composition

ANS: Composition enables a class to reuse code by storing an instance an instance of another class in a field. Public methods of the member class can be called by methods in the composite class.

20.5 Manually provide the inorder, preorder and postorder traversals of the binary search tree of Fig. 20.20.

ANS: The inorder traversal is

```
11 18 19 28 32 40 44 49 69 71 72 83 92 97 99
```

The preorder traversal is

```
49 28 18 11 19 40 32 44 83 71 69 72 97 92 99
```

The postorder traversal is

```
11 19 18 32 44 40 28 69 72 71 92 99 97 83 49
```

EXERCISES

20.6 Write a program that concatenates two linked-list objects of characters. Class `ListConcatenate` should include a method `concatenate` that takes references to both list objects as arguments and concatenates the second list to the first list.

ANS:

```
1  // Exercise 20.6 Solution: ListConcatenate.java
2  // Program concatenates two lists
3  import com.deitel.jhtp5.ch20.*;
4
```

```
 5  public class ListConcatenate {
 6
 7     public static void main( String args[] )
 8     {
 9        // create two linked lists
10        List list1 = new List();
11        List list2 = new List();
12
13        // create objects to store in list1
14        Character a1 = new Character( '5' );
15        Character b1 = new Character( '@' );
16        Character c1 = new Character( 'V' );
17        Character d1 = new Character( '+' );
18
19        // use List insert methods
20        System.out.println( "List 1:" );
21        list1.insertAtFront( a1 );
22        list1.print();
23        list1.insertAtFront( b1 );
24        list1.print();
25        list1.insertAtBack( c1 );
26        list1.print();
27        list1.insertAtBack( d1 );
28        list1.print();
29
30        // create objects to store in list2
31        Character a2 = new Character( 'P' );
32        Character b2 = new Character( 'c' );
33        Character c2 = new Character( 'M' );
34        Character d2 = new Character( '&' );
35
36        // use List insert methods
37        System.out.println( "List 2:" );
38        list2.insertAtFront( a2 );
39        list2.print();
40        list2.insertAtFront( b2 );
41        list2.print();
42        list2.insertAtBack( c2 );
43        list2.print();
44        list2.insertAtBack( d2 );
45        list2.print();
46
47        // concatenate lists using method concatenate
48        concatenate( list1, list2 );
49        System.out.println( "Concatenated list is:" );
50        list1.print();
51     }
52
53     // concatenates two lists and stores the results in the first list
54     public static void concatenate( List one, List two )
55     {
56        while ( !two.isEmpty() )
57           one.insertAtBack( two.removeFromFront() );
58     }
```

```
59
60 } // end class ListConcatenate
```

```
List 1:
The list is: 5

The list is: @ 5

The list is: @ 5 V

The list is: @ 5 V +

List 2:
The list is: P

The list is: c P

The list is: c P M

The list is: c P M &

Concatenated list is:
The list is: @ 5 V + c P M &
```

20.7 Write a program that merges two ordered-list objects of integers into a single ordered list object of integers. Method `merge` of class `ListMerge` should receive references to each of the list objects to be merged and should return a reference to the merged list object.

ANS:

```
1  package com.deitel.jhtp5.ch20;
2
3  // class to represent one node in a list
4  public class ListNode {
5
6     // public access members
7     public Object data;
8     public ListNode nextNode;
9
10    // create a ListNode that refers to object
11    public ListNode( Object object )
12    {
13       this( object, null );
14    }
15
16    // create ListNode that refers to Object and to next ListNode
17    public ListNode( Object object, ListNode node )
18    {
19       data = object;
20       nextNode = node;
21    }
22
```

```
23     public Object getObject()
24     {
25        return data; // return Object in this node
26     }
27
28     public ListNode getNext()
29     {
30        return nextNode; // get next node
31     }
32
33     public void setNext( ListNode next )
34     {
35        nextNode = next;
36     }
37
38  } // end class ListNode
```

ANS:

```
1   // Exercise 20.7 Solution: ListMerge.java
2   // Program inserts and sorts random numbers in two lists and merges them
3   import com.deitel.jhtp5.ch20.*;
4
5   class List3 extends List {
6
7      // constructor takes name
8      public List3( String name )
9      {
10        super( name );
11     }
12
13     // default constructor
14     public List3()
15     {
16        super();
17     }
18
19     // insert number into the sorted list
20     public void insert( Integer number )
21     {
22        // number is the first element in list
23        if ( isEmpty() ) {
24           ListNode newNode = new ListNode( number );
25           firstNode = lastNode = newNode;
26        }
27
28        // list already contains elements
29        else {
30
31           // if number is less than first value
32           if ( ( ( Integer ) firstNode.getData() ).intValue() >
33                                              number.intValue() )
```

```
34
35             insertAtFront( number );
36
37          // if number is greater than last value
38          else if ( ( ( Integer ) lastNode.getData() ).intValue() <
39                                                 number.intValue() )
40
41             insertAtBack( number );
42
43          // search through list for correct placement
44          else {
45             ListNode current = firstNode.getNext();
46             ListNode previous = firstNode;
47             ListNode newNode = new ListNode( number );
48
49             while ( current != lastNode && ( ( Integer )
50                current.getData() ).intValue() < number.intValue() ) {
51
52                previous = current;
53                current = current.getNext();
54             }
55
56             // insert node into list by changing references
57             previous.setNext( newNode );
58             newNode.setNext( current );
59          }
60       }
61
62    }  // end method insert
63
64 }  // end class List3
65
66 // class tests List3 by merging lists
67 public class ListMerge {
68
69    // merges two lists
70    private static List3 merge( List3 one, List3 two )
71    {
72       List3 both = one;
73
74       while ( !two.isEmpty() )
75          both.insert( (Integer) two.removeFromFront() );
76
77       return both;
78    }
79
80    public static void main( String args[] )
81    {
82       List3 first = new List3();
83       List3 second = new List3();
84       Integer newNumber = null;
85
86       // create Integers to store in each list
87       for ( int k = 1; k <= 15; k++ ) {
```

```
88
89             newNumber = new Integer( ( int ) ( Math.random() * 101 ) );
90             first.insert( newNumber );
91
92             newNumber = new Integer( ( int ) ( Math.random() * 101 ) );
93             second.insert( newNumber );
94          }
95
96          System.out.println( "First list is: " );
97          first.print();
98
99          System.out.println( "Second list is: " );
100         second.print();
101
102         List3 merged = merge( first, second);
103         System.out.println( "Merged list is: " );
104         merged.print();
105      }
106
107   }  // end class ListMerge
```

```
First list is:
The list is: 1 4 9 24 34 41 50 56 60 75 76 76 91 91 92

Second list is:
The list is: 2 17 20 21 27 38 38 66 66 66 74 75 87 93 98

Merged list is:
The list is: 1 2 4 9 17 20 21 24 27 34 38 38 41 50 56 60 66 66 66 74 75 75 76
76 87 91 91 92 93 98
```

20.8 Write a program that inserts 25 random integers from 0 to 100 in order into a linked list object. The program should calculate the sum of the elements and the floating-point average of the elements.

ANS:

```
1   // Exercise 20.8 Solution: ListTest.java
2   // Program inserts and sorts random numbers in a list,
3   // prints the sum, and displays the average.
4   import com.deitel.jhtp5.ch20.*;
5
6   class List2 extends List {
7
8      // constructor takes name
9      public List2( String name )
10     {
11        super( name );
12     }
13
14     // default constructor
15     public List2()
16     {
```

```
17          super();
18       }
19
20       // insert number into the sorted list
21       public void insert( Integer number )
22       {
23          // number is the first element in list
24          if ( isEmpty() ) {
25             ListNode newNode = new ListNode( number );
26             firstNode = lastNode = newNode;
27          }
28
29          // list already contains elements
30          else {
31
32             // if number is less than first value
33             if ( ( ( Integer ) firstNode.getObject() ).intValue() >
34                                              number.intValue() )
35
36                insertAtFront( number );
37
38             // if number is greater than last value
39             else if ( ( ( Integer ) lastNode.getObject() ).intValue() <
40                                                 number.intValue() )
41
42                insertAtBack( number );
43
44             // search through list for correct placement
45             else {
46                ListNode current = firstNode.getNext();
47                ListNode previous = firstNode;
48                ListNode newNode = new ListNode( number );
49
50                while ( current != lastNode && ( ( Integer )
51                   current.getObject() ).intValue() < number.intValue() ) {
52
53                   previous = current;
54                   current = current.getNext();
55                }
56
57                // insert node into list by changing references
58                previous.setNext( newNode );
59                newNode.setNext( current );
60             }
61          }
62
63       }  // end method insert
64
65       // calculates and returns sum of every value in list
66       public int add()
67       {
68          int sum = 0;
69          ListNode current = firstNode;
70
```

```
71          // cycle through list adding values to total
72          while ( current != null ) {
73             sum += ( ( Integer ) current.getObject() ).intValue();
74             current = current.getNext();
75          }
76
77          return sum;
78       }
79
80    }  // end class List2
81
82    // class tests List2 by inserting random numbers
83    public class ListTest {
84
85       public static void main( String args[] )
86       {
87          List2 list = new List2();
88          Integer newNumber = null;
89
90          // create Integers to store in list
91          for ( int k = 1; k <= 25; k++ ) {
92             newNumber = new Integer( ( int ) ( Math.random() * 101 ) );
93             list.insert( newNumber );
94          }
95
96          list.print();
97
98          int sum = list.add();
99          System.out.println( "Sum is: " + sum + "\nAverage: " +
100            ( ( float ) sum / 25.0f ) );
101      }
102
103   }  // end class ListTest
```

```
The list is: 2 6 8 8 9 11 24 24 26 34 34 40 45 49 52 55 69 71 73 75 82 85 87
92 93

Sum is: 1154
Average: 46.16
```

20.9 Write a program that creates a linked list object of 10 characters, then creates a second list object containing a copy of the first list, but in reverse order.

ANS:

```
1  // Exercise 20.9 Solution: ListReverse.java
2  // Program creates a list, then creates a reverse of the list
3  import com.deitel.jhtp5.ch20.*;
4
5  public class ListReverse {
6
7     public static void main( String args[] )
8     {
```

```
 9         // create two linked lists
10         List list1 = new List();
11         List list2 = new List();
12
13         // use List insert methods
14         list1.insertAtFront( new Character( '5' ) );
15         list1.insertAtFront( new Character( '@' ) );
16         list1.insertAtBack( new Character( 'V' ) );
17         list1.insertAtBack( new Character( '+' ) );
18         list1.insertAtFront( new Character( 'P' ) );
19         list1.insertAtFront( new Character( 'c' ) );
20         list1.insertAtBack( new Character( 'M' ) );
21         list1.insertAtBack( new Character( '&' ) );
22         list1.insertAtFront( new Character( 'y' ) );
23         list1.insertAtBack( new Character( 'X' ) );
24
25         System.out.println( "List: " );
26         list1.print();
27
28         list2 = reverse( list1 ); // reverse lists
29         System.out.println( "Reversed list is:" );
30         list2.print();
31      }
32
33      // reverses a list and returns it to the caller
34      public static List reverse( List one )
35      {
36         List reversed = new List();
37
38         while ( !one.isEmpty() )
39            reversed.insertAtFront( one.removeFromFront() );
40
41         return reversed;
42      }
43
44   }  // end class ListReverse
```

```
List:
The list is: y c P @ 5 V + M & X

Reversed list is:
The list is: X & M + V 5 @ P c y
```

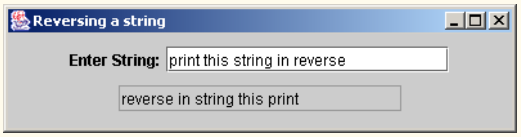
20.10 Write a program that inputs a line of text and uses a stack object to print the words of the line in reverse order.

ANS:

```
1  // Exercise 20.10 Solution: StackTest.java
2  // Program prints the words of a line in reverse
3  import java.awt.*;
4  import java.util.*;
```

```
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  import com.deitel.jhtp5.ch20.*;
9
10 public class StackTest extends JFrame {
11
12    private JTextField inputField, outputField;
13    private JLabel prompt;
14    private JPanel panel;
15
16    public StackTest()
17    {
18       super( "Reversing a string" );
19
20       // create stack
21       final StackComposition stack = new StackComposition();
22
23       // create GUI components
24       prompt = new JLabel( "Enter String:" );
25       inputField = new JTextField( 20 );
26
27       inputField.addActionListener(
28
29          new ActionListener() { // anonymous inner class
30
31             public void actionPerformed( ActionEvent event )
32             {
33                // take each word from tokenizer and push on stack
34                String text = inputField.getText();
35                StringTokenizer tokenizer = new StringTokenizer( text );
36                StringBuffer buffer = new StringBuffer( text.length() );
37
38                while ( tokenizer.hasMoreTokens() )
39                   stack.push( tokenizer.nextToken() );
40
41                // build reverse string by popping words from stack.
42                while ( !stack.isEmpty() ) {
43                   Object removedObject = stack.pop();
44                   buffer.append( removedObject.toString() + " " );
45                }
46
47                outputField.setText( buffer.toString() );
48             }
49
50          }  // end anonymous inner class
51       );
52
53       outputField = new JTextField( 20 );
54       outputField.setEditable( false );
55
56       // set up layout and add components
57       Container container = getContentPane();
58       container.setLayout( new FlowLayout() );
```

```
59          JPanel panel = new JPanel();
60          panel.add( prompt );
61          panel.add( inputField );
62          container.add( panel );
63          container.add( outputField );
64
65          setSize( 400, 100 );
66          setVisible( true );
67       }
68
69       public static void main( String args[] )
70       {
71          StackTest application = new StackTest();
72          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
73       }
74
75    }  // end class StackTest
```



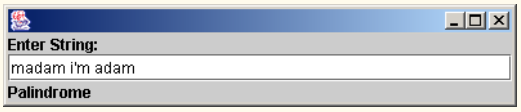
20.11 Write a program that uses a stack to determine whether a string is a palindrome (i.e., the string is spelled identically backward and forward). The program should ignore spaces and punctuation.

ANS:

```
1  // Exercise 20.11 Solution: StackTest2.java
2  // Program tests for a palindrome.
3  import java.awt.*;
4  import java.util.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7  import com.deitel.jhtp5.ch20.*;
8
9  public class StackTest2 extends JFrame {
10    private JLabel output;
11
12    public StackTest2()
13    {
14       // create stack
15       final StackComposition stack = new StackComposition();
16
17       // create GUI components
18       final JLabel prompt = new JLabel( "Enter String:" );
19       final JTextField input = new JTextField( 20 );
20       input.addActionListener(
21
22          new ActionListener() { // anonymous inner class
23
24             public void actionPerformed( ActionEvent event )
25             {
```

```
26                String text = input.getText();
27                char letter;
28
29                // cycle through input one char at a time to
30                // create stack of all relevant characters
31                for ( int i = 0; i < text.length(); i++ ) {
32                   letter = text.charAt( i );
33
34                   if ( Character.isLetterOrDigit( letter ) )
35                      stack.push( new Character( letter ) );
36                }
37
38                Object removedObject = null;
39                boolean flag = false;
40
41                // test for palindrome
42                try {
43
44                   for ( int count = 0; count < text.length()
45                      && !stack.isEmpty(); count++ ) {
46
47                      letter = text.charAt( count );
48
49                      // ignore spaces and punctuation
50                      if ( !Character.isLetterOrDigit( letter ) )
51                         continue;
52
53                      removedObject = stack.pop();
54
55                      // not palindrome
56                      if ( letter != ( ( Character )
57                            removedObject ).charValue() ) {
58                         flag = true;
59                         break;
60                      }
61                   }
62
63                   // palindrome
64                   if ( flag == false )
65                      output.setText( "Palindrome" );
66
67                   // not palindrome
68                   else
69                      output.setText( "Not a Palindrome" );
70                }
71
72                // catch operations performed on empty list
73                catch ( EmptyListException exception ) {
74                   System.err.println( "\n" + exception.toString() );
75                }
76
77             }  // end method actionPerformed
78
79          }  // end anonymous inner class
80       );
```

```
81
82          output = new JLabel( " " );
83
84          // add components to GUI
85          Container container = getContentPane();
86          container.setLayout( new BorderLayout() );
87          container.add( prompt, BorderLayout.NORTH );
88          container.add( input, BorderLayout.CENTER );
89          container.add( output, BorderLayout.SOUTH );
90
91          setSize( 400, 80 );
92          setVisible( true );
93       }
94
95       public static void main( String args[] )
96       {
97          StackTest2 application = new StackTest2();
98          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
99       }
100
101 }  // end class StackTest2
```



20.12 Stacks are used by compilers to help in the process of evaluating expressions and generating machine language code. In this and the next exercise, we investigate how compilers evaluate arithmetic expressions consisting only of constants, operators and parentheses.

Humans generally write expressions like 3 + 4 and 7 / 9 in which the operator (+ or / here) is written between its operands—this is called *infix notation*. Computers "prefer" *postfix notation*, in which the operator is written to the right of its two operands. The preceding infix expressions would appear in postfix notation as 3 4 + and 7 9 /, respectively.

To evaluate a complex infix expression, a compiler would first convert the expression to postfix notation and evaluate the postfix version of the expression. Each of these algorithms requires only a single left-to-right pass of the expression. Each algorithm uses a stack object in support of its operation, and in each algorithm, the stack is used for a different purpose.

In this exercise, you will write a Java version of the infix-to-postfix conversion algorithm. In the next exercise, you will write a Java version of the postfix expression evaluation algorithm. In a later exercise, you will discover that code you write in this exercise can help you implement a complete working compiler.

Write class `InfixToPostfixConverter` to convert an ordinary infix arithmetic expression (assume a valid expression is entered) with single-digit integers such as

```
(6 + 2) * 5 - 8 / 4
```

to a postfix expression. The postfix version of the preceding infix expression is (note that no parentheses are needed)

```
6 2 + 5 * 8 4 / -
```

The program should read the expression into `StringBuffer infix` and use one of the stack classes implemented in this chapter to help create the postfix expression in `StringBuffer postfix`. The algorithm for creating a postfix expression is as follows:

a) Push a left parenthesis `'('` on the stack.
b) Append a right parenthesis `')'` to the end of `infix`.
c) While the stack is not empty, read `infix` from left to right and do the following:
    If the current character in `infix` is a digit, append it to `postfix`.
    If the current character in `infix` is a left parenthesis, push it onto the stack.
    If the current character in `infix` is an operator:
        Pop operators (if there are any) at the top of the stack while they have equal or higher precedence than the current operator, and append the popped operators to `postfix`.
        Push the current character in `infix` onto the stack.
    If the current character in `infix` is a right parenthesis:
        Pop operators from the top of the stack and append them to `postfix` until a left parenthesis is at the top of the stack.
        Pop (and discard) the left parenthesis from the stack.

The following arithmetic operations are allowed in an expression:

`+` addition
`-` subtraction
`*` multiplication
`/` division
`^` exponentiation
`%` remainder

The stack should be maintained with stack nodes that each contain an instance variable and a reference to the next stack node. Some of the methods you may want to provide are as follows:

a) Method `convertToPostfix`, which converts the infix expression to postfix notation.
b) Method `isOperator`, which determines whether `c` is an operator.
c) Method `precedence`, which determines if the precedence of `operator1` (from the infix expression) is less than, equal to or greater than the precedence of `operator2` (from the stack). The method returns `true` if `operator1` has lower precedence than `operator2`. Otherwise, `false` is returned.
d) Method `stackTop` (this should be added to the stack class), which returns the top value of the stack without popping the stack.

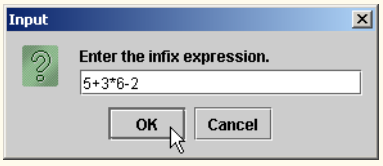
ANS:

```
1  // Exercise 20.12 Solution:InfixToPostfixConverter.java
2  // Infix to postfix conversion
3  import java.awt.*;
4  import javax.swing.*;
5  import com.deitel.jhtp5.ch20.*;
6
7  public class InfixToPostfixConverter {
8
9     public static void main( String args[] )
10    {
11       StringBuffer infix = new StringBuffer(
12          JOptionPane.showInputDialog( "Enter the infix expression.\n" ) );
13
14       System.out.println( "The original infix expression is:\n" +
15          infix + "\n" );
```

```
16
17          // change from infix notation into postfix notation
18          StringBuffer postfix = convertToPostfix( infix );
19
20          System.out.println( "The expression in postfix notation is:\n" +
21             postfix + "\n" );
22
23          System.exit( 0 );
24
25       } // end main
26
27       // take out the infix and change it into postfix
28       private static StringBuffer convertToPostfix( StringBuffer infix )
29       {
30          CharacterStack charStack = new CharacterStack();
31          StringBuffer temporary = new StringBuffer( "" );
32
33          // push a left paren onto the stack and add a right paren to infix
34          charStack.pushChar( '(' );
35          charStack.print();
36          infix.append( ')' );
37
38          // convert the infix expression to postfix
39          for ( int infixCount = 0; !charStack.isEmpty(); ++infixCount ) {
40
41             if ( Character.isDigit( infix.charAt( infixCount ) ) )
42                temporary.append( infix.charAt( infixCount ) + " " );
43
44             else if ( infix.charAt( infixCount ) == '(' )
45                charStack.pushChar( '(' );
46
47             else if ( isOperator( infix.charAt( infixCount ) ) ) {
48
49                while (  isOperator( charStack.stackTop() ) &&
50                   precedence( charStack.stackTop(),
51                      infix.charAt( infixCount ) ) )
52
53                   temporary.append( charStack.popChar() + " " );
54
55                charStack.pushChar( infix.charAt( infixCount ) );
56
57             } // end else if
58
59             else if ( infix.charAt( infixCount ) == ')' ) {
60
61                while ( charStack.stackTop() != '(' )
62                   temporary.append( charStack.popChar() + " " );
63
64                charStack.popChar();
65
66             } // end else if
67
68          } // end for loop
69
70          return temporary;
```

```
71
72       } // end function convert
73
74       // check if c is an operator
75       private static boolean isOperator( char c )
76       {
77          if ( c == '+' || c == '-' || c == '*' || c == '/' || c == '^' )
78             return true;
79
80          else
81             return false;
82
83       } // end function isOperator
84
85       // ensure proper order of operations
86       private static boolean precedence( char operator1, char operator2 )
87       {
88          if ( operator1 == '^' )
89             return true;
90
91          else if ( operator2 == '^' )
92             return false;
93
94          else if ( operator1 == '*' || operator1 == '/' )
95             return true;
96
97          else if ( operator1 == '+' || operator1 == '-' )
98
99             if ( operator2 == '*' || operator2 == '/' )
100               return false;
101
102            else
103               return true;
104
105         return false;
106
107      } // end function precedence
108
109   } // end class InfixToPostfixConverter
110
111   class CharacterStack extends StackComposition {
112
113      public char stackTop() {
114         char temp = popChar();
115         pushChar( temp );
116
117         return temp;
118      }
119
120      public char popChar() {
121         return ( ( Character )super.pop() ).charValue();
122      }
123
124      public void pushChar( char c ) {
125         super.push( new Character( c ) );
```

```
126     }
127
128  } // end class CharacterStack
```




```
The original infix expression is:
5+3*6-2

The stack is: (

The expression in postfix notation is:
5 3 6 * + 2 -
```

20.13 Write class `PostfixEvaluator`, which evaluates a postfix expression such as

```
6 2 + 5 * 8 4 / -
```

The program should read a postfix expression consisting of digits and operators into a `StringBuffer`. Using modified versions of the stack methods implemented earlier in this chapter, the program should scan the expression and evaluate it (assume it is valid) . The algorithm is as follows:

a) Append a right parenthesis (`')'`) to the end of the postfix expression. When the right-parenthesis character is encountered, no further processing is necessary.

b) When the right-parenthesis character has not been encountered, read the expression from left to right.

   If the current character is a digit do the following:

   Push its integer value on the stack (the integer value of a digit character is its value in the computer's character set minus the value of `'0'` in Unicode).

   Otherwise, if the current character is an *operator*:

   Pop the two top elements of the stack into variables `x` and `y`.

   Calculate `y` *operator* `x`.

   Push the result of the calculation onto the stack.

c) When the right parenthesis is encountered in the expression, pop the top value of the stack. This is the result of the postfix expression.

[*Note*: In b) above (based on the sample expression at the beginning of this exercises), if the operator is `'/'`, the top of the stack is `2` and the next element in the stack is `8`, then pop `2` into `x`, pop `8` into `y`, evaluate `8 / 2` and push the result, `4`, back on the stack. This note also applies to operator `'-'`.] The arithmetic operations allowed in an expression are:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`^`	exponentiation
`%`	remainder

The stack should be maintained with one of the stack classes introduced in this chapter. You may want to provide the following methods:

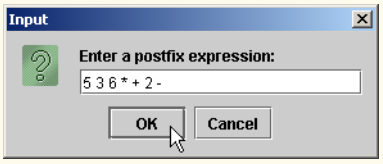
a) Method `evaluatePostfixExpression`, which evaluates the postfix expression.
b) Method `calculate`, which evaluates the expression `op1 operator op2`.
c) Method `push`, which pushes a value onto the stack.
d) Method `pop`, which pops a value off the stack.
e) Method `isEmpty`, which determines whether the stack is empty.
f) Method `printStack`, which prints the stack.

ANS:

```
1  // Exercise 20.13 Solution:PostfixEvaluator.java
2  // Using a stack to evaluate an expression in postfix notation
3  import javax.swing.*;
4  import com.deitel.jhtp5.ch20.*;
5
6  public class PostfixEvaluator {
7
8     public static void main( String args[] )
9     {
10       StringBuffer expression = new StringBuffer(
11          JOptionPane.showInputDialog( "Enter a postfix expression:" ) );
12
13       int answer = evaluatePostfixExpression( expression );
14       System.out.println( "The value of the expression is: " +
15          answer + "\n" );
16
17       System.exit( 0 );
18
19    } // end main
20
21    // evaluate the postfix notation
22    private static int evaluatePostfixExpression( StringBuffer expr )
23    {
24       int i, popVal1, popVal2, pushVal;
25       IntegerStack intStack = new IntegerStack();
26       char c;
27
28       expr.append( ")" );
29
30       // until it reaches ")"
31       for ( i = 0; expr.charAt( i ) != ')'; ++i )
32
33          if ( Character.isDigit( expr.charAt( i ) ) ) {
34
35             pushVal = expr.charAt( i ) - '0';
36             intStack.pushInt( pushVal );
37             intStack.print();
38          }
39
40          else if ( !Character.isWhitespace( expr.charAt( i ) ) ) {
41
42             popVal2 = intStack.popInt();
43             intStack.print();
```

```
44              popVal1 = intStack.popInt();
45              intStack.print();
46              pushVal = calculate( popVal1, popVal2, expr.charAt( i ) );
47              intStack.pushInt( pushVal );
48              intStack.print();
49           }
50
51        return intStack.popInt();
52
53     } // end PostfixEvaluator
54
55     // do the calculation
56     private static int calculate( int op1, int op2, char oper )
57     {
58        switch( oper ) {
59
60           case '+':
61              return op1 + op2;
62
63           case '-':
64              return op1 - op2;
65
66           case '*':
67              return op1 * op2;
68
69           case '/':
70              return op1 / op2;
71
72           case '^':   // exponentiation
73              return ( int )Math.pow( op1, op2 );
74        }
75
76        return 0;
77
78     } // end function calculate
79
80  } // end class PostfixEvaluator
81
82  class IntegerStack extends StackInheritance {
83
84     public int stackTop() {
85        int temp = popInt();
86        pushInt( temp );
87
88        return temp;
89     }
90
91     public int popInt()
92     {
93        return ( ( Integer )super.pop() ).intValue();
94     }
95
96     public void pushInt( int c )
97     {
```

```
98          super.push( new Integer( c ) );
99       }
100
101   } // end class IntegerStack
```




```
The stack is: 5

The stack is: 3 5

The stack is: 6 3 5

The stack is: 3 5

The stack is: 5

The stack is: 18 5

The stack is: 5

Empty stack
The stack is: 23

The stack is: 2 23

The stack is: 23

Empty stack
The stack is: 21

The value of the expression is: 21
```

20.14 Modify the postfix evaluator program of Exercise 20.13 so that it can process integer operands larger than 9.

20.15 *(Supermarket Simulation)* Write a program that simulates a checkout line at a supermarket. The line is a queue object. Customers (i.e., customer objects) arrive in random integer intervals of from 1 to 4 minutes. Also, each customer is serviced in random integer intervals of from 1 to 4 minutes. Obviously, the rates need to be balanced. If the average arrival rate is larger than the average service rate, the queue will grow infinitely. Even with "balanced" rates, randomness can still cause long lines. Run the supermarket simulation for a 12-hour day (720 minutes), using the following algorithm:

a) Choose a random integer between 1 and 4 to determine the minute at which the first customer arrives.

b) At the first customer's arrival time, do the following:
Determine customer's service time (random integer from 1 to 4).
Begin servicing the customer.
Schedule arrival time of next customer (random integer 1 to 4 added to the current time).
c) For each minute of the day, consider the following:
If the next customer arrives, proceed as follows:
Say so.
Enqueue the customer.
Schedule the arrival time of the next customer.
If service was completed for the last customer, do the following:
Say so.
Dequeue next customer to be serviced.
Determine customer's service completion time (random integer from 1 to 4 added to the current time).

Now run your simulation for 720 minutes and answer each of the following:
a) What is the maximum number of customers in the queue at any time?
b) What is the longest wait any one customer experiences?
c) What happens if the arrival interval is changed from 1 to 4 minutes to 1 to 3 minutes?

20.16 Modify Fig. 20.17 and Fig. 20.18 to allow the binary tree to contain duplicates.
ANS:

```
1  package com.deitel.jhtp5.ch20;
2
3  // class TreeNode definition
4  public class TreeNode {
5
6     // public access members
7     public TreeNode leftNode;
8     public int data;
9     public TreeNode rightNode;
10
11    // initialize data and make this a leaf node
12    public TreeNode( int nodeData )
13    {
14       data = nodeData;
15       leftNode = rightNode = null;  // node has no children
16    }
17
18    // locate insertion point and insert new node
19    public synchronized void insert( int insertValue )
20    {
21       // insert in left subtree
22       if ( insertValue < data ) {
23
24          // insert new TreeNode
25          if ( leftNode == null )
26             leftNode = new TreeNode( insertValue );
27
28          else // continue traversing left subtree
29             leftNode.insert( insertValue );
30       }
```

```
31
32          // insert in right subtree
33          else {
34
35             // insert new TreeNode
36             if ( rightNode == null )
37                rightNode = new TreeNode( insertValue );
38
39             else // continue traversing right subtree
40                rightNode.insert( insertValue );
41          }
42
43       } // end method insert
44
45       // get right child
46       public synchronized TreeNode getRight()
47       {
48          return rightNode;
49       }
50
51       // get left child
52       public synchronized TreeNode getLeft()
53       {
54          return leftNode;
55       }
56
57       // return the data
58       public synchronized Object getData()
59       {
60          return new Integer( data );
61       }
62
63    }  // end class TreeNode
```

ANS:

```
1  // Exercise 20.16 Solution: TreeTest.java
2  // This program tests the Tree class. The solution to 20.16 is mostly
3  // found in the code of TreeNode, in package com.deitel.jhtp5.ch20
4  import java.util.*;
5  import com.deitel.jhtp5.ch20.*;
6
7  public class TreeTest {
8
9     public static void main( String args[] )
10    {
11       Tree tree = new Tree();
12       int intVal;
13
14       System.out.println( "Inserting the following values: " );
15
```

```
16         // randomly generate numbers and insert in the tree
17         for ( int i = 1; i <= 10; i++ ) {
18            intVal = ( int ) ( Math.random() * 100 );
19            System.out.print( intVal + " " );
20            tree.insertNode( intVal );
21         }
22
23         // print each of the traversals
24         System.out.println ( "\n\nPreorder traversal" );
25         tree.preorderTraversal();
26
27         System.out.println ( "\n\nInorder traversal" );
28         tree.inorderTraversal();
29
30         System.out.println ( "\n\nPostorder traversal" );
31         tree.postorderTraversal();
32         System.out.println();
33      }
34
35   }  // end class Tree
```

```
Inserting the following values:
42 11 97 76 83 85 83 19 44 64

Preorder traversal
42 11 19 97 76 44 64 83 85 83

Inorder traversal
11 19 42 44 64 76 83 83 85 97

Postorder traversal
19 11 64 44 83 85 83 76 97 42
```

20.17 Write a program based on the program of Fig. 20.17 and Fig. 20.18 that inputs a line of text, tokenizes the sentence into separate words (you might want to use the `StreamTokenizer` class from the `java.io` package), inserts the words in a binary search tree and prints the inorder, preorder and post-order traversals of the tree.

ANS:

```
1  // Exercise 20.17 Solution: TreeNode2.java
2  // Class TreeNode2 definition.
3
4  public class TreeNode2 {
5
6     // public access members
7     public TreeNode2 leftNode;
8     public String data;
9     public TreeNode2 rightNode;
10
11    // initialize data and make this a leaf node
12    public TreeNode2( String value )
13    {
```

```
14          data = value;
15          leftNode = rightNode = null;  // node has no children
16       }
17
18       // get right child
19       public synchronized TreeNode2 getRight()
20       {
21          return rightNode;
22       }
23
24       // get left child
25       public synchronized TreeNode2 getLeft()
26       {
27          return leftNode;
28       }
29
30       // return the data
31       public synchronized Object getData()
32       {
33          return data;
34       }
35
36       // insert node
37       public void insert( String string )
38       {
39          // insert in left subtree
40          if ( string.compareTo( data ) < 0 ) {
41
42             // insert new TreeNode2
43             if ( leftNode == null )
44                leftNode = new TreeNode2( string );
45
46             else // continue traversing left subtree
47                leftNode.insert( string );
48          }
49
50          // insert in right subtree
51          else {
52
53             // insert new TreeNode2
54             if ( rightNode == null )
55                rightNode = new TreeNode2( string );
56
57             else // continue traversing right subtree
58                rightNode.insert( string );
59          }
60       }
61
62    }  // end class TreeNode2
```

ANS:

```
// Exercise 20.17 Solution: Tree2.java
// Class Tree2 definition.
import com.deitel.jhtp5.ch20.*;

public class Tree2 {
   private TreeNode2 root;

   public Tree2()
   {
      root = null;
   }

   // begin preorder traversal
   public synchronized void preorderTraversal()
   {
      preorderHelper( root );
   }

   // recursive method to perform preorder traversal
   private void preorderHelper( TreeNode2 node )
   {
      if ( node == null )
         return;

      System.out.print( node.data + " " ); // output node data
      preorderHelper( node.leftNode );     // traverse left subtree
      preorderHelper( node.rightNode );    // traverse right subtree
   }

   // begin inorder traversal
   public synchronized void inorderTraversal()
   {
      inorderHelper( root );
   }

   // recursive method to perform inorder traversal
   private void inorderHelper( TreeNode2 node )
   {
      if ( node == null )
         return;

      inorderHelper( node.leftNode );      // traverse left subtree
      System.out.print( node.data + " " ); // output node data
      inorderHelper( node.rightNode );     // traverse right subtree
   }

   // begin postorder traversal
   public synchronized void postorderTraversal()
   {
      postorderHelper( root );
   }

```

```
53    // recursive method to perform postorder traversal
54    private void postorderHelper( TreeNode2 node )
55    {
56       if ( node == null )
57          return;
58
59       postorderHelper( node.leftNode );    // traverse left subtree
60       postorderHelper( node.rightNode );   // traverse right subtree
61       System.out.print( node.data + " " ); // output node data
62    }
63
64    public void insertNode( String string )
65    {
66       // tree is empty
67       if ( root == null )
68          root = new TreeNode2( string );
69
70       else // call TreeNode2 method insert on root
71          root.insert( string );
72    }
73
74 }  // end class Tree2
```

ANS:

```
1  // Exercise 20.17 Solution: Tree2Test.java
2  // Program tests the Tree2 class.
3  import java.awt.*;
4  import java.util.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class Tree2Test extends JFrame {
9     private JLabel prompt;
10    private JTextField inputField;
11
12    public Tree2Test()
13    {
14       super( "Tokenizer" );
15       prompt = new JLabel( "Enter String:" );
16       inputField = new JTextField( 25 );
17       inputField.addActionListener(
18
19          new ActionListener() {
20
21             public void actionPerformed( ActionEvent event )
22             {
23                Tree2 tree = new Tree2();
24
25                StringTokenizer tokens =
26                   new StringTokenizer( inputField.getText() );
27
```

```
                while ( tokens.hasMoreTokens() )
                   tree.insertNode( tokens.nextToken() );

                System.out.println( "\n\nPreorder traversal" );
                tree.preorderTraversal();

                System.out.println( "\n\nInorder traversal" );
                tree.inorderTraversal();

                System.out.println( "\n\nPostorder traversal" );
                tree.postorderTraversal();
             }
          }
       );

       Container container = getContentPane();
       container.add( prompt, BorderLayout.NORTH );
       container.add( inputField, BorderLayout.SOUTH );

       setSize( 400, 100 );
       setVisible( true );
    }

    public static void main( String args[] )
    {
       Tree2Test application = new Tree2Test();
       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

} // end class Tree2Test
```


```
Preorder traversal
I have been a inserted into tree

Inorder traversal
I a been have inserted into tree

Postorder traversal
a been tree into inserted have I
```

20.18 In this chapter, we saw that duplicate elimination is straightforward when creating a binary search tree. Describe how you would perform duplicate elimination when using only a one-dimensional array. Compare the performance of array-based duplicate elimination with the performance of binary-search-tree-based duplicate elimination.

20.19 Write a method `depth` that receives a binary tree and determines how many levels it has.

ANS:

```
1  // Fig. 20.17: Tree.java
2  // Definition of class TreeNode and class Tree.
3
4  class TreeNode {
5
6     // package access members
7     TreeNode leftNode;
8     int data;
9     TreeNode rightNode;
10
11    // initialize data and make this a leaf node
12    public TreeNode( int nodeData )
13    {
14       data = nodeData;
15       leftNode = rightNode = null;  // node has no children
16    }
17
18    // locate insertion point and insert new node; ignore duplicate values
19    public synchronized void insert( int insertValue )
20    {
21       // insert in left subtree
22       if ( insertValue < data ) {
23
24          // insert new TreeNode
25          if ( leftNode == null )
26             leftNode = new TreeNode( insertValue );
27
28          else // continue traversing left subtree
29             leftNode.insert( insertValue );
30       }
31
32       // insert in right subtree
33       else if ( insertValue > data ) {
34
35          // insert new TreeNode
36          if ( rightNode == null )
37             rightNode = new TreeNode( insertValue );
38
39          else // continue traversing right subtree
40             rightNode.insert( insertValue );
41       }
42
43    } // end method insert
44
45    public TreeNode getLeft()
46    {
47       return leftNode;
48    }
49
50    public TreeNode getRight()
51    {
52       return rightNode;
53    }
```

```
54
55 } // end class TreeNode
56
57 // class Tree definition
58 public class Tree {
59    private TreeNode root;
60
61    public TreeNode getRoot()
62    {
63       return root;
64    }
65
66    // construct an empty Tree of integers
67    public Tree()
68    {
69       root = null;
70    }
71
72    // insert a new node in the binary search tree
73    public synchronized void insertNode( int insertValue )
74    {
75       if ( root == null )
76          root = new TreeNode( insertValue ); // create the root node here
77
78       else
79          root.insert( insertValue ); // call the insert method
80    }
81
82    // begin preorder traversal
83    public synchronized void preorderTraversal()
84    {
85       preorderHelper( root );
86    }
87
88    // recursive method to perform preorder traversal
89    private void preorderHelper( TreeNode node )
90    {
91       if ( node == null )
92          return;
93
94       System.out.print( node.data + " " ); // output node data
95       preorderHelper( node.leftNode );     // traverse left subtree
96       preorderHelper( node.rightNode );    // traverse right subtree
97    }
98
99    // begin inorder traversal
100   public synchronized void inorderTraversal()
101   {
102      inorderHelper( root );
103   }
104
105   // recursive method to perform inorder traversal
106   private void inorderHelper( TreeNode node )
107   {
```

```
108       if ( node == null )
109          return;
110
111       inorderHelper( node.leftNode );       // traverse left subtree
112       System.out.print( node.data + " " ); // output node data
113       inorderHelper( node.rightNode );      // traverse right subtree
114    }
115
116    // begin postorder traversal
117    public synchronized void postorderTraversal()
118    {
119       postorderHelper( root );
120    }
121
122    // recursive method to perform postorder traversal
123    private void postorderHelper( TreeNode node )
124    {
125       if ( node == null )
126          return;
127
128       postorderHelper( node.leftNode );    // traverse left subtree
129       postorderHelper( node.rightNode );   // traverse right subtree
130       System.out.print( node.data + " " ); // output node data
131    }
132
133 } // end class Tree
```

```
Inserting the following values:
66 79 32 81 39 90 71 16 80 66

Preorder traversal
66 32 16 39 79 71 81 80 90

Inorder traversal
16 32 39 66 71 79 80 81 90

Postorder traversal
16 39 32 71 80 90 81 79 66

Tree has a depth of: 3
```

20.20 (*Recursively Print a List Backwards*) Write a method `printListBackwards` that recursively outputs the items in a linked list object in reverse order. Write a test program that creates a sorted list of integers and prints the list in reverse order.

ANS:

```
1  // Exercise 20.20 Solution: List3.java
2  // Class List3 definition.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class List3 extends List {
```

```
 6
 7     public List3( String string )
 8     {
 9        super( string );
10     }
11
12     public List3()
13     {
14        this( "list" );
15     }
16
17     // insert Integer
18     public void insert( Integer number )
19     {
20        // empty list
21        if ( isEmpty() ) {
22           ListNode newNode = new ListNode( number );
23           firstNode = lastNode = newNode;
24        }
25
26        // insert into list
27        else {
28
29           // insert at front
30           if ( ( ( Integer ) firstNode.getObject() ).intValue() >
31                                                 number.intValue() )
32
33              insertAtFront( number );
34
35           // insert at back
36           else if ( ( ( Integer ) lastNode.getObject() ).intValue() <
37                                                    number.intValue() )
38
39              insertAtBack( number );
40
41           // insert in middle of list
42           else {
43              ListNode current = firstNode.getNext();
44              ListNode previous = firstNode;
45              ListNode newNode = new ListNode( number );
46
47              // locate proper place to insert
48              while ( current != lastNode && ( ( Integer )
49                 current.getObject() ).intValue() < number.intValue() ) {
50
51                 previous = current;
52                 current = current.getNext();
53              }
54
55              previous.setNext( newNode );
56              newNode.setNext( current );
57           }
58        }
59
60     }  // end method insert
```

```
61
62    // print list backwards
63    public void printListBackwards()
64    {
65       System.out.print( "\nReverse ordered list: " );
66       reverse( firstNode );
67       System.out.println();
68    }
69
70    private void reverse( ListNode currentNode )
71    {
72       if ( currentNode == null )
73          return;
74
75       else
76          reverse( currentNode.getNext() );
77
78       System.out.print( ( ( Integer )
79          currentNode.getObject() ).intValue() + " " );
80    }
81
82 }  // end class List3
```

ANS:

```
1  // Exercise 20.20 Solution: List2Test.java
2  // Program recursively prints a list of random numbers backwards.
3
4  public class List2Test {
5
6     public static void main( String args[] )
7     {
8        List3 list = new List3();
9        Integer number = null;
10
11       // create objects to store in the List
12       for ( int i = 1; i <= 25; i++ ) {
13          number = new Integer( ( int ) ( Math.random() * 101 ) );
14          list.insert( number );
15       }
16
17       list.print();
18       list.printListBackwards();
19    }
20
21 }  // end class List2Test
```

```
The list is: 0 6 11 16 19 32 35 36 36 41 44 48 48 49 52 58 59 65 65 72 73 79
79 84 95


Reverse ordered list: 95 84 79 79 73 72 65 65 59 58 52 49 48 48 44 41 36 36 35
32 19 16 11 6 0
```

20.21 (*Recursively Search a List*) Write a method `searchList` that recursively searches a linked list object for a specified value. Method `searchList` should return a reference to the value if it is found; otherwise, `null` should be returned. Use your method in a test program that creates a list of integers. The program should prompt the user for a value to locate in the list.

ANS:

```
1  // Exercise 20.21 Solution: List3Test.java
2  // Program recursively searches a list of numbers.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class List3Test {
6
7     private static Object search( Integer input, ListNode node )
8     {
9        if ( node == null )
10          return null;
11       else if ( input.compareTo( node.getObject() ) == 0 )
12          return node.getObject();
13       else
14          return search( input, node.getNext() );
15    }
16
17    public static void main( String args[] )
18    {
19       List3 list = new List3();
20       Integer number = null;
21
22       // create objects to store in the List
23       for ( int i = 1; i <= 25; i++ ) {
24          number = new Integer( ( int ) ( Math.random() * 101 ) );
25          list.insert( number );
26       }
27
28       list.print();
29
30       number = ( Integer ) search( new Integer ( 34 ), list.firstNode );
31       if ( number != null )
32          System.out.println( "Value found: 34" );
33       else
34          System.out.println( "Value not found: 34" );
35
36       number = ( Integer ) search( new Integer ( 50 ), list.firstNode );
37       if ( number != null )
38          System.out.println( "Value found: 50" );
```

```
39         else
40            System.out.println( "Value not found: 50" );
41
42         number = ( Integer ) search( new Integer ( 72 ), list.firstNode );
43         if ( number != null )
44            System.out.println( "Value found: 72" );
45         else
46            System.out.println( "Value not found: 72" );
47      }
48
49   }  // end class List3Test
```

```
The list is: 0 5 8 9 16 18 22 22 22 27 27 27 36 37 41 48 49 52 66 69 72 74 75
88 93

Value not found: 34
Value not found: 50
Value found: 72
```

20.22 (*Binary Tree Delete*) In this exercise, we discuss deleting items from binary search trees. The deletion algorithm is not as straightforward as the insertion algorithm. There are three cases that are encountered when deleting an item—the item is contained in a leaf node (i.e., it has no children), the item is contained in a node that has one child or the item is contained in a node that has two children.

If the item to be deleted is contained in a leaf node, the node is deleted and the reference in the parent node is set to null.

If the item to be deleted is contained in a node with one child, the reference in the parent node is set to reference the child node and the node containing the data item is deleted. This causes the child node to take the place of the deleted node in the tree.

The last case is the most difficult. When a node with two children is deleted, another node in the tree must take its place. However, the reference in the parent node cannot simply be assigned to reference one of the children of the node to be deleted. In most cases, the resulting binary search tree would not adhere to the following characteristic of binary search trees (with no duplicate values): *The values in any left subtree are less than the value in the parent node, and the values in any right subtree are greater than the value in the parent node.*

Which node is used as a *replacement node* to maintain this characteristic? It is either the node containing the largest value in the tree less than the value in the node being deleted, or the node containing the smallest value in the tree greater than the value in the node being deleted. Let us consider the node with the smaller value. In a binary search tree, the largest value less than a parent's value is located in the left subtree of the parent node and is guaranteed to be contained in the rightmost node of the subtree. This node is located by walking down the left subtree to the right until the reference to the right child of the current node is null. We are now referencing the replacement node, which is either a leaf node or a node with one child to its left. If the replacement node is a leaf node, the steps to perform the deletion are as follows:

a) Store the reference to the node to be deleted in a temporary reference variable.
b) Set the reference in the parent of the node being deleted to reference the replacement node.
c) Set the reference in the parent of the replacement node to `null`.
d) Set the reference to the right subtree in the replacement node to reference the right subtree of the node to be deleted.
e) Set the reference to the left subtree in the replacement node to reference the left subtree of the node to be deleted.

The deletion steps for a replacement node with a left child are similar to those for a replacement node with no children, but the algorithm also must move the child into the replacement node's position in the tree. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

a) Store the reference to the node to be deleted in a temporary reference variable.
b) Set the reference in the parent of the node being deleted to reference the replacement node.
c) Set the reference in the parent of the replacement node reference to the left child of the replacement node.
d) Set the reference to the right subtree in the replacement node reference to the right subtree of the node to be deleted.
e) Set the reference to the left subtree in the replacement node to reference the left subtree of the node to be deleted.

Write method `deleteNode`, which takes as its argument the value to delete. Method `deleteNode` should locate in the tree the node containing the value to delete and use the algorithms discussed here to delete the node. If the value is not found in the tree, the method should print a message that indicates whether the value is deleted. Modify the program of Fig. 20.17 and Fig. 20.18 to use this method. After deleting an item, call the methods `inorderTraversal`, `preorderTraversal` and `postorderTraversal` to confirm that the delete operation was performed correctly.

ANS:

```
1  // Exercise 20.22 Solution: Tree5.java
2  // Deleting items from a tree.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class Tree5 {
6     protected TreeNode root;
7
8     public Tree5()
9     {
10       root = null;
11    }
12
13    // Insert new node in the binary tree. If the root is null, create the
14    // root here. Otherwise, call the insert method of class TreeNode.
15    public synchronized void insertNode( Integer d )
16    {
17       if ( root == null )
18          root = new TreeNode( d.intValue() );
19
20       else
21          root.insert( d.intValue() );
22    }
23
24    // Preorder Traversal
25    public synchronized void preorderTraversal()
26    {
27       preorderHelper( root );
28    }
29
30    // Recursive method to perform preorder traversal
31    private synchronized void preorderHelper( TreeNode node )
32    {
```

```
33          if ( node == null )
34             return;
35
36          System.out.print( node.getData() + " " );
37          preorderHelper( node.getLeft() );
38          preorderHelper( node.getRight() );
39       }
40
41       // Inorder Traversal
42       public synchronized void inorderTraversal()
43       {
44          inorderHelper( root );
45       }
46
47       // Recursive method to perform inorder traversal
48       private synchronized void inorderHelper( TreeNode node )
49       {
50          if ( node == null )
51             return;
52
53          inorderHelper( node.getLeft() );
54          System.out.print( node.getData() + " " );
55          inorderHelper( node.getRight() );
56       }
57
58       // Postorder Traversal
59       public synchronized void postorderTraversal()
60       {
61          postorderHelper( root );
62       }
63
64       // Recursive method to perform postorder traversal
65       private synchronized void postorderHelper( TreeNode node )
66       {
67          if ( node == null )
68             return;
69
70          postorderHelper( node.getLeft() );
71          postorderHelper( node.getRight() );
72          System.out.print( node.getData() + " " );
73       }
74
75       // top level method call
76       public synchronized void deleteItem( Integer d )
77       {
78          // if the tree is empty
79          if ( root == null )
80             return;
81
82          Integer test = ( Integer )root.getData();
83
84          // if the root is the value to be deleted
85          if ( test.compareTo( d ) == 0 ) {
86
```

```
87            // if the left child is null, set root to the right
88            if ( root.getLeft() == null )
89               root = root.getRight();
90
91            // if the right child is null, set root to the left
92            else if ( root.getRight() == null )
93               root = root.getLeft();
94
95            // both children have values
96            else {
97
98               boolean later = false;
99               TreeNode parent = root;
100
101              // find the largest value smaller than the root
102              TreeNode replacement = root.getLeft();
103              while ( replacement.getRight() != null ) {
104                 later = true;
105                 parent = replacement;
106                 replacement = replacement.getRight();
107              }
108
109              // create a temporary value
110              TreeNode temp = root;
111
112              // set the root to the replacement node
113              root = replacement;
114
115              // break the old link to the replacement node
116              if ( later )
117                 parent.setRight( replacement.getLeft() );
118              else
119                 // only occurs if replacement is immediately to the left
120                 parent.setLeft( replacement.getLeft() );
121
122              replacement.setRight( temp.getRight() );
123              replacement.setLeft( temp.getLeft() );
124
125           } // end else
126
127        } // end if
128
129        // the node is to the left of the root
130        else if ( test.compareTo( d ) > 0 )
131           root.setLeft( deleteNode( root.getLeft(), d ) );
132
133        // the node is to the right of the root
134        else if ( test.compareTo( d ) < 0 )
135           root.setRight( deleteNode( root.getRight(), d ) );
136     }
137
138     private synchronized TreeNode deleteNode( TreeNode top, Integer d )
139     {
```

```
140       // if the tree is empty
141       if ( top == null )
142          return top;
143
144       Integer test = ( Integer )top.getData();
145
146       // if the top is the value to be deleted
147       if ( test.compareTo( d ) == 0 ) {
148
149          // if the left child is null, set top to its right
150          if ( top.getLeft() == null )
151             return top.getRight();
152
153          // if the right child is null, set top to its left
154          else if ( top.getRight() == null )
155             return top.getLeft();
156
157          // both children have values
158          else {
159             boolean later = false;
160             TreeNode parent = root;
161
162             // find the largest value smaller than the top
163             TreeNode replacement = top.getLeft();
164             while ( replacement.getRight() != null ) {
165                later = true;
166                parent = replacement;
167                replacement = replacement.getRight();
168             }
169
170             // create a temporary value
171             TreeNode temp = top;
172
173             // set the top to the replacement node
174             top = replacement;
175
176             // break the old link to the replacement node
177             if ( later )
178                parent.setRight( replacement.getLeft() );
179             else
180                // only occurs if replacement is immediately to the left
181                parent.setLeft( replacement.getLeft() );
182
183             replacement.setRight( temp.getRight() );
184             replacement.setLeft( temp.getLeft() );
185
186          } // end else
187
188       } // end if
189
190       // the node is to the left of the root
191       else if ( test.compareTo( d ) > 0 )
192          top.setLeft( deleteNode( top.getLeft(), d ) );
193
```

```
194         // the node is to the right of the root
195         else if ( test.compareTo( d ) < 0 )
196            top.setRight( deleteNode( top.getRight(), d ) );
197
198         return top;
199      }
200
201   } // end class Tree5
```

ANS:

```
1   // Exercise 20.22 Solution: TreeTest3.java
2   // Program extends a binary tree to allow deletion.
3   import java.util.*;
4   import com.deitel.jhtp5.ch20.*;
5
6   public class TreeTest3 {
7
8      public static void main( String args[] )
9      {
10        Tree5 tree = new Tree5();
11        int number;
12
13        System.out.println( "Inserting the following values: " );
14
15        // create tree of random numbers
16        for ( int i = 1; i <= 20; i++ ) {
17           number = ( int ) ( Math.random() * 100 );
18           System.out.print( number + " " );
19           tree.insertNode( new Integer( number ) );
20        }
21
22        System.out.println ( "\n\nPreorder traversal" );
23        tree.preorderTraversal();
24
25        System.out.println ( "\n\nInorder traversal" );
26        tree.inorderTraversal();
27
28        System.out.println ( "\n\nPostorder traversal" );
29        tree.postorderTraversal();
30
31        int deleteValue = ( int ) ( Math.random() * 100 );
32
33        System.out.println( "\n\nDeleted value: " + deleteValue );
34
35        tree.deleteItem( new Integer( deleteValue ) );
36
37        System.out.println ( "\n\nPreorder traversal" );
38        tree.preorderTraversal();
39
40        System.out.println ( "\n\nInorder traversal" );
41        tree.inorderTraversal();
```

```
42
43          System.out.println ( "\n\nPostorder traversal" );
44          tree.postorderTraversal();
45
46       } // end method main
47
48    }  // end class TreeTest3
```

```
Inserting the following values:
51 15 95 19 40 70 42 51 87 37 86 33 41 89 8 83 56 19 38 31

Preorder traversal
51 15 8 19 40 37 33 31 38 42 41 95 70 56 87 86 83 89

Inorder traversal
8 15 19 31 33 37 38 40 41 42 51 56 70 83 86 87 89 95

Postorder traversal
8 31 33 38 37 41 42 40 19 15 56 83 86 89 87 70 95 51

Deleted value: 31


Preorder traversal
51 15 8 19 40 37 33 38 42 41 95 70 56 87 86 83 89

Inorder traversal
8 15 19 33 37 38 40 41 42 51 56 70 83 86 87 89 95

Postorder traversal
8 33 38 37 41 42 40 19 15 56 83 86 89 87 70 95 51
```

20.23 (*Binary Tree Search*) Write method `binaryTreeSearch`, which attempts to locate a specified value in a binary search tree object. The method should take as an argument a search key to be located. If the node containing the search key is found, the method should return a reference to that node; otherwise, the method should return a null reference.

ANS:

```
1  // Exercise 20.23 Solution: Tree4.java
2  // Class Tree4 definition.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class Tree4 extends Tree {
6
7     // begin binary tree search
8     public TreeNode binaryTreeSearch( Integer key )
9     {
10       return search( root, key );
11    }
12
```

```
13    // recursive method to perform binary tree search
14    private TreeNode search( TreeNode currentNode, Integer key )
15    {
16       // key not found
17       if ( currentNode == null )
18          return null;
19
20       // key found
21       if ( key.intValue() ==
22          ( ( Integer ) currentNode.getData() ).intValue() )
23
24          return currentNode;
25
26       // traverse down left child subtree
27       else if ( key.intValue() <
28          ( ( Integer ) currentNode.getData() ).intValue() )
29
30          return search(  currentNode.getLeft(), key );
31
32       //  traverse down right child subtree
33       else
34          return search(  currentNode.getRight(), key );
35    }
36
37 }  // end class Tree4
```

ANS:

```
1  // Exercise 20.23 Solution: Tree3Test.java
2  // Program performs a binary tree search.
3  import java.util.*;
4  import com.deitel.jhtp5.ch20.TreeNode;
5
6  public class Tree3Test {
7
8     public static void main( String args[] )
9     {
10       Tree4 tree = new Tree4();
11       int number;
12
13       System.out.println( "Inserting the following values: " );
14
15       // create Objects to store in tree
16       for ( int i = 1; i <= 10; i++ ) {
17          number = ( int ) ( Math.random() * 100 );
18          System.out.print( number + " " );
19          tree.insertNode( number );
20       }
21
22       // create Object to search for in tree
23       int searchNumber = ( int ) ( Math.random() * 100 );
24
```

```
25         // search
26         TreeNode myNode =
27            tree.binaryTreeSearch( new Integer( searchNumber ) );
28
29         // Object not in tree
30         if ( myNode == null )
31            System.out.println(
32               "\n" + searchNumber + " is not in the tree." );
33
34         // Object found in tree
35         else
36            System.out.println(
37               "\n" + searchNumber + " found in the tree." );
38      }
39
40   }  // end Tree3Test
```

```
Inserting the following values:
59 37 43 73 39 6 89 36 94 65
88 is not in the tree.
```

20.24 (*Level-Order Binary Tree Traversal*) The program of Fig. 20.17 and Fig. 20.18 illustrated three recursive methods of traversing a binary tree—inorder, preorder and postorder traversals. This exercise presents the *level-order traversal* of a binary tree, in which the node values are printed level-by-level, starting at the root node level. The nodes on each level are printed from left to right. The level-order traversal is not a recursive algorithm. It uses a queue object to control the output of the nodes. The algorithm is as follows:

a) Insert the root node in the queue.
b) While there are nodes left in the queue, do the following:
    Get the next node in the queue.
    Print the node's value.
    If the reference to the left child of the node is not null:
        Insert the left child node in the queue.
    If the reference to the right child of the node is not null:
        Insert the right child node in the queue.

Write method `levelOrder` to perform a level-order traversal of a binary tree object. Modify the program of Fig. 20.17 and Fig. 20.18 to use this method. [*Note*: You will also need to use queue-processing methods of Fig. 20.13 in this program.]

ANS:

```
1  // Exercise 20.24 Solution: Tree.java
2  // Class Tree definition
3  import com.deitel.jhtp5.ch20.*;
4
5  public class Tree {
6     protected TreeNode root;
7
8     public Tree()
9     {
10       root = null;
11    }
```

```
12
13    public void levelOrderTraversal()
14    {
15       Queue traversal = new Queue(); // create the queue
16
17       traversal.enqueue( root ); // add the root to the queue
18
19       try {
20
21          while ( true ) {
22
23             // get the first node in the queue
24             TreeNode recent = ( TreeNode ) traversal.dequeue();
25
26             // print the node
27             System.out.print( recent.getData() + " " );
28
29             // add a left node if it exists
30             if ( recent.getLeft() != null )
31                traversal.enqueue( recent.getLeft() );
32
33             // add a right node if it exists
34             if ( recent.getRight() != null )
35                traversal.enqueue( recent.getRight() );
36          }
37       }
38
39       // the queue is empty and the method can end
40       catch ( EmptyListException e ) {
41       }
42    }
43
44    // Insert a new node in the binary tree. If the root is null, create
45    // the root here. Otherwise, call the insert method of class TreeNode.
46    public synchronized void insertNode( Integer d )
47    {
48       if ( root == null )
49          root = new TreeNode( d.intValue() );
50
51       else
52          root.insert( d.intValue() );
53    }
54
55    // Preorder Traversal
56    public synchronized void preorderTraversal()
57    {
58       preorderHelper( root );
59    }
60
61    // Recursive method to perform preorder traversal
62    private synchronized void preorderHelper( TreeNode node )
63    {
64       if ( node == null )
65          return;
```

```
66
67          System.out.print( node.getData() + " " );
68          preorderHelper( node.getLeft() );
69          preorderHelper( node.getRight() );
70       }
71
72       // Inorder Traversal
73       public synchronized void inorderTraversal()
74       {
75          inorderHelper( root );
76       }
77
78       // Recursive method to perform inorder traversal
79       private synchronized void inorderHelper( TreeNode node )
80       {
81          if ( node == null )
82             return;
83
84          inorderHelper( node.getLeft() );
85          System.out.print( node.getData() + " " );
86          inorderHelper( node.getRight() );
87       }
88
89       // Postorder Traversal
90       public synchronized void postorderTraversal()
91       {
92          postorderHelper( root );
93       }
94
95       // Recursive method to perform postorder traversal
96       private synchronized void postorderHelper( TreeNode node )
97       {
98          if ( node == null )
99             return;
100
101         postorderHelper( node.getLeft() );
102         postorderHelper( node.getRight() );
103         System.out.print( node.getData() + " " );
104      }
105
106   }  // end class Tree
```

ANS:

```
1  // Exercise 20.24 Solution: TreeTest.java
2  // This program tests the Tree class.
3  import java.util.*;
4
5  public class TestTree {
6
7     public static void main( String args[] )
8     {
9        Tree tree = new Tree();
```

```
10         int intVal;
11
12         System.out.println( "Inserting the following values: " );
13
14         // randomly generate numbers and insert in the tree
15         for ( int i = 1; i <= 10; i++ ) {
16            intVal = ( int ) ( Math.random() * 100 );
17            System.out.print( intVal + " " );
18            tree.insertNode( new Integer( intVal ) );
19         }
20
21         // print each of the traversals
22         System.out.println ( "\n\nPreorder traversal" );
23         tree.preorderTraversal();
24
25         System.out.println ( "\n\nInorder traversal" );
26         tree.inorderTraversal();
27
28         System.out.println ( "\n\nPostorder traversal" );
29         tree.postorderTraversal();
30
31         System.out.println( "\n\nLevel-order traversal" );
32         tree.levelOrderTraversal();
33         System.out.println();
34      }
35
36   }  // end class TestTree
```

```
Inserting the following values:
57 61 31 90 42 48 56 82 74 82

Preorder traversal
57 31 42 48 56 61 90 82 74

Inorder traversal
31 42 48 56 57 61 74 82 90

Postorder traversal
56 48 42 31 74 82 90 61 57

Level-order traversal
57 31 61 42 90 48 82 56 74
```

20.25 (*Printing Trees*) Write a recursive method `outputTree` to display a binary tree object on the screen. The method should output the tree row-by-row, with the top of the tree at the left of the screen and the bottom of the tree toward the right of the screen. Each row is output vertically. For example, the binary tree illustrated in Fig. 20.20 is output as shown in Fig. 20.21.

Note that the rightmost leaf node appears at the top of the output in the rightmost column and the root node appears at the left of the output. Each column of output starts five spaces to the right of the preceding column. Method `outputTree` should receive an argument `totalSpaces` representing the number of spaces preceding the value to be output. (This variable should start at zero so the root node is output at the left of the screen.) The method uses a modified inorder traversal to output the tree—it starts at the rightmost node in the tree and works back to the left. The algorithm is as follows:

While the reference to the current node is not null, perform the following:
Recursively call `outputTree` with the right subtree of the current node and `totalSpaces + 5`.
Use a `for` statement to count from 1 to `totalSpaces` and output spaces.
Output the value in the current node.
Set the reference to the current node to refer to the left subtree of the current node.
Increment `totalSpaces` by 5.

ANS:

```
1  // Exercise 20.25 Solution: Tree2.java
2  // Class Tree which can print itself.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class Tree2 {
6     private TreeNode root;
7
8     public Tree2()
9     {
10       root = null;
11    }
12
13    // insert a new node in the binary search tree
14    public synchronized void insertNode( Integer value )
15    {
16       if ( root == null )
17          root = new TreeNode( value.intValue() );
18
19       else
20          root.insert( value.intValue() );
21    }
22
23    // begin preorder traversal
24    public synchronized void preorderTraversal()
25    {
26       preorderHelper( root );
27    }
28
29    // recursive method to perform preorder traversal
30    private void preorderHelper( TreeNode node )
31    {
32       if ( node == null )
33          return;
34
35       System.out.print( node.data + " " );
36       preorderHelper( node.leftNode );
37       preorderHelper( node.rightNode );
38    }
39
40    // begin inorder traversal
41    public synchronized void inorderTraversal()
42    {
43       inorderHelper( root );
44    }
```

```
45
46     // recursive method to perform inorder traversal
47     private void inorderHelper( TreeNode node )
48     {
49        if ( node == null )
50           return;
51
52        inorderHelper( node.leftNode );
53        System.out.print( node.data + " " );
54        inorderHelper( node.rightNode );
55     }
56
57     // begin postorder traversal
58     public synchronized void postorderTraversal()
59     {
60        postorderHelper( root );
61     }
62
63     // recursive method to perform postorder traversal
64     private void postorderHelper( TreeNode node )
65     {
66        if ( node == null )
67           return;
68
69        postorderHelper( node.leftNode );
70        postorderHelper( node.rightNode );
71        System.out.print( node.data + " " );
72     }
73
74     // begin printing tree
75     public void outputTree()
76     {
77        outputTreeHelper( root, 0 );
78     }
79
80     // recursive method to print tree
81     private void outputTreeHelper( TreeNode currentNode, int spaces )
82     {
83        // recursively print right branch, then left
84        if ( currentNode != null ) {
85           outputTreeHelper( currentNode.getRight(), spaces + 5 );
86
87           for ( int k = 1; k <= spaces; k++ )
88              System.out.print( " " );
89
90           System.out.println( currentNode.getData().toString() );
91           outputTreeHelper( currentNode.getLeft(), spaces + 5 );
92        }
93     }
94
95  }  // end class Tree2
```

ANS:

```
1  // Exercise 20.25 Solution: Tree2Test.java
2  // This program tests the Tree2 class.
3  import java.util.*;
4
5  public class Tree2Test {
6
7     public static void main( String args[] )
8     {
9        Tree2 tree = new Tree2();
10       int intVal;
11
12       System.out.println( "Inserting the following values: " );
13
14       // create Objects to store in tree
15       for ( int i = 1; i <= 10; i++ ) {
16          intVal = ( int ) ( Math.random() * 100 );
17          System.out.print( intVal + " " );
18          tree.insertNode( new Integer( intVal ) );
19       }
20
21       // run three different traversal types
22       System.out.println ( "\n\nPreorder traversal" );
23       tree.preorderTraversal();
24
25       System.out.println ( "\n\nInorder traversal" );
26       tree.inorderTraversal();
27
28       System.out.println ( "\n\nPostorder traversal" );
29       tree.postorderTraversal();
30
31       // print a depiction of the tree
32       System.out.println( "\n\n" );
33       tree.outputTree();
34    }
35
36 }  // end class Tree2Test
```

```
Inserting the following values:
45 22 83 15 49 53 23 30 83 36

Preorder traversal
45 22 15 23 30 36 83 49 53

Inorder traversal
15 22 23 30 36 45 49 53 83

Postorder traversal
15 36 30 23 22 53 49 83 45


     83
               53
          49
45
                    36
               30
          23
     22
          15
```

SPECIAL SECTION: BUILDING YOUR OWN COMPILER

In Exercise 7.43 and Exercise 7.44, we introduced Simpletron Machine Language (SML), and you implemented a Simpletron computer simulator to execute programs written in SML. In this section, we build a compiler that converts programs written in a high-level programming language to SML. This section "ties" together the entire programming process. You will write programs in this new high-level language, compile these programs on the compiler you build and run the programs on the simulator you built in Exercise 7.44. You should make every effort to implement your compiler in an object-oriented manner.

20.26 (*The Simple Language*) Before we begin building the compiler, we discuss a simple, yet powerful high-level language similar to early versions of the popular language Basic. We call the language *Simple*. Every Simple *statement* consists of a *line number* and a Simple *instruction*. Line numbers must appear in ascending order. Each instruction begins with one of the following Simple *commands*: `rem`, `input`, `let`, `print`, `goto`, `if/goto` or `end` (see Fig. 20.22). All commands except `end` can be used repeatedly. Simple evaluates only integer expressions using the `+`, `-`, `*` and `/` operators. These operators have the same precedence as in Java. Parentheses can be used to change the order of evaluation of an expression.

Our Simple compiler recognizes only lowercase letters. All characters in a Simple file should be lowercase. (Uppercase letters result in a syntax error unless they appear in a `rem` statement, in which case they are ignored.) A *variable name* is a single letter. Simple does not allow descriptive variable names, so variables should be explained in remarks to indicate their use in a program. Simple uses only integer variables. Simple does not have variable declarations—merely mentioning a variable name in a program causes the variable to be declared and initialized to zero. The syntax of Simple does not allow string manipulation (reading a string, writing a string, comparing strings etc.). If a string is encountered in a Simple program (after a command other than `rem`), the compiler generates a syntax error. The first version of our compiler assumes that Simple programs are entered correctly. Exercise 20.29 asks the reader to modify the compiler to perform syntax error checking.

```
                99
          97
                92
     83
                72
          71
                69
49
                44
          40
                32
     28
                19
          18
                11
```

Fig. 20.21 Sample output of recursive method `outputTree`.

Command	Example statement	Description
`rem`	`50 rem this is a remark`	Any text following the command `rem` is for documentation purposes only and is ignored by the compiler.
`input`	`30 input x`	Display a question mark to prompt the user to enter an integer. Read that integer from the keyboard and store the integer in `x`.
`let`	`80 let u = 4 * (j - 56)`	Assign `u` the value of `4 * (j - 56)`. Note that an arbitrarily complex expression can appear to the right of the equal sign.
`print`	`10 print w`	Display the value of `w`.
`goto`	`70 goto 45`	Transfer program control to line `45`.
`if/goto`	`35 if i == z goto 80`	Compare `i` and `z` for equality and transfer program control to line `80` if the condition is true; otherwise, continue execution with the next statement.
`end`	`99 end`	Terminate program execution.

Fig. 20.22 Simple commands.

Simple uses the conditional `if`/`goto` and unconditional `goto` statements to alter the flow of control during program execution. If the condition in the `if`/`goto` statement is true, control is transferred to a specific line of the program. The following relational and equality operators are valid in an `if`/`goto` statement: <, >, <=, >=, == or !=. The precedence of these operators is the same as in Java.

Let us now consider several programs that demonstrate Simple's features. The first program (Fig. 20.23) reads two integers from the keyboard, stores the values in variables `a` and `b` and computes and prints their sum (stored in variable `c`).

The program of Fig. 20.24 determines and prints the larger of two integers. The integers are input from the keyboard and stored in `s` and `t`. The `if/goto` statement tests the condition `s >= t`. If the condition is true, control is transferred to line 90 and `s` is output; otherwise, `t` is output and control is transferred to the `end` statement in line 99, where the program terminates.

Simple does not provide a repetition statement (such as Java's `for`, `while` or `do...while`). However, Simple can simulate each of Java's repetition statements by using the `if/goto` and `goto` statements. Figure 20.25 uses a sentinel-controlled loop to calculate the squares of several integers. Each integer is input from the keyboard and stored in variable `j`. If the value entered is the sentinel value `-9999`, control is transferred to line 99, where the program terminates. Otherwise, `k` is assigned the square of `j`, `k` is output to the screen and control is passed to line 20, where the next integer is input.

```
1   10 rem   determine and print the sum of two integers
2   15 rem
3   20 rem   input the two integers
4   30 input a
5   40 input b
6   45 rem
7   50 rem   add integers and store result in c
8   60 let c = a + b
9   65 rem
10  70 rem   print the result
11  80 print c
12  90 rem   terminate program execution
13  99 end
```

Fig. 20.23 Simple program that determines the sum of two integers.

```
1   10 rem   determine and print the larger of two integers
2   20 input s
3   30 input t
4   32 rem
5   35 rem   test if s >= t
6   40 if s >= t goto 90
7   45 rem
8   50 rem   t is greater than s, so print t
9   60 print t
10  70 goto 99
11  75 rem
12  80 rem   s is greater than or equal to t, so print s
13  90 print s
14  99 end
```

Fig. 20.24 Simple program that finds the larger of two integers.

```
1   10 rem   calculate the squares of several integers
2   20 input j
3   23 rem
4   25 rem   test for sentinel value
5   30 if j == -9999 goto 99
```

Fig. 20.25 Calculate the squares of several integers. (Part 1 of 2.)

```
 6   33 rem
 7   35 rem   calculate square of j and assign result to k
 8   40 let k = j * j
 9   50 print k
10   53 rem
11   55 rem   loop to get next j
12   60 goto 20
13   99 end
```

Fig. 20.25 Calculate the squares of several integers. (Part 2 of 2.)

Using the sample programs of Fig. 20.23–Fig. 20.25 as your guide, write a Simple program to accomplish each of the following:

a) Input three integers, determine their average and print the result.
b) Use a sentinel-controlled loop to input 10 integers and compute and print their sum.
c) Use a counter-controlled loop to input 7 integers, some positive and some negative, and compute and print their average.
d) Input a series of integers and determine and print the largest. The first integer input indicates how many numbers should be processed.
e) Input 10 integers and print the smallest.
f) Calculate and print the sum of the even integers from 2 to 30.
g) Calculate and print the product of the odd integers from 1 to 9.

20.27 (*Building A Compiler; Prerequisites: Complete Exercise 7.43, Exercise 7.44, Exercise 20.12, Exercise 20.13 and Exercise 20.26*) Now that the Simple language has been presented (Exercise 20.26), we discuss how to build a Simple compiler. First, we consider the process by which a Simple program is converted to SML and executed by the Simpletron simulator (see Fig. 20.26). A file containing a Simple program is read by the compiler and converted to SML code. The SML code is output to a file on disk, in which SML instructions appear one per line. The SML file is then loaded into the Simpletron simulator, and the results are sent to a file on disk and to the screen. Note that the Simpletron program developed in Exercise 7.44 took its input from the keyboard. It must be modified to read from a file so it can run the programs produced by our compiler.

The Simple compiler performs two *passes* of the Simple program to convert it to SML. The first pass constructs a *symbol table* (object) in which every *line number* (object), *variable name* (object) and *constant* (object) of the Simple program is stored with its type and corresponding location in the final SML code (the symbol table is discussed in detail below). The first pass also produces the corresponding SML instruction object(s) for each of the Simple statements (object, etc.). If the Simple program contains statements that transfer control to a line later in the program, the first pass results in an SML program containing some "unfinished" instructions. The second pass of the compiler locates and completes the unfinished instructions and outputs the SML program to a file.

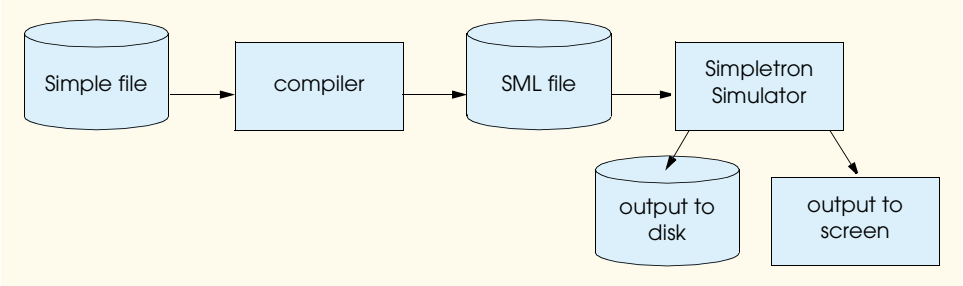


Fig. 20.26 Writing, compiling and executing a Simple language program.

First Pass

The compiler begins by reading one statement of the Simple program into memory. The line must be separated into its individual *tokens* (i.e., "pieces" of a statement) for processing and compilation. (The `StreamTokenizer` class from the `java.io` package can be used.) Recall that every statement begins with a line number followed by a command. As the compiler breaks a statement into tokens, if the token is a line number, a variable or a constant, it is placed in the symbol table. A line number is placed in the symbol table only if it is the first token in a statement. The `symbolTable` object is an array of `tableEntry` objects representing each symbol in the program. There is no restriction on the number of symbols that can appear in the program. Therefore, the `symbolTable` for a particular program could be large. Make the `symbolTable` a 100-element array for now. You can increase or decrease its size once the program is working.

Each `tableEntry` object contains three fields. Field `symbol` is an integer containing the Unicode representation of a variable (remember that variable names are single characters), a line number or a constant. Field `type` is one of the following characters indicating the symbol's type: `'C'` for constant, `'L'` for line number or `'V'` for variable. Field `location` contains the Simpletron memory location (00 to 99) to which the symbol refers. Simpletron memory is an array of 100 integers in which SML instructions and data are stored. For a line number, the location is the element in the Simpletron memory array at which the SML instructions for the Simple statement begin. For a variable or constant, the location is the element in the Simpletron memory array in which the variable or constant is stored. Variables and constants are allocated from the end of Simpletron's memory backwards. The first variable or constant is stored at location 99, the next at location 98, etc.

The symbol table plays an integral part in converting Simple programs to SML. We learned in Chapter 7 that an SML instruction is a four-digit integer comprised of two parts—the *operation code* and the *operand*. The operation code is determined by commands in Simple. For example, the simple command `input` corresponds to SML operation code 10 (read), and the Simple command `print` corresponds to SML operation code 11 (write). The operand is a memory location containing the data on which the operation code performs its task (e.g., operation code 10 reads a value from the keyboard and stores it in the memory location specified by the operand). The compiler searches `symbolTable` to determine the Simpletron memory location for each symbol, so the corresponding location can be used to complete the SML instructions.

The compilation of each Simple statement is based on its command. For example, after the line number in a `rem` statement is inserted in the symbol table, the remainder of the statement is ignored by the compiler because a remark is for documentation purposes only. The `input`, `print`, `goto` and `end` statements correspond to the SML *read*, *write*, *branch* (to a specific location) and *halt* instructions. Statements containing these Simple commands are converted directly to SML. (*Note*: A `goto` statement may contain an unresolved reference if the specified line number refers to a statement further into the Simple program file; this is sometimes called a forward reference.)

When a `goto` statement is compiled with an unresolved reference, the SML instruction must be *flagged* to indicate that the second pass of the compiler must complete the instruction. The flags are stored in a 100-element array `flags` of type `int` in which each element is initialized to `-1`. If the memory location to which a line number in the Simple program refers is not yet known (i.e., it is not in the symbol table), the line number is stored in array `flags` in the element with the same index as the incomplete instruction. The operand of the incomplete instruction is set to `00` temporarily. For example, an unconditional branch instruction (making a forward reference) is left as `+4000` until the second pass of the compiler. The second pass of the compiler will be described shortly.

Compilation of `if`/`goto` and `let` statements is more complicated than other statements—they are the only statements that produce more than one SML instruction. For an `if`/`goto` statement, the compiler produces code to test the condition and to branch to another line if necessary. The result of the branch could be an unresolved reference. Each of the relational and equality operators can be simulated by using SML's *branch zero* and *branch negative* instructions (or possibly a combination of both).

For a `let` statement, the compiler produces code to evaluate an arbitrarily complex arithmetic expression consisting of integer variables and/or constants. Expressions should separate each operand and operator with spaces. Exercise 20.12 and Exercise 20.13 presented the infix-to-postfix conversion algorithm and the postfix evaluation algorithm used by compilers to evaluate expressions. Before proceeding with your compiler, you should complete each of these exercises. When a compiler encounters an expression, it converts the expression from infix notation to postfix notation, then evaluates the postfix expression.

How is it that the compiler produces the machine language to evaluate an expression containing variables? The postfix evaluation algorithm contains a "hook" where the compiler can generate SML instructions rather than actually evaluating the expression. To enable this "hook" in the compiler, the postfix evaluation algorithm must be modified to search the symbol table for each symbol it encounters (and possibly insert it), determine the symbol's corresponding memory location and *push the memory location on the stack (instead of the symbol).* When an operator is encountered in the postfix expression, the two memory locations at the top of the stack are popped, and machine language for effecting the operation is produced by using the memory locations as operands. The result of each subexpression is stored in a temporary location in memory and pushed back onto the stack so the evaluation of the postfix expression can continue. When postfix evaluation is complete, the memory location containing the result is the only location left on the stack. This is popped, and SML instructions are generated to assign the result to the variable at the left of the `let` statement.

Second Pass

The second pass of the compiler performs two tasks: Resolve any unresolved references and output the SML code to a file. Resolution of references occurs as follows:

a) Search the `flags` array for an unresolved reference (i.e., an element with a value other than `-1`).
b) Locate the object in array `symbolTable` containing the symbol stored in the `flags` array (be sure that the type of the symbol is `'L'` for line number).
c) Insert the memory location from field `location` into the instruction with the unresolved reference (remember that an instruction containing an unresolved reference has operand `00`).
d) Repeat steps (a), (b) and (c) until the end of the `flags` array is reached.

After the resolution process is complete, the entire array containing the SML code is output to a disk file with one SML instruction per line. This file can be read by the Simpletron for execution (after the simulator is modified to read its input from a file). Compiling your first Simple program into an SML file and executing that file should give you a real sense of personal accomplishment.

A Complete Example

The following example illustrates complete conversion of a Simple program to SML as it will be performed by the Simple compiler. Consider a Simple program that inputs an integer and sums the values from 1 to that integer. The program and the SML instructions produced by the first pass of the Simple compiler are illustrated in Fig. 20.27. The symbol table constructed by the first pass is shown in Fig. 20.28.

Most Simple statements convert directly to single SML instructions. The exceptions in this program are remarks, the `if`/`goto` statement in line `20` and the `let` statements. Remarks do not translate into machine language. However, the line number for a remark is placed in the symbol table in case the line number is referenced in a `goto` statement or an `if`/`goto` statement. Line `20` of the program specifies that, if the condition `y == x` is true, program control is transferred to line `60`. Since line `60` appears later in the program, the first pass of the compiler has not as yet placed `60` in the symbol table. (Statement line numbers are placed in the symbol table only when they appear as the first token in a statement.) Therefore, it is not possible at this time to determine the operand of the SML *branch zero* instruction at location `03` in the array of SML instructions. The compiler places `60` in location `03` of the `flags` array to indicate that the second pass completes this instruction.

Simple program	SML location and instruction	Description
`5 rem sum 1 to x`	*none*	`rem` ignored
`10 input x`	`00 +1099`	read `x` into location `99`
`15 rem check y == x`	*none*	`rem` ignored
`20 if y == x goto 60`	`01 +2098`	load `y` (`98`) into accumulator
	`02 +3199`	sub `x` (`99`) from accumulator
	`03 +4200`	branch zero to unresolved location
`25 rem increment y`	*none*	`rem` ignored
`30 let y = y + 1`	`04 +2098`	load `y` into accumulator
	`05 +3097`	add `1` (`97`) to accumulator
	`06 +2196`	store in temporary location `96`
	`07 +2096`	load from temporary location `96`
	`08 +2198`	store accumulator in `y`
`35 rem add y to total`	*none*	`rem` ignored
`40 let t = t + y`	`09 +2095`	load `t` (`95`) into accumulator
	`10 +3098`	add `y` to accumulator
	`11 +2194`	store in temporary location `94`
	`12 +2094`	load from temporary location `94`
	`13 +2195`	store accumulator in `t`
`45 rem loop y`	*none*	`rem` ignored
`50 goto 20`	`14 +4001`	branch to location `01`
`55 rem output result`	*none*	`rem` ignored
`60 print t`	`15 +1195`	output `t` to screen
`99 end`	`16 +4300`	terminate execution

Fig. 20.27 SML instructions produced after the compiler's first pass.

Symbol	Type	Location
5	L	00
10	L	00
'x'	V	99
15	L	01
20	L	01
'y'	V	98
25	L	04
30	L	04
1	C	97
35	L	09
40	L	09
't'	V	95
45	L	14
50	L	14
55	L	15
60	L	15
99	L	16

Fig. 20.28 Symbol table for program of Fig. 20.27.

We must keep track of the next instruction location in the SML array because there is not a one-to-one correspondence between Simple statements and SML instructions. For example, the `if`/`goto` statement of line 20 compiles into three SML instructions. Each time an instruction is produced, we must increment the *instruction counter* to the next location in the SML array. Note that the size of Simpletron's memory could present a problem for Simple programs with many statements, variables and constants. It is conceivable that the compiler will run out of memory. To test for this case, your program should contain a *data counter* to keep track of the location at which the next variable or constant will be stored in the SML array. If the value of the instruction counter is larger than the value of the data counter, the SML array is full. In this case, the compilation process should terminate, and the compiler should print an error message indicating that it ran out of memory during compilation. This serves to emphasize that, although the programmer is freed from the burdens of managing memory by the compiler, the compiler itself must carefully determine the placement of instructions and data in memory and must check for such errors as memory being exhausted during the compilation process.

A Step-by-Step View of the Compilation Process

Let us now walk through the compilation process for the Simple program in Fig. 20.27. The compiler reads the first line of the program

```
5 rem sum 1 to x
```

into memory. The first token in the statement (the line number) is determined using the `StringTokenizer` class. (See Chapter 11 for a discussion of this class.) The token returned by the `StringTokenizer` is converted to an integer by using `static` method `Integer.parseInt()`, so

the symbol 5 can be located in the symbol table. If the symbol is not found, it is inserted in the symbol table.

We are at the beginning of the program and this is the first line, and no symbols are in the table yet. Therefore, 5 is inserted into the symbol table as type L (line number) and assigned the first location in the SML array (00). Although this line is a remark, a space in the symbol table is still allocated for the line number (in case it is referenced by a goto or an if/goto). No SML instruction is generated for a rem statement, so the instruction counter is not incremented.

```
10 input x
```

is tokenized next. The line number 10 is placed in the symbol table as type L and assigned the first location in the SML array (00 because a remark began the program, so the instruction counter is currently 00). The command input indicates that the next token is a variable (only a variable can appear in an input statement). input corresponds directly to an SML operation code; therefore, the compiler simply has to determine the location of x in the SML array. Symbol x is not found in the symbol table. So, it is inserted into the symbol table as the Unicode representation of x, given type V and assigned location 99 in the SML array (data storage begins at 99 and is allocated backwards). SML code can now be generated for this statement. Operation code 10 (the SML read operation code) is multiplied by 100, and the location of x (as determined in the symbol table) is added to complete the instruction. The instruction is then stored in the SML array at location 00. The instruction counter is incremented by one, because a single SML instruction was produced.

The statement

```
15 rem   check y == x
```

is tokenized next. The symbol table is searched for line number 15 (which is not found). The line number is inserted as type L and assigned the next location in the array, 01. (Remember that rem statements do not produce code, so the instruction counter is not incremented.)

The statement

```
20 if y == x goto 60
```

is tokenized next. Line number 20 is inserted in the symbol table and given type L at the next location in the SML array 01. The command if indicates that a condition is to be evaluated. The variable y is not found in the symbol table, so it is inserted and given the type V and the SML location 98. Next, SML instructions are generated to evaluate the condition. There is no direct equivalent in SML for the if/goto; it must be simulated by performing a calculation using x and y and branching according to the result. If y is equal to x, the result of subtracting x from y is zero, so the *branch zero* instruction can be used with the result of the calculation to simulate the if/goto statement. The first step requires that y be loaded (from SML location 98) into the accumulator. This produces the instruction 01 +2098. Next, x is subtracted from the accumulator. This produces the instruction 02 +3199. The value in the accumulator may be zero, positive or negative. The operator is ==, so we want to *branch zero*. First, the symbol table is searched for the branch location (60 in this case), which is not found. So, 60 is placed in the flags array at location 03, and the instruction 03 +4200 is generated. (We cannot add the branch location because we have not yet assigned a location to line 60 in the SML array.) The instruction counter is incremented to 04.

The compiler proceeds to the statement

```
25 rem   increment y
```

The line number 25 is inserted in the symbol table as type L and assigned SML location 04. The instruction counter is not incremented.

When the statement

```
30 let y = y + 1
```

is tokenized, the line number 30 is inserted in the symbol table as type L and assigned SML location 04. Command `let` indicates that the line is an assignment statement. First, all the symbols on the line are inserted in the symbol table (if they are not already there). The integer 1 is added to the symbol table as type C and assigned SML location 97. Next, the right side of the assignment is converted from infix to postfix notation. Then the postfix expression (`y 1 +`) is evaluated. Symbol `y` is located in the symbol table, and its corresponding memory location is pushed onto the stack. Symbol `1` is also located in the symbol table, and its corresponding memory location is pushed onto the stack. When the operator + is encountered, the postfix evaluator pops the stack into the right operand of the operator and pops the stack again into the left operand of the operator, then produces the SML instructions

```
04 +2098    (load y)
05 +3097    (add 1)
```

The result of the expression is stored in a temporary location in memory (96) with instruction

```
06 +2196    (store temporary)
```

and the temporary location is pushed onto the stack. Now that the expression has been evaluated, the result must be stored in `y` (i.e., the variable on the left side of =). So, the temporary location is loaded into the accumulator and the accumulator is stored in `y` with the instructions

```
07 +2096    (load temporary)
08 +2198    (store y)
```

The reader should immediately notice that SML instructions appear to be redundant. We will discuss this issue shortly.

When the statement

```
35 rem   add y to total
```

is tokenized, line number 35 is inserted in the symbol table as type L and assigned location 09.

The statement

```
40 let t = t + y
```

is similar to line 30. The variable `t` is inserted in the symbol table as type V and assigned SML location 95. The instructions follow the same logic and format as line 30, and the instructions 09 +2095, 10 +3098, 11 +2194, 12 +2094 and 13 +2195 are generated. Note that the result of `t + y` is assigned to temporary location 94 before being assigned to `t` (95). Once again, the reader should note that the instructions in memory locations 11 and 12 appear to be redundant. Again, we will discuss this shortly.

The statement

```
45 rem   loop y
```

is a remark, so line 45 is added to the symbol table as type L and assigned SML location 14.

The statement

```
50 goto 20
```

transfers control to line 20. Line number 50 is inserted in the symbol table as type L and assigned SML location 14. The equivalent of `goto` in SML is the *unconditional branch* (40) instruction that transfers control to a specific SML location. The compiler searches the symbol table for line 20 and finds that it corresponds to SML location 01. The operation code (40) is multiplied by 100, and location 01 is added to it to produce the instruction 14 +4001.

The statement

```
55 rem   output result
```

is a remark, so line 55 is inserted in the symbol table as type L and assigned SML location 15.

The statement

```
60 print t
```

is an output statement. Line number 60 is inserted in the symbol table as type L and assigned SML location 15. The equivalent of print in SML is operation code 11 (*write*). The location of t is determined from the symbol table and added to the result of the operation code multiplied by 100.

The statement

```
99 end
```

is the final line of the program. Line number 99 is stored in the symbol table as type L and assigned SML location 16. The end command produces the SML instruction +4300 (43 is *halt* in SML), which is written as the final instruction in the SML memory array.

This completes the first pass of the compiler. We now consider the second pass. The flags array is searched for values other than -1. Location 03 contains 60, so the compiler knows that instruction 03 is incomplete. The compiler completes the instruction by searching the symbol table for 60, determining its location and adding the location to the incomplete instruction. In this case, the search determines that line 60 corresponds to SML location 15, so the completed instruction 03 +4215 is produced, replacing 03 +4200. The Simple program has now been compiled successfully.

To build the compiler, you will have to perform each of the following tasks:

a) Modify the Simpletron simulator program you wrote in Exercise 7.44 to take its input from a file specified by the user (see Chapter 17). The simulator should output its results to a disk file in the same format as the screen output. Convert the simulator to be an object-oriented program. In particular, make each part of the hardware an object. Arrange the instruction types into a class hierarchy using inheritance. Then execute the program polymorphically simply by telling each instruction to execute itself with an executeInstruction message.

b) Modify the infix-to-postfix evaluation algorithm of Exercise 20.12 to process multidigit integer operands and single-letter variable name operands. (*Hint*: Class StringTokenizer can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers by using Integer class method parseInt.) [*Note*: The data representation of the postfix expression must be altered to support variable names and integer constants.]

c) Modify the postfix evaluation algorithm to process multidigit integer operands and variable name operands. Also, the algorithm should now implement the "hook" discussed earlier so that SML instructions are produced rather than directly evaluating the expression. (*Hint*: Class StringTokenizer can be used to locate each constant and variable in an expression, and constants can be converted from strings to integers by using Integer class method parseInt.) [*Note*: The data representation of the postfix expression must be altered to support variable names and integer constants.]

d) Build the compiler. Incorporate parts b) and c) for evaluating expressions in let statements. Your program should contain a method that performs the first pass of the compiler and a method that performs the second pass of the compiler. Both methods can call other methods to accomplish their tasks. Make your compiler as object oriented as possible.

20.28 (*Optimizing the Simple Compiler*) When a program is compiled and converted into SML, a set of instructions is generated. Certain combinations of instructions often repeat themselves, usually

in triplets called *productions*. A production normally consists of three instructions, such as *load*, *add* and *store*. For example, Fig. 20.29 illustrates five of the SML instructions that were produced in the compilation of the program in Fig. 20.27. The first three instructions are the production that adds 1 to y. Note that instructions 06 and 07 store the accumulator value in temporary location 96, then load the value back into the accumulator so instruction 08 can store the value in location 98. Often a production is followed by a load instruction for the same location that was just stored. This code can be *optimized* by eliminating the store instruction and the subsequent load instruction that operate on the same memory location, thus enabling the Simpletron to execute the program faster. Figure 20.30 illustrates the optimized SML for the program of Fig. 20.27. Note that there are four fewer instructions in the optimized code—a memory-space savings of 25%.

```
1  04  +2098  (load)
2  05  +3097  (add)
3  06  +2196  (store)
4  07  +2096  (load)
5  08  +2198  (store)
```

Fig. 20.29 Unoptimized code from the program of Fig. 19.25.

Simple program	SML location and instruction	Description
`5 rem sum 1 to x`	*none*	`rem` ignored
`10 input x`	00 +1099	read x into location 99
`15 rem check y == x`	*none*	`rem` ignored
`20 if y == x goto 60`	01 +2098	load y (98) into accumulator
	02 +3199	sub x (99) from accumulator
	03 +4211	branch to location 11 if zero
`25 rem increment y`	*none*	`rem` ignored
`30 let y = y + 1`	04 +2098	load y into accumulator
	05 +3097	add 1 (97) to accumulator
	06 +2198	store accumulator in y (98)
`35 rem add y to total`	*none*	`rem` ignored
`40 let t = t + y`	07 +2096	load t from location (96)
	08 +3098	add y (98) accumulator
	09 +2196	store accumulator in t (96)
`45 rem loop y`	*none*	`rem` ignored
`50 goto 20`	10 +4001	branch to location 01
`55 rem output result`	*none*	`rem` ignored
`60 print t`	11 +1196	output t (96) to screen
`99 end`	12 +4300	terminate execution

Fig. 20.30 Optimized code for the program of Fig. 20.27.

20.29 (*Modifications to the Simple Compiler*) Perform the following modifications to the Simple compiler. Some of these modifications might also require modifications to the Simpletron simulator program written in Exercise 7.44.

a) Allow the remainder operator (%) to be used in `let` statements. Simpletron Machine Language must be modified to include a remainder instruction.

b) Allow exponentiation in a `let` statement using `^` as the exponentiation operator. Simpletron Machine Language must be modified to include an exponentiation instruction.

c) Allow the compiler to recognize uppercase and lowercase letters in Simple statements (e.g., `'A'` is equivalent to `'a'`). No modifications to the Simpletron simulator are required.

d) Allow `input` statements to read values for multiple variables such as `input x, y`. No modifications to the Simpletron simulator are required to perform this enhancement to the Simple compiler.

e) Allow the compiler to output multiple values from a single `print` statement, such as `print a, b, c`. No modifications to the Simpletron simulator are required to perform this enhancement.

f) Add syntax-checking capabilities to the compiler so error messages are output when syntax errors are encountered in a Simple program. No modifications to the Simpletron simulator are required.

g) Allow arrays of integers. No modifications to the Simpletron simulator are required to perform this enhancement.

h) Allow subroutines specified by the Simple commands `gosub` and `return`. Command `gosub` passes program control to a subroutine and command `return` passes control back to the statement after the `gosub`. This is similar to a method call in Java. The same subroutine can be called from many `gosub` commands distributed throughout a program. No modifications to the Simpletron simulator are required.

i) Allow repetition statements of the form

```
for x = 2 to 10 step 2
    Simple statements
next
```

This `for` statement loops from `2` to `10` with an increment of `2`. The `next` line marks the end of the body of the `for` line. No modifications to the Simpletron simulator are required.

j) Allow repetition statements of the form

```
for x = 2 to 10
    Simple statements
next
```

This `for` statement loops from `2` to `10` with a default increment of `1`. No modifications to the Simpletron simulator are required.

k) Allow the compiler to process string input and output. This requires the Simpletron simulator to be modified to process and store string values. [*Hint*: Each Simpletron word (i.e., memory location) can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the Unicode decimal equivalent of a character. Add a machine-language instruction that will print a string beginning at a certain Simpletron memory location. The first half of the Simpletron word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half word contains one Unicode character expressed as two decimal digits. The machine language instruction checks the length and prints the string by translating each two-digit number into its equivalent character.]

l) Allow the compiler to process floating-point values in addition to integers. The Simpletron Simulator must also be modified to process floating-point values.

20.30 (*A Simple Interpreter*) An interpreter is a program that reads a high-level language program statement, determines the operation to be performed by the statement and executes the operation immediately. The high-level language program is not converted into machine language first. Interpreters execute more slowly than compilers do, because each statement encountered in the program being interpreted must first be deciphered at execution time. If statements are contained in a loop, the statements are deciphered each time they are encountered in the loop. Early versions of the Basic programming language were implemented as interpreters. Most Java programs are run interpretively.

Write an interpreter for the Simple language discussed in Exercise 20.26. The program should use the infix-to-postfix converter developed in Exercise 20.12 and the postfix evaluator developed in Exercise 20.13 to evaluate expressions in a `let` statement. The same restrictions placed on the Simple language in Exercise 20.26 should be adhered to in this program. Test the interpreter with the Simple programs written in Exercise 20.26. Compare the results of running these programs in the interpreter with the results of compiling the Simple programs and running them in the Simpletron simulator built in Exercise 7.44.

20.31 *(Insert/Delete Anywhere in a Linked List)* Our linked-list class allowed insertions and deletions at only the front and the back of the linked list. These capabilities were convenient for us when we used inheritance or composition to produce a stack class and a queue class with a minimal amount of code simply by reusing the list class. Linked lists are normally more general than those we provided. Modify the linked-list class we developed in this chapter to handle insertions and deletions anywhere in the list.

ANS:

```
1  // Exercise 20.31 Solution: List2.java
2  // Definition of class List2.
3  import com.deitel.jhtp5.ch20.List;
4  import com.deitel.jhtp5.ch20.ListNode;
5
6  class List2 extends List {
7
8     // delete node
9     public boolean deleteNode( Integer value )
10    {
11       // empty list
12       if ( isEmpty() )
13          return false;
14
15       else {
16
17          // delete first node
18          if ( ( ( Integer ) firstNode.getObject() ).intValue() ==
19                                                value.intValue() ) {
20
21             removeFromFront();
22             return true;
23          }
24
25          // delete last node
26          else if ( ( ( Integer ) lastNode.getObject() ).intValue() ==
27                                                   value.intValue() ) {
28
29             removeFromBack();
```

```
30             return true;
31          }
32
33          // delete internal node
34          else {
35             ListNode current = firstNode.getNext(), previous = firstNode;
36
37             // search for node to be deleted
38             while ( current != lastNode && ( ( Integer )
39                current.getObject() ).intValue() < value.intValue() ) {
40
41                previous = current;
42                current = current.getNext();
43             }
44
45             // node to be deleted found in list
46             if ( ( ( Integer ) current.getObject() ).intValue() ==
47                                                 value.intValue() ) {
48
49                previous.setNext( current.getNext() );
50                return true;
51             }
52
53             // node with appropriate value is not in list
54             else
55                return false;
56          }
57       }
58
59    }  // end method deleteNode
60
61    // insert new node
62    public void insertInOrder( Integer value )
63    {
64       // empty list
65       if ( isEmpty() ) {
66          ListNode newNode = new ListNode( value );
67          firstNode = lastNode = newNode;
68       }
69
70       else {
71
72          // insert at front
73          if ( ( ( Integer ) firstNode.getObject() ).intValue() >
74                                               value.intValue() )
75
76             insertAtFront( value );
77
78          // insert at back
79          else if ( ( ( Integer ) lastNode.getObject() ).intValue() <
80                                                    value.intValue() )
81
82             insertAtBack( value );
83
```

```
84            // insert within list
85            else {
86               ListNode current = firstNode.getNext(), previous = firstNode,
87                        newNode = new ListNode( value );
88
89               // search for appropriate insertion location
90               while ( current != lastNode && ( ( Integer )
91                  current.getObject() ).intValue() < value.intValue() ) {
92
93                  previous = current;
94                  current = current.getNext();
95               }
96
97               // insert new node
98               previous.setNext( newNode );
99               newNode.setNext( current );
100           }
101        }
102
103     }  // end method insertInOrder
104
105  }  // end class List2
```

ANS:

```
1  // Exercise 20.31 Solution: ListTest2.java
2  // Program allows insertion and deletion anywhere in a linked list.
3  import com.deitel.jhtp5.ch20.EmptyListException;
4
5  public class ListTest2 {
6
7     public static void main( String args[] )
8     {
9        List2 list = new List2();
10
11       // create list
12       for ( int a = 20; a >= 2; a -= 2 )
13          list.insertInOrder( new Integer( a ) );
14
15       // print list before insertion
16       System.out.println( "Before insertion:" );
17       list.print();
18
19       Integer i = new Integer( 7 ); // insert new object
20       list.insertInOrder( i );
21       System.out.println( i.toString() + " inserted.\n" );
22
23       // print list after insertion
24       System.out.println( "After insertion:" );
25       list.print();
26
27       try {
```

```
28
29            // delete node
30            if ( list.deleteNode( i ) )
31               System.out.println( i.toString() + " deleted.\n" );
32
33            else
34               System.out.println( i.toString() + " not deleted.\n" );
35
36            // print list after deletion
37            System.out.println( "After deletion:" );
38            list.print();
39            System.out.println();
40         }
41
42         catch ( EmptyListException exception ) {
43            System.err.println( "\n" + exception.toString() );
44         }
45      }
46
47   }  // end class ListTest2
```

```
Before insertion:
The list is: 2 4 6 8 10 12 14 16 18 20

7 inserted.

After insertion:
The list is: 2 4 6 7 8 10 12 14 16 18 20

7 deleted.

After deletion:
The list is: 2 4 6 8 10 12 14 16 18 20
```

20.32 *(Lists and Queues without Tail References)* Our implementation of a linked list (Fig. 20.3) used both a `firstNode` and a `lastNode`. The `lastNode` was useful for the `insertAtBack` and `removeFromBack` methods of the `List` class. The `insertAtBack` method corresponds to the `enqueue` method of the `Queue` class.

Rewrite the `List` class so that it does not use a `lastNode`. Thus, any operations on the tail of a list must begin searching the list from the front. Does this affect our implementation of the `Queue` class (Fig. 20.13)?

ANS:

```
1  // Exercise 20.32 Solution: List.java
2  // An implementation of a list without a lastNode.
3
4  public class List {
5     protected ListNode firstNode;
6     private String name;  // String like "list" used in printing
7
8     public List( String s )
9     {
```

```
10          name = s;
11          firstNode = null;
12       }
13
14       public List()
15       {
16          this( "list" );
17       }
18
19       // insert an object at the front of the list
20       public synchronized void insertAtFront( Object insertItem )
21       {
22          if ( isEmpty() )
23             firstNode = new ListNode( insertItem );
24
25          else
26             firstNode = new ListNode( insertItem, firstNode );
27       }
28
29       // insert an object at the end of the list
30       public synchronized void insertAtBack( Object insertItem )
31       {
32          if ( isEmpty() )
33             firstNode = new ListNode( insertItem );
34
35          else if ( firstNode.next == null )
36             firstNode.next = new ListNode( insertItem );
37
38          else {
39             ListNode temp = firstNode;
40             while ( temp.next != null )
41                temp = temp.next;
42
43             temp.next = new ListNode( insertItem );
44          }
45       }
46
47       // remove the first node from the list.
48       public synchronized Object removeFromFront() throws EmptyListException
49       {
50          Object removeItem = null;
51
52          if ( isEmpty() )
53             throw new EmptyListException( name );
54
55          removeItem = firstNode.data;  // retrieve the data
56          firstNode = firstNode.next;
57
58          return removeItem;
59       }
60
61       // remove the last node from the list.
62       public synchronized Object removeFromBack() throws EmptyListException
63       {
```

```
64          Object removeItem = null;
65
66          if ( isEmpty() )
67             throw new EmptyListException( name );
68
69          if ( firstNode.next == null ) {
70             removeItem = firstNode.data;  // retrieve the data
71             firstNode = null;
72          }
73
74          else {
75             ListNode temp = firstNode;
76             while ( temp.next.next != null )
77                temp = temp.next;
78
79             removeItem = temp.next.data;  // retrieve the data
80             temp.next = null;
81          }
82
83          return removeItem;
84       }
85
86       // return true if the list is empty
87       public synchronized boolean isEmpty()
88       {
89          return firstNode == null;
90       }
91
92       // output the list contents
93       public synchronized void print()
94       {
95          if ( isEmpty() ) {
96             System.out.print( "\n" );
97             return;
98          }
99
100         ListNode current = firstNode;
101
102         while ( current != null ) {
103            System.out.print( current.data.toString() + " " );
104            current = current.next;
105         }
106
107         System.out.print( "\n" );
108      }
109
110      public synchronized ListNode getFirstNode()
111      {
112         return firstNode;
113      }
114
115   }  // end class List
```

20.33 *(Performance of Binary Tree Sorting and Searching)* One problem with the binary tree sort is that the order in which the data is inserted affects the shape of the tree—for the same collection of data, different orderings can yield binary trees of dramatically different shapes. The performance of the binary tree sorting and searching algorithms is sensitive to the shape of the binary tree. What shape would a binary tree have if its data were inserted in increasing order? in decreasing order? What shape should the tree have to achieve maximal searching performance?

20.34 *(Indexed Lists)* As presented in the text, linked lists must be searched sequentially. For large lists, this can result in poor performance. A common technique for improving list-searching performance is to create and maintain an index to the list. An index is a set of references to key places in the list. For example, an application that searches a large list of names could improve performance by creating an index with 26 entries—one for each letter of the alphabet. A search operation for a last name beginning with 'Y' would then first search the index to determine where the 'Y' entries begin, then "jump into" the list at that point and search linearly until the desired name is found. This would be much faster than searching the linked list from the beginning. Use the `List` class of Fig. 20.3 as the basis of an `IndexedList` class.

Write a program that demonstrates the operation of indexed lists. Be sure to include methods `insertInIndexedList`, `searchIndexedList` and `deleteFromIndexedList`.

ANS:

```
1  // Exercise 20.34 Solution: IndexedList.java
2  // An implementation of an indexed list.
3  import com.deitel.jhtp5.ch20.*;
4
5  public class IndexedList {
6     private List[] indexer;
7
8     public IndexedList() {
9        indexer = new List[ 26 ];
10       for (int i = 0; i < 26; i++ )
11       {
12          indexer[ i ] = new List();
13       }
14    }
15
16    public void print()
17    {
18       for ( int i = 0; i < 26; i++ ) {
19          System.out.print( ( char )( i + 'a' ) + ": " );
20          indexer[ i ].print();
21       }
22    }
23
24    private int getIndex( String name )
25    {
26       return name.charAt( 0 ) - 'a';
27    }
28
29    public void insertInIndexedList( String name )
30    {
31       int index = getIndex( name ); // find the index for the string
32
33       // get the list from the index and the first node
34       List start = indexer[ index ];
```

```
35          ListNode front = start.getFirstNode();
36
37          // if the list is empty
38          if ( front == null )
39             start.insertAtFront( name ); // add to the front of the list
40
41          // if the list has only one element
42          else if ( front.getNext() == null )
43
44             // determine whether to add to the front or to the back
45             if ( name.compareTo( ( String )front.getObject() ) > 0 )
46                start.insertAtBack( name );
47             else
48                start.insertAtFront( name );
49
50          // list has more than one element
51          else {
52             // find the place in the list to insert the new element
53             while ( front.getNext().getNext() != null &&
54                name.compareTo( front.getNext().getObject() ) > 0 )
55
56                front = front.getNext();
57
58             // if we hit the end
59             if ( front.getNext().getNext() == null &&
60                name.compareTo( front.getNext().getObject() ) > 0 )
61
62                start.insertAtBack( name ); // add to the back
63             else {
64                // insert in the middle of the list
65                ListNode insert = new ListNode( name );
66                insert.setNext( front.getNext() );
67                front.setNext( insert );
68             }
69
70          } // end else
71
72       } // end insertInIndexedList
73
74       public boolean searchIndexedList( String name )
75       {
76          int index = getIndex( name ); // find the index into the list
77
78          List start = indexer[ index ]; // get the list from the index
79
80          ListNode front = start.getFirstNode(); // get the start of the list
81
82          // if the list is empty
83          if ( front == null )
84             return false;
85
86          // find the spot in the list
87          while ( front.getNext() != null &&
88             !name.equals( front.getObject() ) )
```

```
89
90             front = front.getNext();
91
92          // if we found the item
93          if ( name.equals( front.getObject() ) )
94             return true;
95
96          return false;
97
98       } // end method searchIndexedList
99
100      public void deleteFromIndexedList( String name )
101      {
102         int index = getIndex( name ); // find the index into the list
103
104         List start = indexer[ index ]; // get the list from the index
105
106         ListNode front = start.getFirstNode(); // get the start of the list
107
108         // if the list is empty
109         if ( front == null )
110            return;
111
112         // if the first item is the node to delete
113         if ( name.equals( front.getObject() ) ) {
114            start.removeFromFront();
115            return;
116         }
117
118         // if the list has only one item (which is not the item to delete)
119         if ( front.getNext() == null )
120            return;
121
122         // find the spot in the list to remove
123         while ( front.getNext().getNext() != null &&
124            !name.equals( front.getNext().getObject() ) )
125            front = front.getNext();
126
127         // if we hit the end of the list
128         if ( !name.equals( front.getNext().getObject() ) )
129            return;
130
131         front.setNext( front.getNext().getNext() ); // remove the item
132
133      } // end method deleteFromIndexedList
134
135   } // end class IndexedList
```

ANS:

```
1  // Exercise 20.34 Solution: IndexedTest.java
2  // Program tests an indexed list
3  public class IndexedTest {
```

```
4
5     public static void main( String args[] )
6     {
7        String[] strings = { "ah", "acme", "apple", "fish", "orange",
8           "pickle", "bread", "steak", "green", "tree", "query", "yellow",
9           "frog", "leaf", "potato", "chicken", "duck", "jelly", "jam",
10          "butter", "margarine", "hello", "icecream", "bagel", "elephant",
11          "corn", "knife", "liver", "onions", "carrots", "broccoli",
12          "salmon", "blue", "black", "red", "silver", "grey", "white",
13          "brown", "purple", "turkey", "eagle", "hawk", "falcon",
14          "ostrich", "horse", "pig", "zebra", "lion", "tiger" };
15
16       IndexedList list = new IndexedList();
17
18       // create list
19       for ( int a = 0; a < 50; a++ ) {
20          list.insertInIndexedList( strings[ a ] );
21       }
22
23       // print list before insertion
24       System.out.println( "Indexed list:" );
25       list.print();
26       System.out.print( "\n" );
27
28       String insert = "bear"; // insert new object
29       list.insertInIndexedList( insert );
30       System.out.println( insert + " inserted.\n" );
31
32       // print list after insertion
33       System.out.println( "After insertion:" );
34       list.print();
35       System.out.print( "\n" );
36
37       // try to find the string "green"
38       if ( list.searchIndexedList( "green" ) )
39          System.out.println( "green found in list.\n" );
40       else
41          System.out.println( "green not found in list.\n" );
42
43       // try to find the string "grass"
44       if ( list.searchIndexedList( "grass" ) )
45          System.out.println( "grass found in list.\n" );
46       else
47          System.out.println( "grass not found in list.\n" );
48
49       // delete three items from the list
50       list.deleteFromIndexedList( "tiger" );
51       list.deleteFromIndexedList( "leopard" );
52       list.deleteFromIndexedList( "lion" );
53
54       System.out.println( "List after deletions:" );
55       list.print();
56    }
57
```

```
58   } // end class IndexedTest
```

```
Indexed list:
a: The list is: acme ah apple
b: The list is: bread bagel black blue broccoli brown butter
c: The list is: chicken carrots corn
d: The list is: duck
e: The list is: eagle elephant
f: The list is: fish falcon frog
g: The list is: green grey
h: The list is: hawk hello horse
i: The list is: icecream
j: The list is: jam jelly
k: The list is: knife
l: The list is: leaf lion liver
m: The list is: margarine
n: Empty list
o: The list is: onions orange ostrich
p: The list is: pickle pig potato purple
q: The list is: query
r: The list is: red
s: The list is: salmon silver steak
t: The list is: tree tiger turkey
u: Empty list
v: Empty list
w: The list is: white
x: Empty list
y: The list is: yellow
z: The list is: zebra

bear inserted.

After insertion:
a: The list is: acme ah apple
b: The list is: bread bagel bear black blue broccoli brown butter
c: The list is: chicken carrots corn
d: The list is: duck
e: The list is: eagle elephant
f: The list is: fish falcon frog
g: The list is: green grey
h: The list is: hawk hello horse
i: The list is: icecream
j: The list is: jam jelly
k: The list is: knife
l: The list is: leaf lion liver
m: The list is: margarine
n: Empty list
o: The list is: onions orange ostrich
p: The list is: pickle pig potato purple
q: The list is: query
r: The list is: red
s: The list is: salmon silver steak
t: The list is: tree tiger turkey
u: Empty list
```

```
v: Empty list
w: The list is: white
x: Empty list
y: The list is: yellow
z: The list is: zebra

green found in list.

grass not found in list.

List after deletions:
a: The list is: acme ah apple
b: The list is: bread bagel bear black blue broccoli brown butter
c: The list is: chicken carrots corn
d: The list is: duck
e: The list is: eagle elephant
f: The list is: fish falcon frog
g: The list is: green grey
h: The list is: hawk hello horse
i: The list is: icecream
j: The list is: jam jelly
k: The list is: knife
l: The list is: leaf liver
m: The list is: margarine
n: Empty list
o: The list is: onions orange ostrich
p: The list is: pickle pig potato purple
q: The list is: query
r: The list is: red
s: The list is: salmon silver steak
t: The list is: tree turkey
u: Empty list
v: Empty list
w: The list is: white
x: Empty list
y: The list is: yellow
z: The list is: zebra
```

20.35 In Section 20.5, we created a stack class from class `List` with inheritance (Fig. 20.10) and with composition (Fig. 20.12). In Section 20.6 we created a queue class from class `List` with composition (Fig. 20.13). Create a queue class by inheriting from class `List`. What are the differences between this class and the one we created with composition?

ANS:

```
1  // Exercise 20.35 Solution: QueueInheritance.java
2  // Class QueueInheritance extends class List.
3  package com.deitel.jhtp5.ch20;
4
5  public class QueueInheritance extends List {
6
```

```
7     // construct queue
8     public QueueInheritance()
9     {
10       super( "queue" );
11    }
12
13    // add object to queue
14    public synchronized void enqueue( Object object )
15    {
16       insertAtBack( object );
17    }
18
19    // remove object from queue
20    public synchronized Object dequeue() throws EmptyListException
21    {
22       return removeFromFront();
23    }
24
25 } // end class QueueInheritance
```

ANS:

```
1  // Exercise 20.35 Solution: QueueInheritanceTest.java
2  // Class QueueInheritanceTest.
3  import com.deitel.jhtp5.ch20.QueueInheritance;
4  import com.deitel.jhtp5.ch20.EmptyListException;
5
6  public class QueueInheritanceTest {
7
8     public static void main( String args[] )
9     {
10       QueueInheritance queue = new QueueInheritance();
11
12       // create objects to store in queue
13       Boolean bool = Boolean.TRUE;
14       Character character = new Character( '$' );
15       Integer integer = new Integer( 34567 );
16       String string = "hello";
17
18       // use enqueue method
19       queue.enqueue( bool );
20       queue.print();
21       queue.enqueue( character );
22       queue.print();
23       queue.enqueue( integer );
24       queue.print();
25       queue.enqueue( string );
26       queue.print();
27
28       // remove objects from queue
29       try {
30          Object removedObject = null;
31
```

```
32            while ( true ) {
33               removedObject = queue.dequeue(); // use dequeue method
34               System.out.println( removedObject.toString() + " dequeued" );
35               queue.print();
36            }
37         }
38
39         // process exception if queue is empty when item removed
40         catch ( EmptyListException emptyListException ) {
41            emptyListException.printStackTrace();
42         }
43      }
44
45   } // end class QueueInheritanceTest
```

```
The queue is: true

The queue is: true $

The queue is: true $ 34567

The queue is: true $ 34567 hello

true dequeued
The queue is: $ 34567 hello

$ dequeued
The queue is: 34567 hello

34567 dequeued
The queue is: hello

hello dequeued
Empty queue
com.deitel.jhtp5.ch20.EmptyListException: The queue is empty
        at com.deitel.jhtp5.ch20.List.removeFromFront(List.java:56)
        at com.deitel.jhtp5.ch20.QueueInheritance.dequeue(QueueInheritance.ja-
va:22)
        at QueueInheritanceTest.main(QueueInheritanceTest.java:33)
```

21

Java Utilities Package and Bit Manipulation

Objectives

- To understand containers, such as classes `Vector` and `Stack`, and the `Enumeration` interface.
- To be able to use `Hashtable` objects.
- To be able to use persistent hash tables manipulated with objects of class `Properties`.
- To use bit manipulation to process the individual bits in integer data.
- To be able to use `BitSet` objects.

Nothing can have value without being an object of utility.
Karl Marx

O! many a shaft at sent
Finds mark the archer little meant!
Sir Walter Scott

There was the Door to which I found no Key;
There was the Veil through which I might not see.
Edward Fitzgerald

"It's a poor sort of memory that only works backwards," the Queen remarked.
Lewis Carroll [Charles Lutwidge Dodgson]

Not by age but by capacity is wisdom acquired.
Titus Maccius Plautus

SELF-REVIEW EXERCISES

21.1 Fill in the blanks in each of the following statements:

a) Java class _____ provides the capabilities of array-like data structures that can resize themselves dynamically.

ANS: `Vector`

b) If you do not specify a capacity increment, the system will _____ the size of the `Vector` each time additional capacity is needed.

ANS: double

c) Bits in the result of an expression using operator _____ are set to `1` if the corresponding bits in each operand are set to `1`. Otherwise, the bits are set to zero.

ANS: `&`

d) Bits in the result of an expression using operator _____ are set to `1` if at least one of the corresponding bits in either operand is set to `1`. Otherwise, the bits are set to `0`.

ANS: `|`

e) Bits in the result of an expression using operator _____ are set to 1 if exactly one of the corresponding bits in either operand is set to `1`. Otherwise, the bits are set to `0`.

ANS: `^`

f) The bitwise AND operator (`&`) is often used to _____ bits, that is, to select certain bits from a bit string while setting others to `0`.

ANS: mask

g) The _____ operator is used to shift the bits of a value to the left.

ANS: `<<`

h) The _____ operator shifts the bits of a value to the right with sign extension, and the _____ operator shifts the bits of a value to the right with zero extension.

ANS: `>>`, `>>>`

21.2 Determine whether each of the given statements is *true* or *false*. If *false*, explain why.

a) Values of primitive types may be stored directly in a `Vector`.

ANS: False; a `Vector` stores only `Objects`. A program must use the type-wrapper classes (`Byte`, `Short`, `Integer`, `Long`, `Float`, `Double`, `Boolean` and `Character`) from package `java.lang` to create `Objects` containing the primitive type values.

b) With hashing, as the load factor increases, the chance of collisions decreases.

ANS: False; as the load factor increases, there are fewer available slots relative to the total number of slots, so the chance of selecting an occupied slot (a collision) with a hashing operation increases.

21.3 Under what circumstances is an `EmptyStackException` thrown?

ANS: When a program calls `pop` or `peek` on an empty `Stack`, an `EmptyStackException` occurs.

EXERCISES

21.4 Define each of the following terms in the context of hashing:

a) key

ANS: Value used to determine the hash table cell where the data is stored.

b) collision

ANS: A situation where two keys hash into the same cell.

c) hashing transformation

ANS: A high-speed scheme for converting a application key into a table cell.

d) load factor

ANS: The ratio of the number of occupied cells in the hash table to the size of the hash table.

e) space/time trade-off
ANS: When the load factor is increased, the result is better memory utilization. However, the program runs slower due to increased hashing collisions.
f) `Hashtable` class
ANS: Java utilities package class that enables programmers to use hashing.
g) capacity of a `Hashtable`
ANS: The number of cells in a hash table.

21.5 Explain briefly the operation of each of the following methods of class `Vector`:
a) `add`
ANS: Adds one element to the end of the vector.
b) `insertElementAt`
ANS: Inserts one element at the specified position.
c) `set`
ANS: Sets the element at the specified position.
d) `remove`
ANS: Removes the first occurance of an element from the vector.
e) `removeAllElements`
ANS: Removes all vector elements.
f) `removeElementAt`
ANS: Removes the element at the specified position.
g) `firstElement`
ANS: Returns a reference to the first element in the vector.
h) `lastElement`
ANS: Returns a reference to the last element in the vector.
i) `isEmpty`
ANS: Determines whether or not a vector is empty.
j) `contains`
ANS: Determines if a vector contains a specified search key.
k) `indexOf`
ANS: Returns the position of the first occurance of a specified object.
l) `size`
ANS: The current number of elements in the vector.
m) `capacity`
ANS: The number of elements available for storage (used and unused).

21.6 Explain why inserting additional elements into a `Vector` object whose current size is less than its capacity is a relatively fast operation and why inserting additional elements into a `Vector` object whose current size is at capacity is a relatively slow operation.
ANS: A vector whose current size is less than its capacity has memory available. Insertions are fast because new memory does not need to be allocated. A vector that is at its capacity must have its memory reallocated and the existing values copied into it.

21.7 Explain the use of the `Enumeration` interface with objects of class `Vector`.
ANS: The `Enumeration` interface provides two methods, `hasMoreElements` and `nextElement`, which allow the vector to be walked through one element at a time.

21.8 By extending class `Vector`, Java's designers were able to create class `Stack` quickly. What are the negative aspects of this use of inheritance, particularly for class `Stack`?
ANS: Operations can be performed on `Stack` objects that are not normally allowed, which can lead to corruption of the stack.

21.9 Explain briefly the operation of each of the following methods of class `Hashtable`:
a) `put`

ANS: Adds a key/value pair into the table.

b) `get`

ANS: Locate the value associated with the specified key.

c) `isEmpty`

ANS: Returns a `boolean` value indicating whether or not the hash table is empty.

d) `containsKey`

ANS: Determine whether specified key is in the hash table.

e) `contains`

ANS: Determine whether specified `Object` is in the hash table.

f) `keys`

ANS: Return an `Enumeration` of the keys in the hash table.

21.10 Use a `Hashtable` to create a reusable class for choosing one of the 13 predefined colors in class `Color`. The names of the colors should be used as keys, and the predefined `Color` objects should be used as values. Place this class in a package that can be imported into any Java program. Use your new class in an application that allows the user to select a color and draw a shape in that color.

ANS:

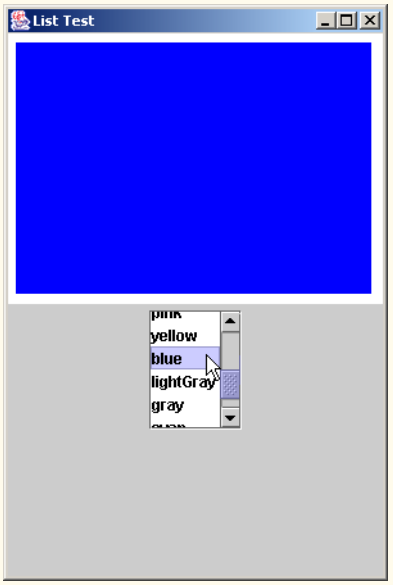
```
1  // Exercise 21.10 Solution: ColorChooser.java
2  // Class that uses a hashtable to store color name-object pairs.
3  package com.deitel.jhtp5.ch21;
4
5  import java.awt.Color;
6  import java.util.*;
7
8  public class ColorChooser {
9     private Hashtable table;
10
11    public ColorChooser() {
12       table = new Hashtable();
13
14       // add the 13 colors to the hashtable
15       table.put( "black", Color.black );
16       table.put( "blue", Color.blue );
17       table.put( "cyan", Color.cyan );
18       table.put( "darkGray", Color.darkGray );
19       table.put( "gray", Color.gray );
20       table.put( "green", Color.green );
21       table.put( "lightGray", Color.lightGray );
22       table.put( "magenta", Color.magenta );
23       table.put( "orange", Color.orange );
24       table.put( "pink", Color.pink );
25       table.put( "red", Color.red );
26       table.put( "white", Color.white );
27       table.put( "yellow", Color.yellow );
28    }
29
30    // return the selected color
31    public Color getColor( String name )
32    {
33       return ( Color )table.get( name );
34    }
35
```

```
36     // return all the color names
37     public Set getKeySet()
38     {
39        return table.keySet();
40     }
41
42  } // end class ColorChooser
```

ANS:

```
1   // Exercise 21.10 Solution: ColorTest.java
2   // Class that uses a ColorChooser to pick the color to draw a square.
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6   import javax.swing.event.*;
7   import com.deitel.jhtp5.ch21.ColorChooser;
8
9   public class ColorTest extends JFrame {
10     private JList colorList;
11     private Container container;
12     private ColorChooser chooser;
13     private JPanel drawPanel;
14
15     ColorTest()
16     {
17        super( "List Test" );
18
19        // get content pane and set its layout
20        container = getContentPane();
21        container.setLayout( new GridLayout( 2, 1 ) );
22
23        chooser = new ColorChooser();
24
25        // create a list with items in colorNames array
26        colorList = new JList( chooser.getKeySet().toArray() );
27        colorList.setVisibleRowCount( 5 );
28        colorList.setSelectedIndex( 0 );
29
30        // do not allow multiple selections
31        colorList.setSelectionMode( ListSelectionModel.SINGLE_SELECTION );
32
33        JPanel choosePanel = new JPanel();
34
35        // add a JScrollPane containing JList to content pane
36        choosePanel.add( new JScrollPane( colorList ) );
37
38        // set up event handler
39        colorList.addListSelectionListener(
40
41           new ListSelectionListener() { // anonymous inner class
42
```

```
43              // handle list selection events
44              public void valueChanged( ListSelectionEvent event )
45              {
46                 repaint();
47              }
48
49           }  // end anonymous inner class
50
51        ); // end call to addListSelectionListener
52
53        drawPanel = new JPanel(); // create a drawing panel
54        drawPanel.setBackground( Color.white );
55
56        // add the panels to the window
57        container.add( drawPanel );
58        container.add( choosePanel );
59
60        setSize( 300, 450 );
61        setVisible( true );
62
63     } // end constructor
64
65     public void paint( Graphics g ) {
66        super.paint(g);
67
68        // obtain the selected value
69        Object selected = colorList.getSelectedValue();
70
71        // set the color
72        g.setColor( chooser.getColor( ( String )selected ) );
73
74        // draw the rectangle
75        Rectangle bounds = drawPanel.getVisibleRect();
76        g.fillRect( bounds.x + 10, bounds.y + 30, bounds.width - 15,
77           bounds.height - 15 );
78     }
79
80     public static void main( String args[] )
81     {
82        ColorTest application = new ColorTest();
83        application.setDefaultCloseOperation( EXIT_ON_CLOSE );
84     }
85
86  } // end class ColorTest
```



21.11 Modify your solution to Exercise 14.17—the polymorphic painting program—to store every shape the user draws in a `Vector` of `MyShape` objects. For the purpose of this exercise, create your own `Vector` subclass called `ShapeVector` that manipulates only `MyShape` objects. Provide the following capabilities in your program:

a) Allow the user of the program to remove any number of shapes from the `Vector` by clicking an **Undo** button.

b) Allow the user to select any shape on the screen and move it to a new location. This operation requires the addition of a new method to the `MyShape` hierarchy. The method's header should be

```
public boolean isInside()
```

This method should be overridden for each subclass of `MyShape` in order to determine whether the coordinates where the user pressed the mouse button are inside the shape.

c) Allow the user to select any shape on the screen and change its color.

d) Allow the user to select any shape on the screen that can be filled or unfilled and change its fill state.

21.12 What does it mean when we state that a `Properties` object is a "persistent" `Hashtable` object? Explain the operation of each of the following methods of the `Properties` class:

a) `load`

ANS: Read the contents from a specified `InputStream`.

b) `store`

ANS: Writes the contents to a specified `OutputStream`.

c) `getProperty`

ANS: Returns the value associated with a key.

d) `propertyNames`

ANS: Returns an `Enumeration`.

e) `list`

ANS: Displays the contents of a `Properties` object.

21.13 Why might you want to use objects of class `BitSet`? Explain the operation of each of the following methods of class `BitSet`:

a) `set`
ANS: Sets the specified bit to on.
b) `clear`
ANS: Sets the specified bits to off.
c) `get`
ANS: Determines whether or not the specified bit is on.
d) `and`
ANS: Performs a bitwise logical AND.
e) `or`
ANS: Performs a bitwise logical OR.
f) `xor`
ANS: Performs a bitwise logical XOR.
g) `size`
ANS: Returns the size of the bit set.
h) `equals`
ANS: Compares two bit sets for equality.
i) `toString`
ANS: Converts a bit set to a `String`.

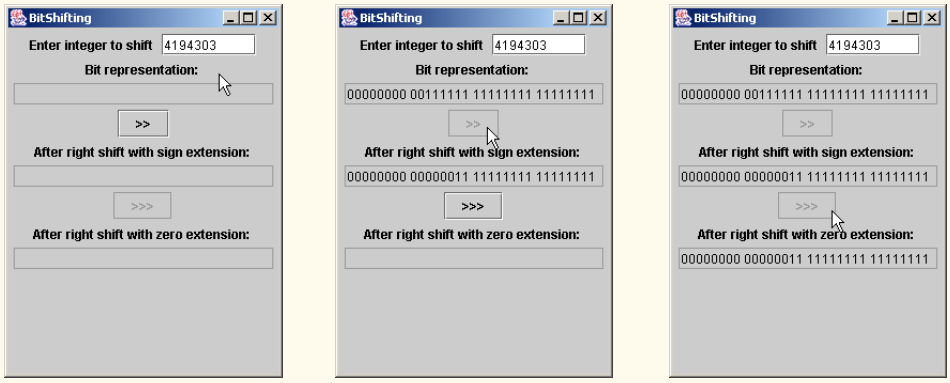
21.14 Write a program that right shifts an integer variable four bits to the right with sign extension, then shifts the same integer variable four bits to the right with zero extension. The program should print the integer in bits before and after each shift operation. Run your program once with a positive integer and once with a negative integer.

ANS:

```
1  // Exercise 21.14 Solution: BitShift.java
2  // Using the bitwise shift operators
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class BitShift extends JFrame
9  {
10    private JLabel prompt, originalBitLabel, signBitLabel, zeroBitLabel;
11    private JTextField inputValue, originalBitField,
12       signBitField, zeroBitField;
13    private JButton rightSign, rightZero;
14    private int number;
15
16    public BitShift()
17    {
18       super( "BitShifting" );
19       prompt = new JLabel( "Enter integer to shift " );
20       inputValue = new JTextField( 8 );
21       originalBitLabel = new JLabel( "Bit representation: " );
22       originalBitField = new JTextField( 22 );
23       originalBitField.setEditable( false );
24
25       // components related to the right shift w/ sign extension
26       rightSign = new JButton( ">>" );
```

```
27          rightSign.addActionListener(
28
29             new ActionListener() { // anonymous inner class
30
31                public void actionPerformed( ActionEvent event )
32                {
33                   // shift bits
34                   number = Integer.parseInt( inputValue.getText() );
35                   originalBitField.setText( getBits( number ) );
36
37                   int number2 = number >> 4;
38                   signBitField.setText( getBits( number2 ) );
39
40                   rightSign.setEnabled( false );
41                   rightZero.setEnabled( true );
42                }
43
44             } // end anonymous inner class
45
46          ); // end call to addActionListener
47
48          signBitLabel = new JLabel(
49             "After right shift with sign extension: " );
50          signBitField = new JTextField( 22 );
51          signBitField.setEditable( false );
52
53          // components related to the right shift w/ zero extension
54          rightZero = new JButton( ">>>" );
55          rightZero.addActionListener(
56
57             new ActionListener() { // anonymous inner class
58
59                public void actionPerformed( ActionEvent event )
60                {
61                   // shift bits
62                   int number2 = number >>> 4;
63                   zeroBitField.setText( getBits( number2 ) );
64                   rightZero.setEnabled( false );
65                }
66
67             } // end anonymous inner class
68
69          ); // end call to addActionListener
70
71          rightZero.setEnabled( false );
72          zeroBitLabel = new JLabel(
73                "After right shift with zero extension: " );
74          zeroBitField = new JTextField( 22 );
75          zeroBitField.setEditable( false );
76
77          // add components to container
78          Container container = getContentPane();
79          container.setLayout( new FlowLayout() );
80          container.add( prompt );
```

```
81          container.add( inputValue );
82          container.add( originalBitLabel );
83          container.add( originalBitField );
84          container.add( rightSign );
85          container.add( signBitLabel );
86          container.add( signBitField );
87          container.add( rightZero );
88          container.add( zeroBitLabel );
89          container.add( zeroBitField );
90
91          setSize(260, 350);
92          setVisible( true );
93       }
94
95       // return String containing the bit representation of the int
96       public String getBits( int value )
97       {
98          int displayMask = 1 << 31;
99          StringBuffer buffer = new StringBuffer( 35 );
100
101         for ( int c = 1; c <= 32; c++ ) {
102
103            // use AND operator and mask to get
104            // binary representation of value
105            buffer.append( ( value & displayMask ) == 0 ? '0' : '1' );
106            value <<= 1;
107
108            // spacing
109            if ( c % 8 == 0 )
110               buffer.append( ' ' );
111         }
112
113         return buffer.toString();
114      }
115
116      public static void main( String args[] )
117      {
118         BitShift application = new BitShift();
119         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
120      }
121
122   }  // end class BitShift
```



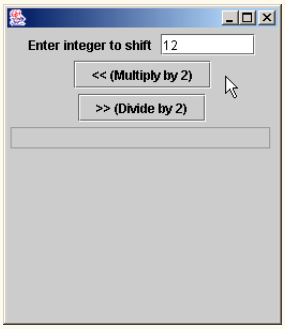
21.15 Show how shifting an integer left by one can be used to perform multiplication by two and how shifting an integer right by one can be used to perform division by two. Be careful to consider issues related to the sign of an integer.

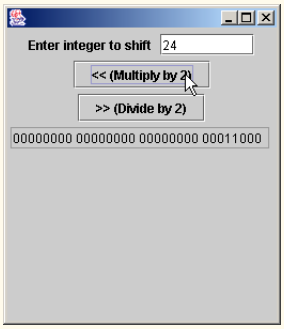
ANS:

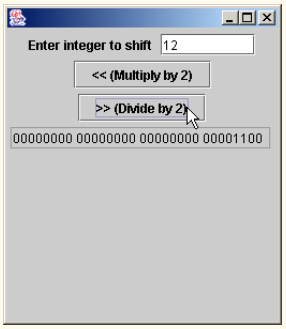
```
1  // Exercise 21.15 Solution: BitShift2.java
2  // Using the bitwise shift operators
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class BitShift2 extends JFrame {
9     private JLabel prompt, status;
10    private JTextField value, bits;
11    private JButton multiply, divide;
12
13    public BitShift2()
14    {
15       prompt = new JLabel( "Enter integer to shift " );
16       status = new JLabel( "" );
17
18       value = new JTextField( 8 );
19       bits = new JTextField( 22 );
20       bits.setEditable( false );
21
22       multiply = new JButton( "<< (Multiply by 2)" );
23       divide = new JButton( ">> (Divide by 2)" );
24       multiply.addActionListener(
25
26          new ActionListener() { // anonymous inner class
27
28             public void actionPerformed( ActionEvent event )
29             {
30                // bit shift
31                int number = Integer.parseInt( value.getText() );
32                number <<= 1;
33
```

```
34                // display results
35                value.setText( Integer.toString( number ) );
36                bits.setText( getBits( number ) );
37
38             }
39
40          } // end anonymous inner class
41
42       ); // end call to addActionListener
43
44       divide.addActionListener(
45
46          new ActionListener() { // anonymous inner class
47
48             public void actionPerformed( ActionEvent event )
49             {
50                // bit shift
51                int number = Integer.parseInt( value.getText() );
52                number >>= 1;
53
54                // display results
55                value.setText( Integer.toString( number ) );
56                bits.setText( getBits( number ) );
57             }
58
59          } // end anonymous inner class
60
61       ); // end call to addActionListener
62
63       Container container = getContentPane();
64       container.setLayout( new FlowLayout() );
65       container.add( prompt );
66       container.add( value );
67       container.add( multiply );
68       container.add( divide );
69       container.add( bits );
70       container.add( status );
71
72       setSize( 260, 300 );
73       setVisible( true );
74    }
75
76    // return String containing the bit representation of the int
77    public String getBits( int value )
78    {
79       int displayMask = 1 << 31;
80       StringBuffer buffer = new StringBuffer( 35 );
81
82       for ( int count = 1; count <= 32; count++ ) {
83          buffer.append( ( value & displayMask ) == 0 ? '0' : '1' );
84          value <<= 1;
85
86          if ( count % 8 == 0 )
87             buffer.append( ' ' );
88       }
```

```
89
90          return buffer.toString();
91       }
92
93       public static void main( String args[] )
94       {
95          BitShift2 application = new BitShift2();
96          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
97       }
98
99    }  // end class BitShift2
```







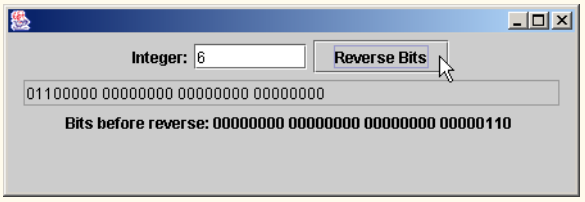
21.16 Write a program that reverses the order of the bits in an integer value. The program should input the value from the user and call method `reverseBits` to print the bits in reverse order. Print the value in bits both before and after the bits are reversed to confirm that the bits are reversed properly. You might want to implement both a recursive and an iterative solution.

ANS:

```
1  // Exercise 21.16 Solution: Reverse.java
2  // Program reverses bits.
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6  import javax.swing.event.*;
7
8  public class Reverse extends JFrame {
9     private JLabel prompt, status;
10    private JTextField valueField, bitsField;
11    private JButton backwardsButton;
12
13    public Reverse()
14    {
15       prompt = new JLabel( "Integer:" );
16       status = new JLabel( "" );
17       valueField = new JTextField( 8 );
18       bitsField = new JTextField( 38 );
19       bitsField.setEditable( false );
20       backwardsButton = new JButton( "Reverse Bits" );
```

```
21         backwardsButton.addActionListener(
22
23            new ActionListener() { // anonymous inner class
24
25               // get bit representations
26               public void actionPerformed( ActionEvent event )
27               {
28                  int number = Integer.parseInt( valueField.getText() );
29
30                  status.setText(
31                     "Bits before reverse: " + getBits( number ) );
32                  bitsField.setText( getBits( reverseBits( number ) ) );
33               }
34
35            } // end anonymous inner class
36
37         ); // end call to addActionListener
38
39         Container container = getContentPane();
40         container.setLayout( new FlowLayout() );
41         container.add( prompt );
42         container.add( valueField );
43         container.add( backwardsButton );
44         container.add( bitsField );
45         container.add( status );
46
47         setSize( 450, 150 );
48         setVisible( true );
49      }
50
51      // reverse bits
52      public int reverseBits( int value )
53      {
54         int mask = 1, temp = 0;
55
56         for ( int x = 1; x <= 32; x++ ) {
57            temp <<= 1;
58            temp |= ( value & mask );
59            value >>= 1;
60         }
61
62         return temp;
63      }
64
65      // get bit representation of value
66      public String getBits( int value )
67      {
68         int displayMask = 1 << 31;
69         StringBuffer buffer = new StringBuffer( 32 );
70
71         for ( int c = 1; c <= 32; c++ ) {
72            buffer.append( ( value & displayMask ) == 0 ? '0' : '1' );
73            value <<= 1;
74
```

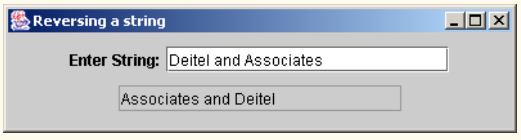
```
75             if ( c % 8 == 0 )
76                buffer.append( ' ' );
77          }
78
79          return buffer.toString();
80       }
81
82       public static void main( String args[] )
83       {
84          Reverse application = new Reverse();
85          application.setDefaultCloseOperation( EXIT_ON_CLOSE );
86       }
87
88    }  // end class Reverse
```



21.17 Modify your solution to Exercise 20.10 to use class `Stack`.
ANS:

```
1  // Exercise 21.17 Solution: ReverseText.java
2  // Program prints the words of a line in reverse.
3  import java.awt.*;
4  import java.util.*;
5  import java.awt.event.*;
6  import javax.swing.*;
7
8  public class ReverseText extends JFrame {
9     private JTextField inputField, outputField;
10    private JLabel prompt;
11    private JPanel panel;
12
13    public ReverseText()
14    {
15       super( "Reversing a string" );
16
17       final Stack stack = new Stack();
18
19       // create GUI components
20       prompt = new JLabel( "Enter String:" );
21       inputField = new JTextField( 20 );
22
23       inputField.addActionListener(
24
25          new ActionListener() { // anonymous inner class
26
```

```
27             public void actionPerformed( ActionEvent event )
28             {
29                // take each word from tokenizer and push on stack
30                String text = inputField.getText();
31                StringTokenizer tokenizer = new StringTokenizer( text );
32                StringBuffer buffer = new StringBuffer( text.length() );
33
34                while ( tokenizer.hasMoreTokens() )
35                   stack.push( tokenizer.nextToken() );
36
37                // build reverse string by popping words from stack.
38                Object removedObject = null;
39
40                   while ( !stack.isEmpty() ) {
41                      removedObject = stack.pop();
42                      buffer.append( removedObject.toString() + " " );
43                   }
44
45                outputField.setText( buffer.toString() );
46             }
47
48          }  // end anonymous inner class
49
50       ); // end call to addActionListener
51
52       outputField = new JTextField( 20 );
53       outputField.setEditable( false );
54
55       // set up layout and add components
56       Container container = getContentPane();
57       container.setLayout( new FlowLayout() );
58       JPanel panel = new JPanel();
59       panel.add( prompt );
60       panel.add( inputField );
61       container.add( panel );
62       container.add( outputField );
63
64       setSize( 400, 100 );
65       setVisible( true );
66    }
67
68    public static void main( String args[] )
69    {
70       ReverseText application = new ReverseText();
71       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
72    }
73
74 }  // end class ReverseText
```

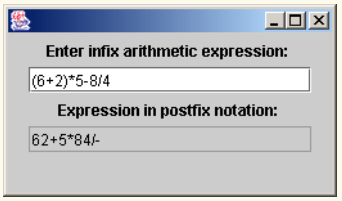


21.18 Modify your solution to Exercise 20.12 to use class `Stack`.
ANS:

```
// Exercise 21.18 Solution: InfixToPostfixConverter.java
// Program converts infix arithmetic expression to a postfix
// expression. Assume a valid expression is entered.
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;

public class InfixToPostfixConverter extends JFrame
{
   private JLabel prompt, postfixLabel;
   private JTextField infixField, postfixField;

   public InfixToPostfixConverter()
   {
      prompt = new JLabel( "Enter infix arithmetic expression: " );
      infixField = new JTextField( 20 );
      infixField.addActionListener(

         new ActionListener() { // anonymous inner class

            public void actionPerformed( ActionEvent event )
            {
               // call convertToPostfix
               StringBuffer input = new StringBuffer(
                  infixField.getText() );
               postfixField.setEnabled( true );
               postfixField.setText( convertToPostfix(
                  input ).toString() );
               postfixField.setEditable( false );
            }

         } // end anonymous inner class

      ); // end call to addActionListener

      postfixLabel = new JLabel( "Expression in postfix notation: " );
      postfixField = new JTextField( 20 );
      postfixField.setEnabled( false );

      Container container = getContentPane();
      container.setLayout( new FlowLayout() );
      container.add( prompt );
      container.add( infixField );
      container.add( postfixLabel );
      container.add( postfixField );

      setSize( 260, 150 );
      setVisible( true );
   }

```

```
52    public StringBuffer convertToPostfix( StringBuffer infix )
53    {
54       StringBuffer postfix = new StringBuffer();
55       Stack stack = new Stack();
56       stack.push( new Character( '(' ) );
57       infix.append( ")" );
58
59       int index = 0;
60
61       // convert expression
62       while( !stack.isEmpty() ) {
63          char temp = infix.charAt( index );
64
65          // digits
66          if ( Character.isDigit( temp ) )
67             postfix.append( temp );
68
69          // left parenthesis
70          else if ( temp == '(' )
71             stack.push( new Character( temp ) );
72
73          // operators
74          else if ( isOperator( temp ) ) {
75             char top = ( ( Character ) stack.peek() ).charValue();
76
77             while ( isOperator( top ) && !precedence( top, temp ) ) {
78
79                postfix.append( ( ( Character) stack.pop() ).charValue() );
80                top = ( ( Character ) stack.peek() ).charValue();
81             }
82
83             stack.push( new Character( temp ) );
84          }
85
86          // right parenthesis
87          else {
88
89             while ( ( ( Character ) stack.peek() ).charValue() != '(' )
90                postfix.append( ( (Character) stack.pop() ).charValue() );
91
92             stack.pop();
93          }
94
95          index++;
96       }
97
98       return postfix;
99
100   }  // end method convertToPostfix
101
102   // determine if c is an operator
103   public boolean isOperator( char c )
104   {
```

```
105         if ( c == '+' || c == '-' || c == '*' ||
106            c == '/' || c == '^' || c == '%' )
107            return true;
108
109         return false;
110      }
111
112      // return true if operator1 has lower precedence than operator2
113      public boolean precedence( char operator1, char operator2 )
114      {
115         if ( ( operator1 == '+' || operator1 == '-' ) &&
116               operator2 != '+' && operator2 != '-' )
117            return true;
118
119         else if ( operator1 != '^' && operator2 == '^' )
120            return true;
121
122         return false;
123      }
124
125      public static void main( String args[] )
126      {
127         InfixToPostfixConverter application = new InfixToPostfixConverter();
128         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
129      }
130
131   }  // end class InfixToPostfixConverter
```



21.19 Modify your solution to Exercise 20.13 to use class `Stack`.
ANS:

```
1  // Exercise 21.19 Solution: PostfixEvaluator.java
2  // Program evaluates a postfix expression.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class PostfixEvaluator extends JFrame {
9
10    private JLabel prompt;
11    private JTextField expressionField, resultField;
12
13    public PostfixEvaluator()
14    {
```

```
15          prompt = new JLabel( "Enter postfix expression to be evaluated:" );
16          expressionField = new JTextField( 15 );
17          expressionField.addActionListener(
18
19             new ActionListener() { // anonymous inner class
20
21                // call method evaluatePostfixExpression
22                public void actionPerformed( ActionEvent event )
23                {
24                   StringBuffer input =
25                      new StringBuffer ( expressionField.getText() );
26                   resultField.setText( "Expression equals " +
27                      evaluatePostfixExpression( input ) );
28                }
29
30             } // end anonymous inner class
31
32          ); // end call to addActionListener
33
34          resultField = new JTextField( 15 );
35          resultField.setEditable( false );
36
37          Container container = getContentPane();
38          container.setLayout( new FlowLayout() );
39
40          container.add( prompt );
41          container.add( expressionField );
42          container.add( resultField );
43
44          setSize( 350, 125 );
45          setVisible( true );
46       }
47
48       // evaluate expression
49       public int evaluatePostfixExpression( StringBuffer buffer )
50       {
51          Stack stack = new Stack();
52          buffer.append( ")" );
53          int index = 0;
54
55          char temp = buffer.charAt( index++ );
56
57          while( temp != ')' ) {
58
59             if ( Character.isDigit( temp ) )
60                stack.push( new Integer( temp - '0' ) );
61
62             else {
63                int x = ( ( Integer ) stack.pop() ).intValue();
64                int y = ( ( Integer ) stack.pop() ).intValue();
65                stack.push( new Integer( calculate( y, x, temp ) ) );
66             }
67
68             temp = buffer.charAt( index++ );
69          }
```

```
70
71          return ( ( Integer ) stack.pop() ).intValue();
72       }
73
74       // perform calculations
75       public int calculate( int value1, int value2, char operator )
76       {
77          if ( operator == '+' )
78             return ( value1 + value2 );
79
80          else if ( operator == '-' )
81             return ( value1 - value2 );
82
83          else if ( operator == '*' )
84             return ( value1 * value2 );
85
86          else if ( operator == '/' )
87             return ( value1 / value2 );
88
89          else if ( operator == '^' )
90             return ( value1 ^ value2 );
91
92          else
93             return ( value1 % value2 );
94       }
95
96       public static void main( String args[] )
97       {
98          PostfixEvaluator application = new PostfixEvaluator();
99          application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
100      }
101
102 }  // end class PostfixEvaluator
```

22

Collections

Objectives

- To understand what collections are.
- To be able to use class `Arrays` for common array manipulations
- To use the collections-framework implementations.
- To be able to use collections-framework algorithms to manipulate various collections.
- To be able to use the collections-framework interfaces to program polymorphically.
- To be able to use iterators to “walk” through the elements of a collection.
- To understand synchronization wrappers and modifiability wrappers.

I think this is the most extraordinary collection of talent, of human knowledge, that has ever been gathered together at the White House—with the possible exception of when Thomas Jefferson dined alone.
John F. Kennedy

The shapes a bright container can contain!
Theodore Roethke

Journey over all the universe in a map.
Miguel de Cervantes

It is an immutable law in business that words are words, explanations are explanations, promises are promises — but only performance is reality.
Harold S. Green

SELF-REVIEW EXERCISES

22.1 Fill in the blanks in each of the following statements:

a) `Objects` in a collection are called _____.

ANS: elements

b) An element in a `List` can be accessed by using the element's _____.

ANS: index

c) `Lists` are sometimes called _____.

ANS: sequences

d) You can use a(n) _____ to create a collection that offers only read-only access to others while allowing read–write access to yourself.

ANS: unmodifiable wrapper

e) _____ can be used to create stacks, queues, trees and deques (double-ended queues).

ANS: `LinkedLists`

22.2 Determine whether each statement is *true* or *false*. If *false*, explain why.

a) A `Set` can contain duplicate values.

ANS: False. A `Set` cannot contain duplicate values.

b) A `Map` can contain duplicate keys.

ANS: False. A `Map` cannot contain duplicate keys.

c) A `LinkedList` can contain duplicate values.

ANS: True.

d) `Collections` is an `interface`.

ANS: False. `Collections` is a `class`, and `Collection` is an `interface`.

e) `Iterators` can remove elements, while `Enumerations` cannot.

ANS: True.

EXERCISES

22.3 Define each of the following terms:

a) `Collection`

ANS: Interface `Collection` is the root interface in the collections hierarchy from which interfaces `Set` and `List` are derived.

b) `Collections`

ANS: Class `Collections` provides `static` methods that manipulate collections polymorphically. These methods implement algorithms for searching, sorting, etc.

c) `Comparator`

ANS: An object that specifies how a collections objects are ordered.

d) `List`

ANS: An interface that descibes the implemention for linked lists.

22.4 Briefly answer the following questions:

a) What is the primary difference between a `Set` and a `Map`?

ANS: `Maps` contain both the key and the value and `Sets` contian only the key.

b) Can a two-dimensional array be passed to `Arrays` method `asList`? If yes, how would an individual element be accessed?

ANS: No.

c) What must you do before adding a primitive type (e.g., `double`) to a collection?

ANS: Construct an object using the appropriate wrapper class (e.g., `Double`, `Integer`, etc.).

22.5 Explain briefly the operation of each of the following `Iterator`-related methods:

a) `iterator`

ANS: Returns an iterator for a collection.

b) `hasNext`

ANS: Determines if a collection has a next element.

c) `next`

ANS: Returns the next element in a collection.

22.6 Determine whether each statement is *true* or *false*. If *false*, explain why.

a) Elements in a `Collection` must be sorted in ascending order before a `binarySearch` may be performed.

ANS:

b) Method `first` gets the first element in a `TreeSet`.

ANS: False. The elements need only be sorted.

c) A `List` created with `Arrays` method `asList` is resizable.

ANS: False. The `List` is fixed length.

d) Class `Arrays` provides static method `sort` for sorting array elements.

ANS: True.

22.7 Rewrite method `printList` of Fig. 22.4 to use a `ListIterator`.

ANS:

```
1  // Exercise 22.7 Solution: ListTest.java
2  // Program modifies and prints Lists.
3  import java.util.*;
4
5  public class ListTest {
6     private String colors[] = { "black", "yellow", "green",
7        "blue", "violet", "silver" };
8     private String colors2[] = { "gold", "white", "brown",
9        "blue", "gray", "silver" };
10
11    // set up and manipulate LinkedList objects
12    public ListTest()
13    {
14       LinkedList link = new LinkedList();
15       LinkedList link2 = new LinkedList();
16
17       // add elements to each list
18       for ( int count = 0; count < colors.length; count++ ) {
19          link.add( colors[ count ] );
20          link2.add( colors2[ count ] );
21       }
22
23       link.addAll( link2 );         // concatenate lists
24       link2 = null;                 // release resources
25
26       printList( link );
27
28       uppercaseStrings( link );
29
30       printList( link );
31
32       System.out.print( "\nDeleting elements 4 to 6..." );
33       removeItems( link, 4, 7 );
34
```

```
35          printList( link );
36       }
37
38       // output List contents
39       public void printList( List list )
40       {
41          ListIterator iterator = list.listIterator();
42
43          System.out.println( "\nlist: " );
44
45          while ( iterator.hasNext() )
46             System.out.print( ( String )iterator.next() + " " );
47
48          System.out.println();
49       }
50
51       // locate String objects and convert to uppercase
52       public void uppercaseStrings( List list )
53       {
54          ListIterator iterator = list.listIterator();
55
56          while ( iterator.hasNext() ) {
57             Object object = iterator.next();   // get item
58
59             if ( object instanceof String )    // check for String
60                iterator.set( ( ( String ) object ).toUpperCase() );
61          }
62       }
63
64       // obtain sublist and use clear method to delete sublist items
65       public void removeItems( List list, int start, int end )
66       {
67          list.subList( start, end ).clear();   // remove items
68       }
69
70       public static void main( String args[] )
71       {
72          new ListTest();
73       }
74
75    }  // end class ListTest
```

```
list:
black yellow green blue violet silver gold white brown blue gray silver

list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER

Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER
```

22.8 Rewrite lines 14–21 in Fig. 22.4 to be more concise by using the `asList` method and the `LinkedList` constructor that takes a `Collection` argument.

ANS:

```
1  // Exercise 22.8 Solution: ListTest2.java
2  // Program modifies and prints Lists.
3  import java.util.*;
4
5  public class ListTest2 {
6     private String colors[] = { "black", "yellow", "green",
7        "blue", "violet", "silver" };
8     private String colors2[] = { "gold", "white", "brown",
9        "blue", "gray", "silver" };
10
11    // set up and manipulate LinkedList objects
12    public ListTest2()
13    {
14       LinkedList link = new LinkedList( Arrays.asList( colors ) );
15       LinkedList link2 = new LinkedList( Arrays.asList( colors2 ) );
16
17       link.addAll( link2 );         // concatenate lists
18       link2 = null;                 // release resources
19
20       printList( link );
21
22       uppercaseStrings( link );
23
24       printList( link );
25
26       System.out.print( "\nDeleting elements 4 to 6..." );
27       removeItems( link, 4, 7 );
28
29       printList( link );
30    }
31
32    // output List contents
33    public void printList( List list )
34    {
35       ListIterator iterator = list.listIterator();
36
37       System.out.println( "\nlist: " );
38
39       while ( iterator.hasNext() )
40          System.out.print( iterator.next() + " " );
41
42       System.out.println();
43    }
44
45    // locate String objects and convert to uppercase
46    public void uppercaseStrings( List list )
47    {
48       ListIterator iterator = list.listIterator();
49
```

```
50         while ( iterator.hasNext() ) {
51            Object object = iterator.next();   // get item
52
53            if ( object instanceof String )    // check for String
54               iterator.set( ( ( String ) object ).toUpperCase() );
55         }
56      }
57
58      // obtain sublist and use clear method to delete sublist items
59      public void removeItems( List list, int start, int end )
60      {
61         list.subList( start, end ).clear();   // remove items
62      }
63
64      public static void main( String args[] )
65      {
66         new ListTest2();
67      }
68
69   }  // end class ListTest2
```

```
list:
black yellow green blue violet silver gold white brown blue gray silver

list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER

Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER
```

22.9 Write a program that reads in a series of first names and stores them in a `LinkedList`. Do not store duplicate names. Allow the user to search for a first name.

ANS:

```
1  // Exercise 22.9 Solution: ListTest.java
2  // Program stores first names in LinkedList.
3  import java.awt.event.*;
4  import java.awt.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class ListTest extends JFrame {
9
10    private LinkedList nameList;
11    private JLabel nameInLabel, nameSearchLabel, resultLabel;
12    private JTextField nameInField, nameSearchField, resultField;
13
14    public ListTest()
15    {
16       super( "First Name List" );
```

```
17
18          nameInLabel = new JLabel( "Enter a name in the list:" );
19          nameInField = new JTextField( 20 );
20          nameInField.addActionListener(
21
22             new ActionListener() { // anonymous inner class
23
24                // insert name if it isn't already in the list
25                public void actionPerformed( ActionEvent event )
26                {
27                   String inputName = nameInField.getText();
28
29                   // insert name
30                   if ( foundName( inputName ) == false ) {
31                      insertName( inputName );
32                      resultField.setText( inputName + " inserted" );
33                   }
34
35                   // name already exists in list
36                   else
37                      resultField.setText( inputName + " exists in list" );
38                      nameInField.setText( "" );
39                }
40
41             } // end anonymous inner class
42
43          ); // end call to addActionListener
44
45          nameSearchLabel = new JLabel( "Enter name for search:" );
46          nameSearchField = new JTextField( 20 );
47          nameSearchField.addActionListener(
48
49             new ActionListener() { // anonymous inner class
50
51                // search for name
52                public void actionPerformed( ActionEvent event )
53                {
54                   String inputName = nameSearchField.getText();
55
56                   // name found
57                   if ( foundName( inputName ) )
58                      resultField.setText( inputName + " found in list" );
59
60                   // name not found
61                   else
62                      resultField.setText( inputName + " not found in list" );
63                      nameSearchField.setText( "" );
64                }
65
66             } // end anonymous inner class
67
68          ); // end call to addActionListener
69
70          resultLabel = new JLabel( "Result:" );
```

```
71          resultField = new JTextField( 20 );
72          resultField.setEditable( false );
73
74          Container container = getContentPane();
75          container.setLayout( new FlowLayout() );
76          container.add( nameInLabel );
77          container.add( nameInField );
78          container.add( nameSearchLabel );
79          container.add( nameSearchField );
80          container.add( resultLabel );
81          container.add( resultField );
82
83          setSize( 230, 200 );
84          setVisible( true );
85
86          nameList = new LinkedList();
87       }
88
89       // search for name
90       public boolean foundName( String name )
91       {
92          for ( int k = 0; k < nameList.size(); k++ )
93             if ( name.equals( ( String ) nameList.get( k ) ) )
94                return true;
95
96          return false;
97       }
98
99       // insert name
100      public void insertName( String name )
101      {
102         nameList.add( name );
103      }
104
105      public static void main( String args[] )
106      {
107         ListTest application = new ListTest();
108         application.setDefaultCloseOperation( EXIT_ON_CLOSE );
109      }
110
111   }  // end class ListTest
```

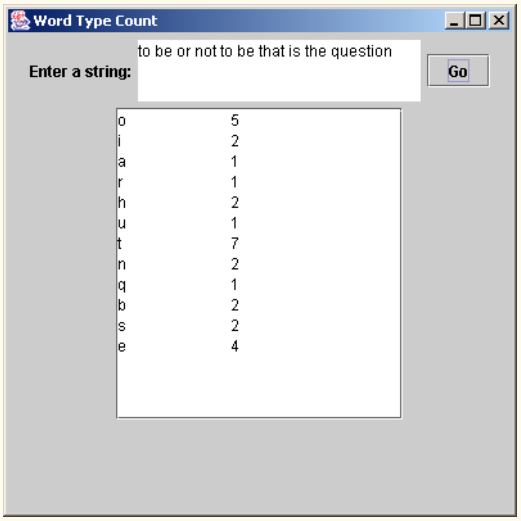
22.10 Modify the program of Fig. 22.14 to count the number of occurrences of each letter rather than of each word. For example, the string "HELLO THERE" contains two Hs, three Es, two Ls, one O, one T and one R. Display the results.

ANS:

```
1  // Exercise 22.10 Solution: MapTest.java
2  // Program stores the number of each letter in an input string.
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.*;
```

```
 6  import javax.swing.*;
 7
 8  public class MapTest extends JFrame {
 9     private JTextArea inputField;
10     private JLabel prompt;
11     private JTextArea display;
12     private JButton goButton;
13
14     private HashMap map;
15
16     public MapTest()
17     {
18        super( "Word Type Count" );
19        inputField = new JTextArea( 3, 20 );
20
21        goButton = new JButton( "Go" );
22        goButton.addActionListener(
23
24           new ActionListener() { // anonymous inner class
25
26              public void actionPerformed( ActionEvent event )
27              {
28                 map = new HashMap();
29                 createMap();
30                 display.setText( createOutput() );
31              }
32
33           }  // end anonymous inner class
34
35        ); // end call to addActionListener
36
37        prompt = new JLabel( "Enter a string:" );
38        display = new JTextArea( 15, 20 );
39        display.setEditable( false );
40
41        JScrollPane displayScrollPane = new JScrollPane( display );
42
43        // add components to GUI
44        Container container = getContentPane();
45        container.setLayout( new FlowLayout() );
46        container.add( prompt );
47        container.add( inputField );
48        container.add( goButton );
49        container.add( displayScrollPane );
50
51        setSize( 400, 400 );
52        setVisible( true );
53
54     }  // end constructor
55
56     private void createMap() {
57
58        String input = inputField.getText();
59        input.toLowerCase();
```

```
60
61          for( int i = 0; i < input.length(); i++ ) {
62             char letter = input.charAt( i );
63             if ( Character.isLetter( letter ) ) {
64                String key = String.valueOf( letter );
65
66                // if the map contains the word
67                if ( map.containsKey( key ) ) {
68
69                   // get the value
70                   Integer count = ( Integer ) map.get( key );
71
72                   // and increment it
73                   map.put( key, new Integer( count.intValue() + 1 ) );
74                }
75                else // otherwise add the word with a value of 1
76                   map.put( key, new Integer( 1 ) );
77             }
78
79           } // end while
80
81       } // end method createMap
82
83       private String createOutput() {
84
85          Set keys = map.keySet();
86          Iterator keyIterator = keys.iterator();
87          String output = "";
88
89          // iterate through the keys
90          while ( keyIterator.hasNext() ) {
91
92             Object currentKey = keyIterator.next();
93
94             // output the key-value pairs
95             output += currentKey + "\t" +
96             map.get( currentKey ) + "\n";
97          }
98
99          return output;
100
101      } // end method createOutput
102
103      public static void main( String args[] )
104      {
105         MapTest application = new MapTest();
106         application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
107      }
108
109   } // end class MapTest
```



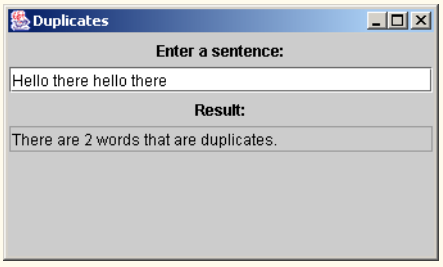
22.11 Write a program that determines and prints the number of duplicate words in a sentence. Treat uppercase and lowercase letters the same. Ignore punctuation.

ANS:

```
1  // Exercise 22.11 Solution: Duplicates.java
2  // Program stores the number of duplicate words in a sentence.
3  import java.awt.event.*;
4  import java.awt.*;
5  import java.util.*;
6  import javax.swing.*;
7
8  public class Duplicates extends JFrame
9  {
10    private String sentence;
11    private JLabel SentenceInLabel, resultLabel;
12    private JTextField SentenceInField, resultField;
13
14    public Duplicates()
15    {
16       super( "Duplicates" );
17
18       SentenceInLabel = new JLabel( "Enter a sentence:" );
19       SentenceInField = new JTextField( 30 );
20       SentenceInField.addActionListener(
21
22          new ActionListener() { // anonymous inner class
23
24             // display result of call to method countDuplicates
25             public void actionPerformed( ActionEvent event ) {
26                sentence = event.getActionCommand();
27                resultField.setText( "There are " +
```

```
28                      countDuplicates( sentence ) +
29                      " words that are duplicates." );
30                }
31
32             } // end anonymous inner class
33
34          ); // end call to addActionListener
35
36          resultLabel = new JLabel( "Result:" );
37          resultField = new JTextField( 30 );
38          resultField.setEditable( false );
39
40          Container container = getContentPane();
41          container.setLayout( new FlowLayout() );
42          container.add( SentenceInLabel );
43          container.add( SentenceInField );
44          container.add( resultLabel );
45          container.add( resultField );
46
47          setSize( 340, 200 );
48          setVisible( true );
49       }
50
51       // count number of duplicate words in input snetence
52       public int countDuplicates( String string )
53       {
54          // tokenizer ignores punctuation
55          StringTokenizer tokens = new StringTokenizer ( string, " ?!.," );
56
57          Integer mapEntry;
58          HashMap map = new HashMap();
59          int duplicates = 0;
60
61          // count duplicates
62          while ( tokens.hasMoreTokens() ) {
63             String token = tokens.nextToken().toLowerCase();
64             mapEntry = ( Integer ) map.get( token );
65
66             // if key is not in map then give it value one
67             // otherwise increment its value by 1
68             if ( mapEntry == null )
69                map.put( new String( token ), new Integer( 1 ) );
70
71             else {
72                duplicates++;
73                map.put( new String( token ),
74                   new Integer( mapEntry.intValue() + 1 ) );
75             }
76          }
77
78          return duplicates;
79       }
80
```

```
81      public static void main( String args[] )
82      {
83         Duplicates application = new Duplicates();
84         application.setDefaultCloseOperation( EXIT_ON_CLOSE );
85      }
86
87   }  // end class Duplicates
```



22.12 Rewrite your solution to Exercise 20.8 to use a `LinkedList` collection.
ANS:

```
1   // Exercise 22.12 Solution: ListTest3.java
2   // Program inserts and sorts random numbers in a list,
3   // prints the sum, and displays the average.
4   import java.util.*;
5
6   public class ListTest3 {
7
8      public static void main( String args[] )
9      {
10        LinkedList list = new LinkedList();
11        Integer newNumber = null;
12
13        // Create objects to store in the List
14        for ( int k = 0; k < 25; k++ ) {
15           newNumber = new Integer( ( int ) ( Math.random() * 101 ) );
16           list.add( newNumber );
17        }
18
19        Collections.sort( list );
20        System.out.println( list.toString() );
21
22        int count = 0;
23
24        ListIterator iterator = list.listIterator();
25
26        while ( iterator.hasNext() )
27           count += ( ( Integer ) iterator.next() ).intValue();
28
29        System.out.println( "Sum is: " + count + "\nAverage is: " +
30              ( ( double ) count / list.size() ) );
31     }
```

```
32
33   }  // end class ListTest3
```

```
[3, 6, 7, 18, 27, 30, 32, 36, 41, 44, 50, 50, 52, 52, 55, 56, 63, 67, 71, 74,
85, 88, 95, 99, 100]
Sum is: 1301
Average is: 52.04
```

22.13 Rewrite your solution to Exercise 20.9 to use a `LinkedList` collection.
ANS:

```
1   // Exercise 22.13 Solution: ListReverse.java
2   // Program creates a list, then creates a reverse of the list
3   import java.util.*;
4
5   public class ListReverse {
6
7      public static void main( String args[] )
8      {
9         // create two linked lists
10        LinkedList list1 = new LinkedList();
11        LinkedList list2 = new LinkedList();
12
13        // use List insert methods
14        list1.addFirst( new Character( '5' ) );
15        list1.addFirst( new Character( '@' ) );
16        list1.addLast( new Character( 'V' ) );
17        list1.addLast( new Character( '+' ) );
18        list1.addFirst( new Character( 'P' ) );
19        list1.addFirst( new Character( 'c' ) );
20        list1.addLast( new Character( 'M' ) );
21        list1.addLast( new Character( '&' ) );
22        list1.addFirst( new Character( 'y' ) );
23        list1.addLast( new Character( 'X' ) );
24
25        System.out.println( "List: " );
26        System.out.println( list1.toString() );
27
28        list2 = reverse( list1 ); // reverse lists using method reverse
29        System.out.println( "Reversed list is:" );
30        System.out.println( list2.toString() );
31     }
32
33     // reverses a list and returns it to the caller
34     public static LinkedList reverse( List one )
35     {
36        LinkedList reversed = new LinkedList();
37
38        while ( !one.isEmpty() )
39           reversed.addFirst( one.remove( 0 ) );
40
```

```
41         return reversed;
42      }
43
44   }  // end class ListReverse
```

```
List:
[y, c, P, @, 5, V, +, M, &, X]
Reversed list is:
[X, &, M, +, V, 5, @, P, c, y]
```

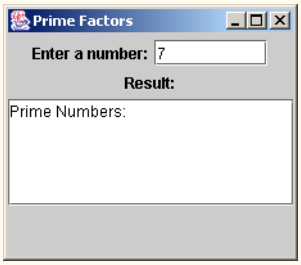
22.14 Write a program that takes a whole-number input from a user and determines whether it is prime. If the number is prime, add it to a `JTextArea`. If the number is not prime, display the prime factors of the number in a `JLabel`. Remember that a prime number's factors are only 1 and the prime number itself. Every number that is not prime has a unique prime factorization. For example, consider the number 54. The prime factors of 54 are 2, 3, 3 and 3. When the values are multiplied together, the result is 54. For the number 54, the prime factors output should be 2 and 3. Use `Sets` as part of your solution.

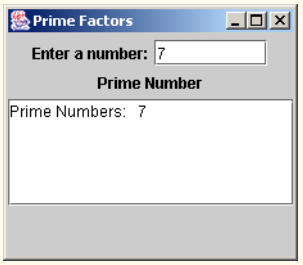
ANS:

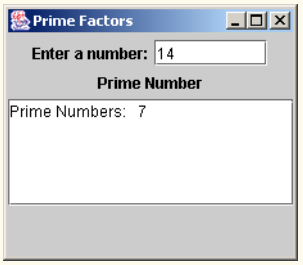
```
1   // Exercise 22.14 Solution: PrimeFactors.java
2   // Program finds the prime factors of a number using sets.
3   import java.awt.event.*;
4   import java.awt.*;
5   import java.util.*;
6   import javax.swing.*;
7
8   public class PrimeFactors extends JFrame
9   {
10     private JTextField numberInField;
11     private JLabel numberInLabel, resultLabel;
12     private JTextArea primeArea;
13
14     public PrimeFactors()
15     {
16        super( "Prime Factors" );
17
18        numberInField = new JTextField( 8 );
19        numberInField.addActionListener(
20
21           new ActionListener() { // anonymous inner class
22
23              // find prime factors of number and display results
24              public void actionPerformed( ActionEvent event )
25              {
26                 int inputNumber =
27                    Integer.parseInt( event.getActionCommand() );
28
29                 HashSet factorSet = new HashSet();
30                 factorize( inputNumber, factorSet );
31
32                 if ( factorSet.isEmpty() ) {
```

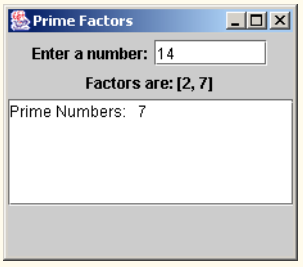
```
33                      primeArea.append( "    " + inputNumber );
34                      resultLabel.setText( "Prime Number" );
35                   }
36
37                   else
38                      resultLabel.setText(
39                      "Factors are: " + factorSet.toString() );
40                }
41
42             } // end anonymous inner class
43
44          ); // end call to addActionListener
45
46          numberInLabel = new JLabel( "Enter a number:" );
47          resultLabel = new JLabel( "Result:" );
48          primeArea = new JTextArea( "Prime Numbers:", 5, 20 );
49          JScrollPane scrollPane = new JScrollPane( primeArea );
50
51          Container container = getContentPane();
52          container.setLayout( new FlowLayout() );
53          container.add( numberInLabel );
54          container.add( numberInField );
55          container.add( resultLabel );
56          container.add( scrollPane );
57
58          setSize( 230, 200 );
59          setVisible( true );
60       }
61
62       // find prime factors
63       public boolean factorize( int number, HashSet set )
64       {
65          for ( int factor = 2 ; factor <= number / 2; factor++ ) {
66
67             if ( number % factor == 0 ) {
68
69                // tryFact is a factor of number, save the factor
70                set.add( new Integer( factor ) );
71
72                // just look at what's left over for more factors
73                if ( !factorize( ( number / factor ), set ) )
74                   set.add( new Integer( number / factor ) );
75
76                return true;
77             }
78          }
79
80          return false;
81       }
82
83       public static void main( String args[] )
84       {
85          PrimeFactors application = new PrimeFactors();
86          application.setDefaultCloseOperation( EXIT_ON_CLOSE );
87       }
```

```
88
89 } // end class PrimeFactors
```









22.15 Rewrite your solution to Exercise 20.21 to use a `LinkedList`.
ANS:

```
1  // Exercise 22.15 Solution: List3.java
2  // Class List3 definition
3  import java.util.*;
4
5  public class List3 extends LinkedList {
6
7     public List3()
8     {
9        super();
10    }
11
12    // insert Integer
13    public void insert( Integer number )
14    {
15       // empty list
16       if ( isEmpty() ) {
17          addFirst( number );
18       }
19
20       // insert into list
21       else {
22
23          // insert at front
24          if ( ( ( Integer ) getFirst() ).intValue() > number.intValue() )
25             addFirst( number );
```

```
26
27          // insert at back
28          else if ( ( ( Integer ) getLast() ).intValue() <
29                                        number.intValue() )
30
31             addLast( number );
32
33          // insert in middle of list
34          else {
35             int index = 0;
36
37             // locate proper place to insert
38             while ( !get( index ).equals( getLast() ) && ( ( Integer )
39                get( index ) ).intValue() < number.intValue() ) {
40                index++;
41             }
42
43             add( index, number );
44          }
45       }
46
47    }  // end method insert
48
49 }  // end class List3
```

ANS:

```
1  // Exercise 22.15 Solution: List3Test.java
2  // Program creates a list of random numbers then searches it.
3
4  public class List3Test {
5
6     public static void main( String args[] )
7     {
8        List3 list = new List3();
9        Integer number = null;
10
11       // create objects to store in the List
12       for ( int i = 1; i <= 25; i++ ) {
13          number = new Integer( ( int ) ( Math.random() * 101 ) );
14          list.insert( number );
15       }
16
17       System.out.println( list.toString() );
18
19       if ( list.contains( new Integer( 34 ) ) )
20          System.out.println( "Value found: 34" );
21       else
22          System.out.println( "Value not found: 34" );
23
24       if ( list.contains( new Integer( 50 ) ) )
25          System.out.println( "Value found: 50" );
```

```
26         else
27            System.out.println( "Value not found: 50" );
28
29         if ( list.contains( new Integer( 72 ) ) )
30            System.out.println( "Value found: 72" );
31         else
32            System.out.println( "Value not found: 72" );
33
34      }
35
36   }  // end class List3Test
```

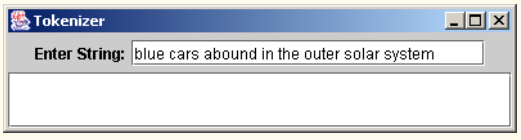
```
[6, 13, 17, 20, 24, 35, 36, 51, 52, 52, 57, 58, 60, 61, 61, 63, 63, 68, 70,
75, 77, 84, 85, 92, 97]
Value not found: 34
Value not found: 50
Value not found: 72
```

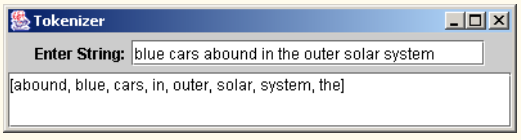
22.16 Write a program that uses a `StringTokenizer` to tokenize a line of text input by the user and places each token in a tree. Print the elements of the sorted tree.

ANS:

```
1   // Exercise 22.16 Solution: TreeTest.java
2   // Program tokenizes text input by user and places each
3   // token in a tree. Sorted tree elements are then printed.
4   import java.awt.*;
5   import java.util.*;
6   import java.awt.event.*;
7   import javax.swing.*;
8
9   public class TreeTest extends JFrame {
10     private JLabel prompt;
11     private JTextField input;
12     private JTextArea display;
13     private JPanel panel;
14
15     public TreeTest()
16     {
17        super( "Tokenizer" );
18
19        prompt = new JLabel( "Enter String:" );
20        input = new JTextField( 25 );
21
22        input.addActionListener(
23
24           new ActionListener() { // anonymous inner class
25
26              // tokenize input text and add each token to tree
27              public void actionPerformed( ActionEvent event )
28              {
29                 TreeSet tree = new TreeSet();
```

```
30                StringTokenizer tokenizer =
31                   new StringTokenizer( input.getText() );
32
33                while ( tokenizer.hasMoreTokens() )
34                   tree.add( tokenizer.nextToken() );
35
36                // print tree in text area
37                display.setText( tree.toString() );
38             }
39
40          } // end anonymous inner class
41
42       ); // end call to addActionListener
43
44       panel = new JPanel();
45       panel.add( prompt, BorderLayout.NORTH );
46       panel.add( input, BorderLayout.SOUTH );
47
48       display = new JTextArea();
49
50       Container container = getContentPane();
51       container.add( panel, BorderLayout.NORTH );
52       container.add( new JScrollPane( display ), BorderLayout.CENTER );
53
54       setSize( 400, 100 );
55       setVisible( true );
56    }
57
58    public static void main( String args[] )
59    {
60       TreeTest application = new TreeTest();
61       application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
62    }
63
64 }  // end class TreeTest
```





23

Java Database Connectivity (JDBC)

Objectives

- To understand relational databases.
- To understand basic database queries using SQL.
- To use the classes and interfaces of package `java.sql` to manipulate databases.

It is a capital mistake to theorize before one has data.
Arthur Conan Doyle

Now go, write it before them in a table, and note it in a book, that it may be for the time to come for ever and ever.
The Holy Bible, Isaiah 30:8

Get your facts first, and then you can distort them as much as you please.
Mark Twain

I like two kinds of men: domestic and foreign.
Mae West

SELF-REVIEW EXERCISES

23.1 Fill in the blanks in each of the following statements:

a) The industry standard database query language is _____.
ANS: SQL

b) A table in a database consists of _____ and _____.
ANS: rows, columns

c) SQL query results are manipulated in Java as _____ objects.
ANS: `ResultSet`

d) The _____ uniquely identifies each row in a table.
ANS: primary key

e) SQL keyword _____ is followed by the selection criteria that specify the rows to select in a query.
ANS: `WHERE`

f) SQL keywords _____ specify the order in which rows are sorted in a query.
ANS: `ORDER BY`

g) Merging rows from multiple database tables is called _____ the tables.
ANS: joining

h) A(n) _____ is an organized collection of data.
ANS: database

i) A(n) _____ is a set of columns whose values match the primary key valkues of another table.
ANS: foreign key

j) Package _____ contains classes and interfaces for manipulating relational databases in Java.
ANS: `java.sql`

k) Interface _____ helps manage the connection between a Java program and a database.
ANS: `Connection`

l) A(n) _____ object is used to submit a query to a database.
ANS: `Statement`

EXERCISES

23.2 Using the techniques shown in this chapter, define a complete query application for the `books` database. Provide a series of predefined queries, with an appropriate name for each query, displayed in a `JComboBox`. Also allow users to supply their own queries and add them to the `JComboBox`. Provide the following predefined queries:

a) Select all authors from the `authors` table.
b) Select all publishers from the `publishers` table.
c) Select a specific author and list all books for that author. Include the title, year and ISBN. Order the information alphabetically by the author's last name and first name.
d) Select a specific publisher and list all books published by that publisher. Include the title, year and ISBN. Order the information alphabetically by title.
e) Provide any other queries you feel are appropriate.

ANS:

```
1  // Exercise 23.2 Solution: DisplayQueryResults.java
2  import java.sql.*;
3  import javax.swing.*;
4  import java.awt.*;
```

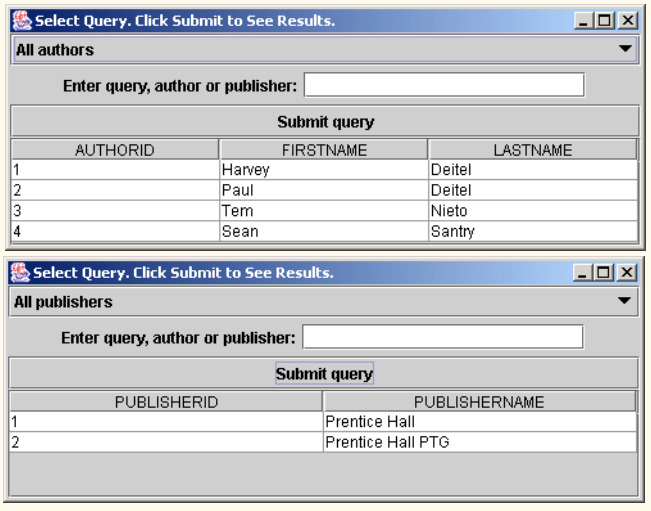
```
5  import java.awt.event.*;
6  import java.util.*;
7
8  public class DisplayQueryResults extends JFrame {
9     private Connection connection;
10    private Statement statement;
11    private ResultSet resultSet;
12    private ResultSetMetaData rsMetaData;
13    private JTable table;
14    private JComboBox inputQuery;
15    private JButton submitQuery;
16    private JTextField input;
17
18    public DisplayQueryResults()
19    {
20       super( "Select Query. Click Submit to See Results." );
21
22       // The URL specifying the books database to which this program
23       // connects to using JDBC
24       String url = "jdbc:db2j:books";
25
26       // Load the driver to allow connection to the database
27       try {
28          Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
29
30          connection = DriverManager.getConnection( url );
31       }
32       catch ( ClassNotFoundException cnfex ) {
33          System.err.println( "Failed to load JDBC driver." );
34          cnfex.printStackTrace();
35          System.exit( 1 );  // terminate program
36       }
37       catch ( SQLException sqlex ) {
38          System.err.println( "Unable to connect" );
39          sqlex.printStackTrace();
40          System.exit( 1 );  // terminate program
41       }
42
43       String names[] = { "All authors", "All publishers", "All books",
44          "A specific author", "A specific publisher" };
45
46       // If connected to database, set up GUI
47       inputQuery = new JComboBox( names );
48
49       submitQuery = new JButton( "Submit query" );
50       submitQuery.addActionListener(
51
52          new ActionListener() {
53
54             public void actionPerformed( ActionEvent e )
55             {
56                getTable();
57             }
58          }
59       );
```

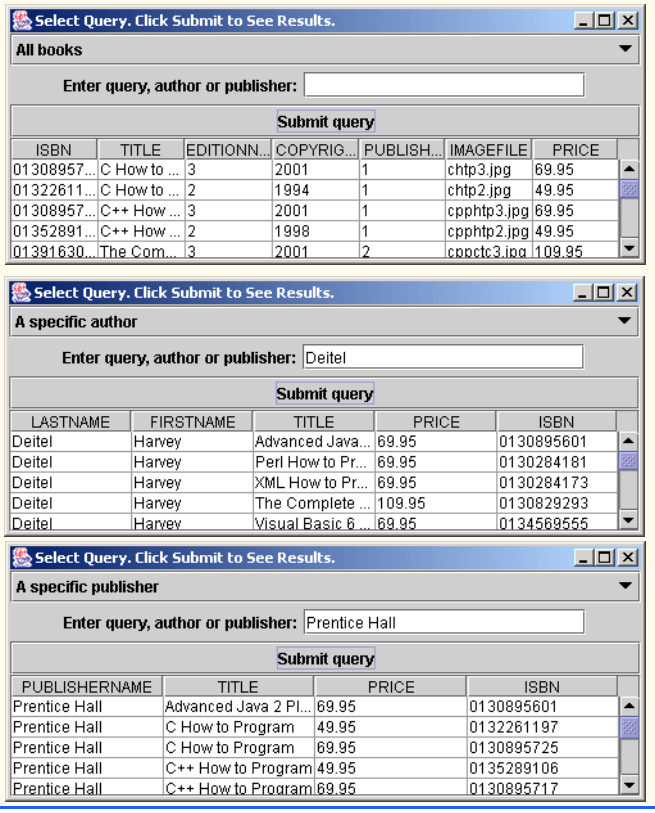
```
60
61          JPanel topPanel = new JPanel();
62          input = new JTextField( 20 );
63          input.addActionListener(
64
65             new ActionListener() {
66
67                public void actionPerformed( ActionEvent e )
68                {
69                   try {
70                      String query = input.getText();
71                      statement = connection.createStatement();
72                      resultSet = statement.executeQuery( query );
73                      displayResultSet( resultSet );
74                   }
75                   catch ( SQLException sqlex ) {
76                      sqlex.printStackTrace();
77                   }
78                }
79             }
80          );
81
82          JPanel centerPanel = new JPanel();
83          centerPanel.setLayout( new FlowLayout() );
84         centerPanel.add( new JLabel( "Enter query, author or publisher:" ) );
85          centerPanel.add( input );
86          topPanel.setLayout( new BorderLayout() );
87          topPanel.add( inputQuery, BorderLayout.NORTH );
88          topPanel.add( centerPanel, BorderLayout.CENTER );
89          topPanel.add( submitQuery, BorderLayout.SOUTH );
90
91          table = new JTable( 4, 4 );
92
93          Container c = getContentPane();
94          c.setLayout( new BorderLayout() );
95          c.add( topPanel, BorderLayout.NORTH );
96          c.add( table, BorderLayout.CENTER );
97
98          getTable();
99
100         setSize( 500, 500 );
101         setVisible( true );
102
103      } // end constructor DisplayQueryResult
104
105      private void getTable()
106      {
107         try {
108            int selection = inputQuery.getSelectedIndex();
109            String query = null;
110
111            switch ( selection ) {
112               case 0:
113                  query = "SELECT * FROM Authors";
114                  break;
```

```
115             case 1:
116                query = "SELECT * FROM Publishers";
117                break;
118             case 2:
119                query = "SELECT * FROM TITLES";
120                break;
121             case 3:
122                query = "SELECT Authors.LastName, Authors.FirstName, "+
123                   "Titles.Title, Titles.Price, " + "Titles.ISBN FROM " +
124                   "Titles INNER JOIN (AuthorISBN INNER JOIN Authors ON" +
125                   " AuthorISBN.AuthorID = Authors.AuthorID) ON " +
126                   "Titles.ISBN = AuthorISBN.ISBN WHERE Authors.LastName" +
127                   " = '" + input.getText() + "' ORDER BY " +
128                   "Authors.LastName, Authors.FirstName ASC";
129                   break;
130             case 4:
131                query = "SELECT Publishers.PublisherName, Titles.Title, " +
132                   "Titles.Price, Titles.ISBN FROM Titles INNER JOIN " +
133                   "Publishers ON Publishers.PublisherID = " +
134                   "Titles.PublisherID WHERE Publishers.PublisherName = '"
135                   + input.getText() + "' ORDER BY Titles.Title ASC";
136                   break;
137          }
138
139          statement = connection.createStatement();
140          resultSet = statement.executeQuery( query );
141          displayResultSet( resultSet );
142
143       } // end try
144
145       catch ( SQLException sqlex ) {
146          sqlex.printStackTrace();
147       }
148
149    } // end method getTable
150
151    private void displayResultSet( ResultSet rs ) throws SQLException
152    {
153       // position to first record
154       boolean moreRecords = rs.next();
155
156       // If there are no records, display a message
157       if ( !moreRecords ) {
158          JOptionPane.showMessageDialog( this,
159             "ResultSet contained no records" );
160          setTitle( "No records to display" );
161          return;
162       }
163
164       Vector columnHeads = new Vector();
165       Vector rows = new Vector();
166
167       try {
168          // get column heads
169          ResultSetMetaData rsmd = rs.getMetaData();
```

```
170
171          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
172             columnHeads.addElement( rsmd.getColumnName( i ) );
173
174          // get row data
175          do {
176             rows.addElement( getNextRow( rs, rsmd ) );
177          } while ( rs.next() );
178
179          // display table with ResultSet contents
180          table = new JTable( rows, columnHeads );
181          JScrollPane scroller = new JScrollPane( table );
182          Container c = getContentPane();
183          c.remove( 1 );
184          c.add( scroller, BorderLayout.CENTER );
185          c.validate();
186
187       } // end try
188
189       catch ( SQLException sqlex ) {
190          sqlex.printStackTrace();
191       }
192
193    } // end method displayResultSet
194
195    private Vector getNextRow( ResultSet rs,
196       ResultSetMetaData rsmd ) throws SQLException
197    {
198       Vector currentRow = new Vector();
199
200       for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
201          switch( rsmd.getColumnType( i ) ) {
202             case Types.VARCHAR:
203             case Types.LONGVARCHAR:
204                   currentRow.addElement( rs.getString( i ) );
205                break;
206             case Types.INTEGER:
207                   currentRow.addElement( new Long( rs.getLong( i ) ) );
208                break;
209             case Types.REAL:
210                   currentRow.addElement( new Float( rs.getDouble( i ) ) );
211                break;
212             default:
213                System.out.println( "Type was: " +
214                   rsmd.getColumnTypeName( i ) );
215          }
216
217       return currentRow;
218
219    } // end method getNextRow
220
221    public void shutDown()
222    {
223       try {
```

```
224            connection.close();
225         }
226         catch ( SQLException sqlex ) {
227            System.err.println( "Unable to disconnect" );
228            sqlex.printStackTrace();
229         }
230      }
231
232      public static void main( String args[] )
233      {
234         final DisplayQueryResults app = new DisplayQueryResults();
235         app.addWindowListener(
236
237            new WindowAdapter() {
238
239               public void windowClosing( WindowEvent e )
240               {
241                  app.shutDown();
242                  System.exit( 0 );
243               }
244            }
245         );
246      }
247
248   } // end class DisplayQueryResults
```





23.3 Modify Exercise 23.2 to define a complete database manipulation application for the `books` database. In addition to the querying, the user should be able to edit existing data and add new data to the database (obeying referential and entity integrity constraints). Allow the user to edit the database in the following ways:

a) Add a new author.
b) Edit the existing information for an author.
c) Add a new title for an author. (Remember that the book must have an entry in the `authorISBN` table.) Be sure to specify the publisher of the title.
d) Add a new publisher.
e) Edit the existing information for a publisher.
f) For each of the preceding database manipulations, design an appropriate GUI to allow the user to perform the data manipulation.

ANS:

```
1  // Exercise 23.3 Solution: DisplayQueryResults.java
2  import java.sql.*;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.util.*;
```

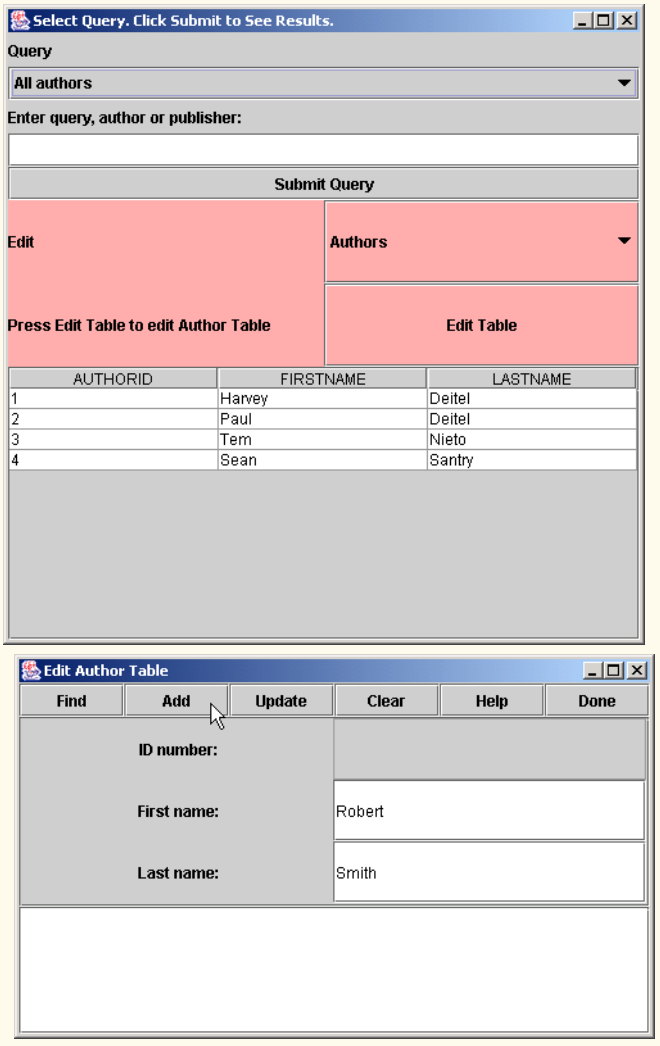
```
 7
 8  public class DisplayQueryResults extends JFrame {
 9     private Connection connection;
10     private Statement statement;
11     private ResultSet resultSet;
12     private ResultSetMetaData rsMetaData;
13     private JTable table;
14     private JComboBox inputQuery;
15     private JButton submitInputQuery;
16     private JTextField inputQueryTF;
17     private EditPanel editPanel;
18
19     public DisplayQueryResults()
20     {
21        super( "Select Query. Click Submit to See Results." );
22
23        // The URL specifying the books database to which
24        // this program connects to
25        String url = "jdbc:db2j:books";
26
27        // Load the driver to allow connection to the database
28        try {
29           Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
30
31           connection = DriverManager.getConnection( url );
32        }
33        catch ( ClassNotFoundException cnfex ) {
34           System.err.println( "Failed to load JDBC/ODBC driver." );
35           cnfex.printStackTrace();
36           System.exit( 1 );  // terminate program
37        }
38        catch ( SQLException sqlex ) {
39           System.err.println( "Unable to connect" );
40           sqlex.printStackTrace();
41           System.exit( 1 );  // terminate program
42        }
43
44        // If connected to database, set up GUI
45        String queryNames[] = { "All authors", "All publishers",
46           "All books", "A specific author", "A specific publisher" };
47        inputQuery = new JComboBox( queryNames );
48        JLabel inputLabel = new JLabel("Query");
49        submitInputQuery = new JButton( "Submit Query" );
50        submitInputQuery.addActionListener(
51
52           new ActionListener() {
53
54              public void actionPerformed( ActionEvent e )
55              {
56                 getTable();
57              }
58           }
59        );
60
```

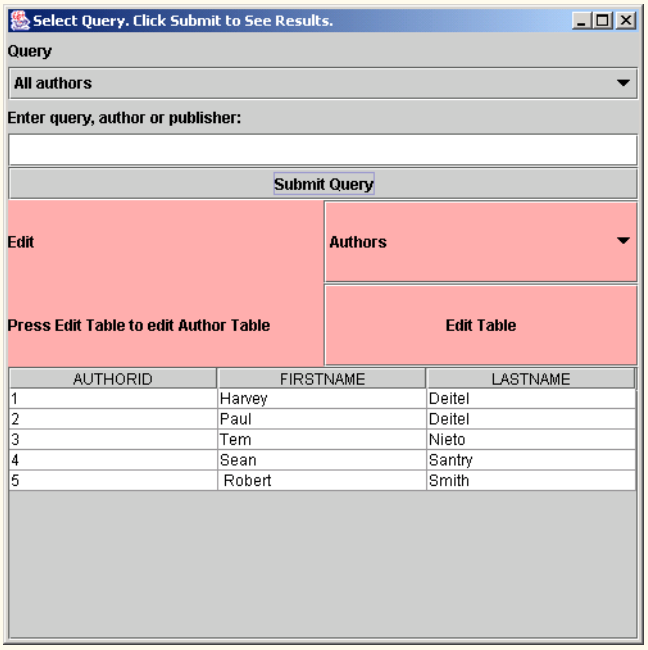
```
61          inputQueryTF = new JTextField( 30 );
62          inputQueryTF.addActionListener(
63
64             new ActionListener() {
65
66                public void actionPerformed( ActionEvent e )
67                {
68                   try {
69                      String query = inputQueryTF.getText();
70                      statement = connection.createStatement();
71                      resultSet = statement.executeQuery( query );
72                      displayResultSet( resultSet );
73                   }
74                   catch ( SQLException sqlex ) {
75                      sqlex.printStackTrace();
76                   }
77                }
78             }
79          );
80
81          JPanel inputPanel = new JPanel();
82          inputPanel.setLayout( new GridLayout( 5, 1 ) );
83          inputPanel.add( inputLabel );
84          inputPanel.add( inputQuery);
85          inputPanel.add( new JLabel( "Enter query, author or publisher:" ) );
86          inputPanel.add( inputQueryTF );
87          inputPanel.add( submitInputQuery);
88
89          editPanel = new EditPanel(connection);
90
91          JPanel instructionPanel = new JPanel();
92          instructionPanel.setLayout( new GridLayout( 2, 1 ) );
93          instructionPanel.add( inputPanel );
94          instructionPanel.add( editPanel.getThePanel() );
95
96          JPanel topPanel = new JPanel();
97          topPanel.setLayout( new BorderLayout() );
98          topPanel.add( instructionPanel, BorderLayout.NORTH );
99
100         table = new JTable( 4, 4 );
101
102         Container c = getContentPane();
103         c.setLayout( new BorderLayout() );
104         c.add( topPanel, BorderLayout.NORTH );
105         c.add( table, BorderLayout.CENTER );
106
107         getTable();
108
109         setSize( 500, 500 );
110         show();
111
112      } // end constructor DisplayQueryResults
113
114      public Connection getConnection() { return connection; }
```

```
115
116    private void getTable()
117    {
118       try {
119          int selection = inputQuery.getSelectedIndex();
120          String query = null;
121
122          switch ( selection ) {
123             case 0:
124                query = "SELECT * FROM Authors";
125                break;
126             case 1:
127                query = "SELECT * FROM Publishers";
128                break;
129             case 2:
130                query = "SELECT * FROM TITLES";
131                break;
132             case 3:
133                query = "SELECT Authors.LastName, Authors.FirstName, "+
134                   "Titles.Title, Titles.Price, " + "Titles.ISBN FROM " +
135                  "Titles INNER JOIN (AuthorISBN INNER JOIN Authors ON " +
136                   "AuthorISBN.AuthorID = Authors.AuthorID) ON " +
137                   "Titles.ISBN = AuthorISBN.ISBN WHERE Authors.LastName" +
138                   " = '" + inputQueryTF.getText() +
139                   "' ORDER BY Authors.LastName, Authors.FirstName ASC";
140                   break;
141             case 4:
142                query = "SELECT Publishers.PublisherName, Titles.Title, " +
143                   "Titles.Price, Titles.ISBN FROM Titles INNER JOIN " +
144                   "Publishers ON Publishers.PublisherID = " +
145                   "Titles.PublisherID WHERE Publishers.PublisherName = '"
146                  + inputQueryTF.getText() + "' ORDER BY Titles.Title ASC";
147                   break;
148          }
149
150          statement = connection.createStatement();
151          resultSet = statement.executeQuery( query );
152          displayResultSet( resultSet );
153
154       } // end try
155
156       catch ( SQLException sqlex ) {
157          sqlex.printStackTrace();
158       }
159
160    } // end method getTable
161
162    public void displayResultSet( ResultSet rs ) throws SQLException
163    {
164       // position to first record
165       boolean moreRecords = rs.next();
166
167       // If there are no records, display a message
168       if ( !moreRecords ) {
```

```
169            JOptionPane.showMessageDialog( this,
170               "ResultSet contained no records" );
171            setTitle( "No records to display" );
172            return;
173         }
174
175         Vector columnHeads = new Vector();
176         Vector rows = new Vector();
177
178         try {
179            // get column heads
180            ResultSetMetaData rsmd = rs.getMetaData();
181
182            for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
183               columnHeads.addElement( rsmd.getColumnName( i ) );
184
185            // get row data
186            do {
187               rows.addElement( getNextRow( rs, rsmd ) );
188            } while ( rs.next() );
189
190            // display table with ResultSet contents
191            table = new JTable( rows, columnHeads );
192            JScrollPane scroller = new JScrollPane( table );
193            Container c = getContentPane();
194            c.remove( 1 );
195            c.add( scroller, BorderLayout.CENTER );
196            c.validate();
197
198         } // end try
199
200         catch ( SQLException sqlex ) {
201            sqlex.printStackTrace();
202         }
203
204      } // end method displayResultSet
205
206      private Vector getNextRow( ResultSet rs,
207         ResultSetMetaData rsmd ) throws SQLException
208      {
209         Vector currentRow = new Vector();
210
211         for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
212            switch( rsmd.getColumnType( i ) ) {
213               case Types.VARCHAR:
214               case Types.LONGVARCHAR:
215                     currentRow.addElement( rs.getString( i ) );
216                  break;
217               case Types.INTEGER:
218                     currentRow.addElement( new Long( rs.getLong( i ) ) );
219                  break;
220               case Types.REAL:
221                     currentRow.addElement( new Float( rs.getDouble( i ) ) );
222                  break;
```

```
223             default:
224                System.out.println( "Type was: " +
225                   rsmd.getColumnTypeName( i ) );
226          }
227
228       return currentRow;
229
230    } // end method getNextRow
231
232    public void shutDown()
233    {
234       try {
235          connection.close();
236       }
237       catch ( SQLException sqlex ) {
238          System.err.println( "Unable to disconnect" );
239          sqlex.printStackTrace();
240       }
241    }
242
243    public static void main( String args[] )
244    {
245       final DisplayQueryResults app = new DisplayQueryResults();
246       app.addWindowListener(
247
248          new WindowAdapter() {
249
250             public void windowClosing( WindowEvent e )
251             {
252                app.shutDown();
253                System.exit( 0 );
254             }
255          }
256       );
257    }
258
259 } // end class DisplayQueryResults
```






```
1  // Exercise 23.3 solution: EditAuthor.java
2  import java.sql.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class EditAuthor extends JFrame {
8     private EditAuthorsControlPanel controls;
9     private AuthorScrollingPanel scrollArea;
10    private JTextArea output;
11    private String url;
12    private Connection connect;
13    private JScrollPane textpane;
14
15    public EditAuthor( Connection connection )
16    {
17       super( "Edit Author Table" );
18       connect = connection;
19
20       Container c = getContentPane();
21
22       // Start screen layout
23       scrollArea = new AuthorScrollingPanel();
24       output = new JTextArea( 6, 30 );
25       c.setLayout( new BorderLayout() );
26       c.add( new JScrollPane( scrollArea ), BorderLayout.CENTER );
27       textpane = new JScrollPane( output );
```

```
28          c.add( textpane, BorderLayout.SOUTH );
29
30          // Complete screen layout
31          controls = new EditAuthorsControlPanel(
32             connect, scrollArea, output, this );
33          c.add( controls, BorderLayout.NORTH );
34
35          setSize( 500, 300 );
36
37       } // end constructor EditAuthor
38
39    } // end class EditAuthor
40
41    class AddAuthor implements ActionListener {
42       private AuthorScrollingPanel fields;
43       private JTextArea output;
44       private Connection connection;
45
46       public AddAuthor( Connection c, AuthorScrollingPanel f, JTextArea o )
47       {
48          connection = c;
49          fields = f;
50          output = o;
51       }
52
53       public void actionPerformed( ActionEvent e )
54       {
55          try {
56             Statement statement = connection.createStatement();
57
58             if ( !fields.last.getText().equals( "" ) &&
59                  !fields.first.getText().equals( "" ) ) {
60                String query = "INSERT INTO Authors ( FirstName, LastName )" +
61                   " VALUES (' " + fields.first.getText() + "', '" +
62                    fields.last.getText() + "')";
63                output.append( "\nSending query: " +
64                   connection.nativeSQL( query ) + "\n" );
65                int result = statement.executeUpdate( query );
66
67                if ( result == 1 )
68                   output.append( "\nInsertion successful\n" );
69                else {
70                   output.append( "\nInsertion failed\n" );
71                   fields.clear();
72                }
73             }
74             else
75                output.append( "\nYou must enter at least first and " +
76                   "last name then press Add\n" );
77
78             statement.close();
79
80          } // end try
81
```

```
82          catch ( SQLException sqlex ) {
83             sqlex.printStackTrace();
84             output.append( sqlex.toString() );
85          }
86
87       } // end method actionPerformed
88
89    } // end class AddAuthor
90
91    class FindAuthor implements ActionListener {
92       private AuthorScrollingPanel fields;
93       private JTextArea output;
94       private Connection connection;
95
96       public FindAuthor( Connection c, AuthorScrollingPanel f, JTextArea o )
97       {
98          connection = c;
99          fields = f;
100         output = o;
101      }
102
103      public void actionPerformed( ActionEvent e )
104      {
105         try {
106
107            if ( !fields.empty() ) {
108               Statement statement = connection.createStatement();
109               String query = "SELECT * FROM Authors WHERE ";
110
111               boolean lastEmpty  = fields.last.getText().equals( "" );
112               boolean firstEmpty = fields.first.getText().equals( "" );
113
114               query += lastEmpty ?
115                  "" : "LastName = '" + fields.last.getText() + "'";
116               query += ( !lastEmpty && !firstEmpty ? " AND " : "" );
117               query += firstEmpty ?
118                  "" : "FirstName = '" + fields.first.getText() + "'";
119
120               output.append( "\nSending query: " +
121                  connection.nativeSQL( query ) + "\n" );
122
123               ResultSet rs = statement.executeQuery( query );
124               display( rs );
125
126               output.append( "\nQuery successful\n" );
127               statement.close();
128
129            } // end if
130
131            else
132               fields.last.setText(
133                  "Enter last name (at least) then press Find" );
134
135         } // end try
```

```
136
137         catch ( SQLException sqlex ) {
138            sqlex.printStackTrace();
139            output.append( sqlex.toString() );
140         }
141
142      } // end method actionPerformed
143
144      // Display results of query. If rs is null
145      public void display( ResultSet rs )
146      {
147         try {
148
149            // If there are no records, display a message
150            if ( !rs.next() ) {
151               output.append( "No records found" );
152               return;
153            }
154
155            int id = rs.getInt( 1 );
156            fields.id.setText( String.valueOf( id ) );
157            fields.first.setText( rs.getString( 2 ) );
158            fields.last.setText( rs.getString( 3 ) );
159         }
160         catch ( SQLException sqlex ) {
161            sqlex.printStackTrace();
162            output.append( sqlex.toString() );
163         }
164
165      } // end method display
166
167   } // end class FindAuthor
168
169   class UpdateAuthor implements ActionListener {
170      private AuthorScrollingPanel fields;
171      private JTextArea output;
172      private Connection connection;
173
174      public UpdateAuthor( Connection c, AuthorScrollingPanel f,
175                           JTextArea o )
176      {
177         connection = c;
178         fields = f;
179         output = o;
180      }
181
182      public void actionPerformed( ActionEvent e )
183      {
184         try {
185            Statement statement = connection.createStatement();
186
187            if ( !fields.id.getText().equals( "" ) ) {
188               String query = "UPDATE Authors SET " + "FirstName='" +
189                  fields.first.getText() + "', LastName='" +
```

```
190                fields.last.getText() + "' WHERE AuthorID=" +
191                fields.id.getText();
192
193             output.append( "\nSending query: " +
194                connection.nativeSQL( query ) + "\n" );
195
196             int result = statement.executeUpdate( query );
197
198             if ( result == 1 )
199                output.append( "\nUpdate successful\n" );
200             else {
201                output.append( "\nUpdate failed\n" );
202                fields.clear( );
203             }
204
205             statement.close();
206
207          } // end if
208
209          else
210             output.append( "\nYou may only update an existing record. " +
211                "The ID field must not be empty. Use Find to locate the " +
212                "record, then modify the information and press Update.\n" );
213
214       } // end try
215
216       catch ( SQLException sqlex ) {
217          sqlex.printStackTrace();
218          output.append( sqlex.toString() );
219       }
220
221    } // end method actionPerformed
222
223 } // end class UpdateAuthor
224
225 class AuthorHelp implements ActionListener {
226    private JTextArea output;
227
228    public AuthorHelp( JTextArea o )
229    {
230       output = o;
231    }
232
233    public void actionPerformed( ActionEvent e )
234    {
235       output.append( "\nClick Find to locate an author record.\nClick " +
236          "Add to insert a new author record.\nClick Update to update " +
237          "the information in an author record.\nClick Clear to empty" +
238          " the textfields.\n" );
239    }
240
241 } // end class AuthorHelp
242
243 class AuthorDone implements ActionListener {
```

```
244    private EditAuthor parent;
245
246    public AuthorDone( EditAuthor p ) { parent = p ; }
247
248    public void actionPerformed( ActionEvent e )
249    {
250       parent.setVisible( false );
251    }
252
253 } // end class AuthorDone
254
255 class EditAuthorsControlPanel extends JPanel {
256    private JButton findName, addName, updateName, clear, help, done;
257
258    public EditAuthorsControlPanel( Connection c, AuthorScrollingPanel s,
259       JTextArea t , EditAuthor p)
260    {
261       setLayout( new GridLayout( 1, 6 ) );
262
263       findName = new JButton( "Find" );
264       findName.addActionListener( new FindAuthor( c, s, t ) );
265       add( findName );
266
267       addName = new JButton( "Add" );
268       addName.addActionListener( new AddAuthor( c, s, t ) );
269       add( addName );
270
271       updateName = new JButton( "Update" );
272       updateName.addActionListener( new UpdateAuthor( c, s, t ) );
273       add( updateName );
274
275       clear = new JButton( "Clear" );
276       clear.addActionListener( new ClearAuthorFields( s ) );
277       add( clear );
278
279       help = new JButton( "Help" );
280       help.addActionListener( new AuthorHelp( t ) );
281       add( help );
282
283       done = new JButton( "Done" );
284       done.addActionListener( new AuthorDone( p ) );
285       add( done );
286
287    } // end constructor EditAuthorsControlPanel
288
289 } // end class EditAuthorsControlPanel
290
291 class AuthorScrollingPanel extends JPanel {
292    private JPanel labelPanel, fieldsPanel;
293    private String labels[] = { "ID number:", "First name:", "Last name:"};
294    JTextField id, first, last;
295
296    public AuthorScrollingPanel()
297    {
```

```
298         // Label panel
299         labelPanel = new JPanel();
300         labelPanel.setLayout( new GridLayout( labels.length, 1 ) );
301
302         ImageIcon ii = new ImageIcon( "images/icon.jpg" );
303
304         for ( int i = 0; i < labels.length; i++ )
305            labelPanel.add( new JLabel( labels[ i ], ii, 0 ) );
306
307         // TextField panel
308         fieldsPanel = new JPanel();
309         fieldsPanel.setLayout( new GridLayout( labels.length, 1 ) );
310         id = new JTextField( 20 );
311         id.setEditable( false );
312         fieldsPanel.add( id );
313         first = new JTextField( 20 );
314         fieldsPanel.add( first );
315         last = new JTextField( 20 );
316         fieldsPanel.add( last );
317
318         setLayout( new GridLayout( 1, 2 ) );
319         add( labelPanel );
320         add( fieldsPanel );
321
322      } // end constructor AuthorScrollingPanel
323
324      // returns if fields are all empty (except ID)
325      public boolean empty()
326      {
327         if ( first.getText().equals( "" ) && last.getText().equals( "" ) )
328           return true;
329
330         return false;
331      }
332
333      public void clear()
334      {
335         id.setText( "" );
336         first.setText( "" );
337         last.setText( "" );
338      }
339
340   } // end class AuthorScrollingPanel
341
342   class ClearAuthorFields implements ActionListener {
343      private AuthorScrollingPanel fields;
344
345      public ClearAuthorFields( AuthorScrollingPanel f )
346      {
347         fields = f;
348      }
349
350      public void actionPerformed( ActionEvent e )
351      {
```

```
352         fields.clear();
353      }
354
355   } // end class ClearAuthorFields
```

```
1   // Exercise 23.3 solution: EditAuthorISBN.java
2   import java.sql.*;
3   import java.awt.*;
4   import java.awt.event.*;
5   import javax.swing.*;
6
7   public class EditAuthorISBN extends JFrame {
8      private EditAuthorISBNsControlPanel controls;
9      private AuthorISBNScrollingPanel scrollArea;
10     private JTextArea output;
11     private String url;
12     private Connection connect;
13     private JScrollPane textpane;
14
15     public EditAuthorISBN( Connection connection )
16     {
17        super( "Edit AuthorISBN Table" );
18        connect = connection;
19
20        Container c = getContentPane();
21
22        // Start screen layout
23        scrollArea = new AuthorISBNScrollingPanel();
24        output = new JTextArea( 6, 30 );
25        c.setLayout( new BorderLayout() );
26        c.add( new JScrollPane( scrollArea ), BorderLayout.CENTER );
27        textpane = new JScrollPane( output );
28        c.add( textpane, BorderLayout.SOUTH );
29
30        // Complete screen layout
31        controls = new EditAuthorISBNsControlPanel(
32           connect, scrollArea, output, this );
33        c.add( controls, BorderLayout.NORTH );
34
35        setSize( 500, 300 );
36
37     } // end constructor EditAuthorISBN
38
39  } // end class EditAuthorISBN
40
41  class AddAuthorISBN implements ActionListener {
42     private AuthorISBNScrollingPanel fields;
43     private JTextArea output;
44     private Connection connection;
45
46     public AddAuthorISBN( Connection c, AuthorISBNScrollingPanel f,
47        JTextArea o )
48     {
```

```
49          connection = c;
50          fields = f;
51          output = o;
52       }
53
54       public void actionPerformed( ActionEvent e )
55       {
56          try {
57             Statement statement = connection.createStatement();
58
59             if ( !fields.anyEmpty() ) {
60                String query = "INSERT INTO AuthorISBN ( ISBN, AuthorID ) " +
61                   "VALUES ('" + fields.isbn.getText() + "','" +
62                   fields.authorID.getText() + "')";
63                output.append( "\nSending query: " +
64                   connection.nativeSQL( query ) + "\n" );
65
66                if ( 1 == statement.executeUpdate( query ) )
67                   output.append( "\nInsertion successful\n" );
68                else {
69                   output.append( "\nInsertion failed\n" );
70                   fields.clear();
71                }
72             }
73             else
74                output.append( "\nEnter ISBN and authorID name and " +
75                   "then press Add\n" );
76
77             statement.close();
78
79          } // end try
80
81          catch ( SQLException sqlex ) {
82             sqlex.printStackTrace();
83             output.append( sqlex.toString() );
84          }
85
86       } // end method actionPerformed
87
88    } // end class AddAuthorISBN
89
90    class FindAuthorISBN implements ActionListener {
91       private AuthorISBNScrollingPanel fields;
92       private JTextArea output;
93       private Connection connection;
94
95       public FindAuthorISBN( Connection c, AuthorISBNScrollingPanel f,
96                              JTextArea o )
97       {
98          connection = c;
99          fields = f;
100         output = o;
101      }
102
```

```
103    public void actionPerformed( ActionEvent e )
104    {
105       try {
106          if ( !fields.empty() ) {
107             Statement statement = connection.createStatement();
108             String query = "SELECT * FROM AuthorISBN WHERE ";
109
110             if ( fields.isbn.getText().equals( "" ) )
111                query += fields.authorID.getText().equals( "" ) ? "" :
112                   "AuthorID=" + fields.authorID.getText();
113             else {
114                query += "ISBN='" + fields.isbn.getText() + "'";
115                query += fields.authorID.getText().equals( "" ) ? "" :
116                   " AND AuthorID=" + fields.authorID.getText();
117             }
118
119             output.append( "\nSending query: " +
120                connection.nativeSQL( query ) + "\n" );
121
122             display( statement.executeQuery( query ) );
123             output.append( "\nQuery successful\n" );
124             statement.close();
125          }
126          else
127             fields.authorID.setText(
128                "Enter authorID here then press Find" );
129
130       } // end try
131
132       catch ( SQLException sqlex ) {
133          sqlex.printStackTrace();
134          output.append( sqlex.toString() );
135       }
136
137    } // end method actionPerformed
138
139    // Display results of query. If rs is null
140    public void display( ResultSet rs )
141    {
142       try {
143          // postition at first record
144          // if there are no records, display a message
145          if ( !rs.next() ) {
146             output.append( "\nNo records found\n" );
147             return;
148          }
149
150          String isbn = rs.getString( 1 );
151
152          if ( !isbn.equals( "" ) ) {
153
154             fields.isbn.setText( isbn );  // set isbn
155
156             // set authorISBN name
157             int authorID = rs.getInt( 2 );
```

```
158             fields.authorID.setText( String.valueOf( authorID ) );
159          }
160          else
161             output.append( "\nNo records found\n" );
162
163       } // end try
164
165       catch ( SQLException sqlex ) {
166          sqlex.printStackTrace();
167          output.append( sqlex.toString() );
168       }
169
170    } // end method display
171
172 } // end class FindAuthorISBN
173
174 class UpdateAuthorISBN implements ActionListener {
175    private AuthorISBNScrollingPanel fields;
176    private JTextArea output;
177    private Connection connection;
178
179    public UpdateAuthorISBN( Connection c, AuthorISBNScrollingPanel f,
180       JTextArea o )
181    {
182       connection = c;
183       fields = f;
184       output = o;
185    }
186
187    public void actionPerformed( ActionEvent e )
188    {
189       try {
190          Statement statement = connection.createStatement();
191
192          if ( ! fields.anyEmpty()) {
193             String query = "UPDATE AuthorISBN SET ISBN='" +
194                fields.isbn.getText() + "' WHERE AuthorID=" +
195                fields.authorID.getText();
196
197             output.append( "\nSending query: " +
198                connection.nativeSQL( query ) + "\n" );
199
200             if ( 1 == statement.executeUpdate( query ) )
201                output.append( "\nUpdate successful\n" );
202             else {
203                output.append( "\nUpdate failed\n" );
204                fields.clear();
205             }
206
207             statement.close();
208          }
209          else
210             output.append( "\nYou may only update an existing record. " +
211                "Use Find to locate the record, then modify the " +
212                "information and press Update.\n" );
```

```
213
214        } // end try
215
216        catch ( SQLException sqlex ) {
217           sqlex.printStackTrace();
218           output.append( sqlex.toString() );
219        }
220
221     } // end method actionPerformed
222
223  } // end class UpdateAuthorISBN
224
225  class AuthorISBNHelp implements ActionListener {
226     private JTextArea output;
227
228     public AuthorISBNHelp( JTextArea o )
229     {
230        output = o;
231     }
232
233     public void actionPerformed( ActionEvent e )
234     {
235        output.append( "\nClick Find to locate an AuthorISBN record.\n" +
236           "Click Add to insert a new AuthorISBN record.\nClick Update to" +
237           " update the information in an AuthorISBN record.\n" +
238           "Click Clear to empty the textfields.\n" );
239     }
240  }
241
242  class AuthorISBNDone implements ActionListener {
243     private EditAuthorISBN parent;
244
245     public AuthorISBNDone( EditAuthorISBN p ) { parent = p ; }
246
247     public void actionPerformed( ActionEvent e )
248     {
249        parent.setVisible( false );
250     }
251
252  } // end class AuthorISBNDone
253
254  class EditAuthorISBNsControlPanel extends JPanel {
255     private JButton findName, addName, updateName, clear, help, done;
256
257     public EditAuthorISBNsControlPanel( Connection c,
258        AuthorISBNScrollingPanel s, JTextArea t, EditAuthorISBN p )
259     {
260        setLayout( new GridLayout( 1, 6 ) );
261
262        findName = new JButton( "Find" );
263        findName.addActionListener( new FindAuthorISBN( c, s, t ) );
264        add( findName );
265
266        addName = new JButton( "Add" );
```

```
267         addName.addActionListener( new AddAuthorISBN( c, s, t ) );
268         add( addName );
269
270         updateName = new JButton( "Update" );
271         updateName.addActionListener( new UpdateAuthorISBN( c, s, t ) );
272         add( updateName );
273
274         clear = new JButton( "Clear" );
275         clear.addActionListener( new ClearAuthorISBNFields( s ) );
276         add( clear );
277
278         help = new JButton( "Help" );
279         help.addActionListener( new AuthorISBNHelp( t ) );
280         add( help );
281
282         done = new JButton( "Done" );
283         done.addActionListener( new AuthorISBNDone( p ) );
284         add( done );
285
286      } // end constructor EditAuthorISBNsControlPanel
287
288   } // end class EditAuthorISBNsControlPanel
289
290   class AuthorISBNScrollingPanel extends JPanel {
291      private JPanel labelPanel, fieldsPanel;
292      private String labels[] = { "ISBN:", "AuthorID:" };
293      JTextField isbn, authorID;
294
295      public AuthorISBNScrollingPanel()
296      {
297         // Label panel
298         labelPanel = new JPanel();
299         labelPanel.setLayout( new GridLayout( labels.length, 1 ) );
300
301         for ( int i = 0; i < labels.length; i++ )
302            labelPanel.add( new JLabel( labels[ i ] ) );
303
304         // TextField panel
305         fieldsPanel = new JPanel();
306         fieldsPanel.setLayout( new GridLayout( labels.length, 1 ) );
307         isbn = new JTextField( 20 );
308         fieldsPanel.add( isbn );
309         authorID = new JTextField( 20 );
310         fieldsPanel.add( authorID );
311
312         setLayout( new GridLayout( 1, 2 ) );
313         add( labelPanel );
314         add( fieldsPanel );
315      }
316
317      // returns true if all are empty
318      public boolean empty()
319      {
320        if ( isbn.getText().equals( "" ) && authorID.getText().equals( "" ) )
321           return true;
```

```
322
323        return false;
324     }
325
326     // returns true if any are empty
327     public boolean anyEmpty()
328     {
329       if ( isbn.getText().equals( "" ) || authorID.getText().equals( "" ) )
330          return true;
331
332        return false;
333     }
334
335     public void clear()
336     {
337        isbn.setText( "" );
338        authorID.setText( "" );
339     }
340
341  } // end class AuthorISBNScrollingPanel
342
343  class ClearAuthorISBNFields implements ActionListener {
344     private AuthorISBNScrollingPanel fields;
345
346     public ClearAuthorISBNFields( AuthorISBNScrollingPanel f )
347     {
348        fields = f;
349     }
350
351     public void actionPerformed( ActionEvent e )
352     {
353         fields.clear();
354     }
355
356  } // end class ClearAuthorISBNFields
```

```
 1  // Exercise 23.3 Solution: EditPanel.java
 2  import java.sql.*;
 3  import javax.swing.*;
 4  import java.awt.*;
 5  import java.awt.event.*;
 6  import java.util.*;
 7
 8  public class EditPanel {
 9     private JPanel thePanel;
10     private JComboBox editTable;
11     private JButton submitEdit;
12     private JLabel editInstructions;
13     private Connection connection;
14     private EditAuthor authorFrame;
15     private EditPublisher publisherFrame;
16     private EditTitle titleFrame;
17     private EditAuthorISBN authorISBNFrame;
```

```
18
19    public EditPanel( Connection c )
20    {
21       connection = c;
22
23       // Edit Panel
24       String editTables[] =
25          { "Authors", "Publishers", "AuthorISBN", "Titles" };
26       editTable = new JComboBox( editTables );
27       editTable.setBackground( Color.pink );
28       editTable.addItemListener(
29          new ItemListener() {
30             public void itemStateChanged( ItemEvent e )
31             {
32                updateEditInstructions( editTable.getSelectedIndex() );
33             }
34          }
35       );
36
37       editInstructions = new JLabel();
38       updateEditInstructions( 0 );
39       submitEdit = new JButton( "Edit Table" );
40       submitEdit.setBackground( Color.pink );
41       submitEdit.addActionListener(
42
43          new ActionListener() {
44
45             public void actionPerformed( ActionEvent e )
46             {
47                getFrame(editTable.getSelectedIndex() );
48             }
49          }
50       );
51
52       thePanel = new JPanel();
53       thePanel.setBackground( Color.pink );
54       thePanel.setLayout( new GridLayout( 2, 2 ) );
55       thePanel.add( new JLabel( "Edit" ) );
56       thePanel.add( editTable );
57       thePanel.add( editInstructions );
58       thePanel.add( submitEdit );
59
60       authorFrame = new EditAuthor( connection );
61       publisherFrame = new EditPublisher( connection );
62       titleFrame = new EditTitle( connection );
63       authorISBNFrame = new EditAuthorISBN( connection );
64
65    } // end constructor EditPanel
66
67    public JPanel getThePanel() { return thePanel; }
68
69    // handle edit requests
70    private void updateEditInstructions( int selection )
71    {
```

```
72          switch ( selection ) {
73             case 0:
74             default:  // add author
75                editInstructions.setText(
76                   "Press Edit Table to edit Author Table" );
77                return;
78             case 1:  // edit author
79                editInstructions.setText(
80                   "Press Edit Table to edit Publishers Table" );
81                return;
82             case 2:  // add title for author
83                editInstructions.setText(
84                   "Press Edit Table to edit AuthorISBN Table" );
85                return;
86             case 3:
87               editInstructions.setText(
88                   "Press Edit Table to edit Titles Table" );
89               return;
90
91          } // end switch
92
93       } // end method updateEditInstructions
94
95       // handle edit requests
96       private void getFrame( int selection )
97       {
98          switch ( selection ) {
99             case 0:
100               // add author
101               authorFrame.setVisible( true );
102               break;
103            case 1:
104               // edit publisher
105               publisherFrame.setVisible( true );
106               break;
107            case 2:
108               // edit authorISBN
109               authorISBNFrame.setVisible( true );
110               break;
111            case 3:
112               // edit Titles
113               titleFrame.setVisible( true );
114               break;
115         }
116
117         return;
118
119      } // end switch
120
121   } // end class EditPanel
```

```
1  // Exercise 23.3 solution: EditPublisher.java
2  import java.sql.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.swing.*;
6
7  public class EditPublisher extends JFrame {
8     private EditPublishersControlPanel controls;
9     private PublisherScrollingPanel scrollArea;
10    private JTextArea output;
11    private String url;
12    private Connection connect;
13    private JScrollPane textpane;
14
15    public EditPublisher( Connection connection )
16    {
17       super( "Edit Publisher Table" );
18       connect = connection;
19
20       Container c = getContentPane();
21
22       // Start screen layout
23       scrollArea = new PublisherScrollingPanel();
24       output = new JTextArea( 6, 30 );
25       c.setLayout( new BorderLayout() );
26       c.add( new JScrollPane( scrollArea ), BorderLayout.CENTER );
27       textpane = new JScrollPane( output );
28       c.add( textpane, BorderLayout.SOUTH );
29
30       // Complete screen layout
31       controls = new EditPublishersControlPanel(
32          connect, scrollArea, output, this );
33       c.add( controls, BorderLayout.NORTH );
34
35       setSize( 500, 250 );
36
37    } // end constructor EditPublisher
38
39 } // end class EditPublisher
40
41 class AddPublisher implements ActionListener {
42    private PublisherScrollingPanel fields;
43    private JTextArea output;
44    private Connection connection;
45
46    public AddPublisher(
47       Connection c, PublisherScrollingPanel f, JTextArea o )
48    {
49       connection = c;
50       fields = f;
51       output = o;
52    }
53
54    public void actionPerformed( ActionEvent e )
55    {
```

```
56          try {
57             Statement statement = connection.createStatement();
58
59             if ( !fields.name.getText().equals( "" ) ) {
60                String query = "INSERT INTO Publishers ( PublisherName ) " +
61                   "VALUES ('" + fields.name.getText() + "')";
62                output.append( "\nSending query: " +
63                               connection.nativeSQL( query ) + "\n" );
64
65                if ( statement.executeUpdate( query ) == 1 )
66                   output.append( "\nInsertion successful\n" );
67                else {
68                   output.append( "\nInsertion failed\n" );
69                   fields.clear();
70                }
71             }
72             else
73               output.append( "\nEnter publisher name and then press Add\n" );
74
75             statement.close();
76
77          } // end try
78
79          catch ( SQLException sqlex ) {
80             sqlex.printStackTrace();
81             output.append( sqlex.toString() );
82          }
83
84       } // end method actionPerformed
85
86    } // end class AddPublisher
87
88    class FindPublisher implements ActionListener {
89       private PublisherScrollingPanel fields;
90       private JTextArea output;
91       private Connection connection;
92
93       public FindPublisher(
94          Connection c, PublisherScrollingPanel f, JTextArea o )
95       {
96          connection = c;
97          fields = f;
98          output = o;
99       }
100
101      public void actionPerformed( ActionEvent e )
102      {
103         try {
104            if ( !fields.empty() ) {
105               Statement statement = connection.createStatement();
106               String query = "SELECT * FROM Publishers WHERE ";
107
108               boolean nameEmpty = fields.name.getText().equals( "" );
109               boolean idEmpty = fields.id.getText().equals( "" );
```

```
110
111             query += nameEmpty ? "" : "PublisherName = '" +
112                fields.name.getText() + "'";
113             query += !nameEmpty && !idEmpty  ? " AND " : "";
114             query += idEmpty  ? "" :
115                "PublisherID = " + fields.id.getText();
116
117             output.append( "\nSending query: " +
118                connection.nativeSQL( query ) + "\n" );
119             display( statement.executeQuery( query ) );
120             output.append( "\nQuery successful\n" );
121             statement.close();
122          }
123          else
124             fields.name.setText( "Enter name here then press Find" );
125
126       } // end try
127
128       catch ( SQLException sqlex ) {
129          sqlex.printStackTrace();
130          output.append( sqlex.toString() );
131       }
132
133    } // end method actionPerformed
134
135    // Display results of query. If rs is null
136    public void display( ResultSet rs )
137    {
138       try {
139          // postition at first record
140          // If there are no records, display a message
141          if ( !rs.next() ) {
142             output.append( "\nNo records found\n" );
143              return;
144          }
145
146          int id = rs.getInt( 1 );
147
148          if ( id != 0 ) {
149
150             // set id
151             fields.id.setText( String.valueOf( id ) );
152             fields.name.setText( rs.getString( 2 ) );
153          }
154          else
155             output.append( "\nNo records found\n" );
156
157       } // end try
158
159       catch ( SQLException sqlex ) {
160          sqlex.printStackTrace();
161          output.append( sqlex.toString() );
162       }
163
164    } // end method display
```

```
165
166 } // end class FindPublisher
167
168 class UpdatePublisher implements ActionListener {
169    private PublisherScrollingPanel fields;
170    private JTextArea output;
171    private Connection connection;
172
173    public UpdatePublisher(
174       Connection c, PublisherScrollingPanel f, JTextArea o )
175    {
176       connection = c;
177       fields = f;
178       output = o;
179    }
180
181    public void actionPerformed( ActionEvent e )
182    {
183       try {
184          Statement statement = connection.createStatement();
185
186          if ( !fields.id.getText().equals( "" ) ) {
187             String query = "UPDATE Publishers SET PublisherName='" +
188                fields.name.getText() + "' WHERE PublisherID=" +
189                fields.id.getText();
190
191             output.append( "\nSending query: " +
192                connection.nativeSQL( query ) + "\n" );
193
194             int result = statement.executeUpdate( query );
195
196             if ( result == 1 )
197                output.append( "\nUpdate successful\n" );
198             else {
199                output.append( "\nUpdate failed\n" );
200                fields.name.setText( "" );
201             }
202
203             statement.close();
204          }
205          else
206             output.append( "\nYou may only update an existing record. " +
207                "Use Find to locate the record, then modify the " +
208                "information and press Update.\n" );
209
210       } // end try
211
212       catch ( SQLException sqlex ) {
213          sqlex.printStackTrace();
214          output.append( sqlex.toString() );
215       }
216
217    } // end method actionPerformed
218
219 } // end class UpdatePublisher
```

```
220
221 class PublisherHelp implements ActionListener {
222    private JTextArea output;
223
224    public PublisherHelp( JTextArea o )
225    {
226       output = o;
227    }
228
229    public void actionPerformed( ActionEvent e )
230    {
231       output.append( "\nClick Find to locate an Publisher record.\n" +
232          "Click Add to insert a new Publisher record.\nClick Update to " +
233          "update the information in an Publisher record.\n" +
234          "Click Clear to empty the textfields.\n" );
235    }
236
237 } // end class PublisherHelp
238
239 class PublisherDone implements ActionListener {
240    private EditPublisher parent;
241
242    public PublisherDone( EditPublisher p ) { parent = p ; }
243
244    public void actionPerformed( ActionEvent e )
245    {
246       parent.setVisible( false );
247    }
248
249 } // end class PublisherDone
250
251 class EditPublishersControlPanel extends JPanel {
252    private JButton findName, addName, updateName, clear, help, done;
253
254    public EditPublishersControlPanel( Connection c,
255       PublisherScrollingPanel s, JTextArea t , EditPublisher p )
256    {
257       setLayout( new GridLayout( 1, 6 ) );
258
259       findName = new JButton( "Find" );
260       findName.addActionListener( new FindPublisher( c, s, t ) );
261       add( findName );
262
263       addName = new JButton( "Add" );
264       addName.addActionListener( new AddPublisher( c, s, t ) );
265       add( addName );
266
267       updateName = new JButton( "Update" );
268       updateName.addActionListener( new UpdatePublisher( c, s, t ) );
269       add( updateName );
270
271       clear = new JButton( "Clear" );
272       clear.addActionListener( new ClearPublisherFields( s ) );
273       add( clear );
```

```
274
275       help = new JButton( "Help" );
276       help.addActionListener( new PublisherHelp( t ) );
277       add( help );
278
279       done = new JButton( "Done" );
280       done.addActionListener( new PublisherDone( p ) );
281       add( done );
282
283    } // end constructor EditPublishersControlPanel
284
285 } // end class EditPublishersControlPanel
286
287 class PublisherScrollingPanel extends JPanel {
288    private JPanel labelPanel, fieldsPanel;
289    private String labels[] = { "Publisher ID:", "Publisher Name:" };
290    JTextField id, name;
291
292    public PublisherScrollingPanel()
293    {
294       // Label panel
295       labelPanel = new JPanel();
296       labelPanel.setLayout( new GridLayout( labels.length, 1 ) );
297
298       ImageIcon ii = new ImageIcon( "images/icon.jpg" );
299
300       for ( int i = 0; i < labels.length; i++ )
301          labelPanel.add( new JLabel( labels[ i ], ii, 0 ) );
302
303       // TextField panel
304       fieldsPanel = new JPanel();
305       fieldsPanel.setLayout( new GridLayout( labels.length, 1 ) );
306       id = new JTextField( 20 );
307       id.setEditable( false );
308       fieldsPanel.add( id );
309       name = new JTextField( 20 );
310       fieldsPanel.add( name );
311
312       setLayout( new GridLayout( 1, 2 ) );
313       add( labelPanel );
314       add( fieldsPanel );
315
316    } // end constructor PublisherScrollingPanel
317
318     public void clear()
319     {
320        id.setText( "" );
321        name.setText( "" );
322     }
323
324     // returns if fields are all empty (except ID)
325     public boolean empty()
326     {
327        if ( name.getText().equals( "" ) )
328           return true;
```

```
329
330         return false;
331      }
332
333   } // end class PublisherScrollingPanel
334
335   class ClearPublisherFields implements ActionListener {
336      private PublisherScrollingPanel fields;
337
338      public ClearPublisherFields( PublisherScrollingPanel f )
339      {
340         fields = f;
341      }
342
343      public void actionPerformed( ActionEvent e )
344      {
345         fields.clear();
346      }
347
348   } // end class ClearPublisherFields
```

```
 1  // Exercise 23.3 solution: EditTitle.java
 2  import java.sql.*;
 3  import java.awt.*;
 4  import java.awt.event.*;
 5  import javax.swing.*;
 6
 7  public class EditTitle extends JFrame {
 8     private EditTitlesControlPanel controls;
 9     private TitleScrollingPanel scrollArea;
10     private JTextArea output;
11     private String url;
12     private Connection connect;
13     private JScrollPane textpane;
14
15     public EditTitle( Connection connection )
16     {
17        super( "Edit Title Table" );
18        connect = connection;
19
20        Container c = getContentPane();
21
22        // Start screen layout
23        scrollArea = new TitleScrollingPanel();
24        output = new JTextArea( 6, 30 );
25        c.setLayout( new BorderLayout() );
26        c.add( new JScrollPane( scrollArea ), BorderLayout.CENTER );
27        textpane = new JScrollPane( output );
28        c.add( textpane, BorderLayout.SOUTH );
29
30        // Complete screen layout
31        controls = new EditTitlesControlPanel(
32           connect, scrollArea, output, this );
```

```
33          c.add( controls, BorderLayout.NORTH );
34
35          setSize( 500, 320 );
36
37       } // end constructor EditTitle
38
39    } // end class Edittitle
40
41    class AddTitle implements ActionListener {
42       private TitleScrollingPanel fields;
43       private JTextArea output;
44       private Connection connection;
45
46       public AddTitle( Connection c, TitleScrollingPanel f, JTextArea o )
47       {
48          connection = c;
49          fields = f;
50          output = o;
51       }
52
53       public void actionPerformed( ActionEvent e )
54       {
55          try {
56             Statement statement = connection.createStatement();
57
58             if ( !fields.isbn.getText().equals( "" ) &&
59                !fields.title.getText().equals( "" ) ) {
60                String query = "INSERT INTO Titles ( ISBN, Title, " +
61                   "EditionNumber, Copyright, Price, PublisherID, ImageFile" +
62                   " ) VALUES ('" + fields.isbn.getText() + "', '" +
63                   fields.title.getText() + "', '" +
64                   fields.editionNumber.getText() + "', '" +
65                   fields.copyright.getText() + "', '" +
66                   fields.price.getText() + "', '" +
67                   fields.publisherID.getText() + "','" +
68                   fields.imageFile.getText() + "')";
69                output.append( "\nExecuting Update: " +
70                   connection.nativeSQL( query ) + "\n" );
71
72                if ( statement.executeUpdate( query ) == 1 ) {
73                   output.append( "\nInsertion successful. " +
74                      "Adding entry to AuthorISBN Table\n" );
75
76                   String authorID =
77                      JOptionPane.showInputDialog( "Enter Author ID:" );
78                   query = "INSERT INTO AuthorISBN (ISBN, AuthorID ) VALUES" +
79                      " ('" + fields.isbn.getText() + "','" + authorID + "')";
80
81                   output.append( "\nExecuting Update: " +
82                      connection.nativeSQL( query ) + "\n" );
83
84                   if ( 1 == statement.executeUpdate( query ) )
85                      output.append( "\nInsertion succeeded\n" );
86                   else {
```

```
87                      output.append( "\nInsertion failed\n" );
88                      fields.clear();
89                   }
90                }
91                else {
92                   output.append( "\nInsertion failed\n" );
93                   fields.clear();
94                }
95             }
96             else
97                output.append( "\nEnter at least first and " +
98                   "last name then press Add\n" );
99
100            statement.close();
101
102         } // end try
103
104         catch ( SQLException sqlex ) {
105            sqlex.printStackTrace();
106            output.append( sqlex.toString() );
107         }
108
109      } // end method actionPerformed
110
111   } // end class AddTitle
112
113   class FindTitle implements ActionListener {
114      private TitleScrollingPanel fields;
115      private JTextArea output;
116      private Connection connection;
117
118      public FindTitle( Connection c, TitleScrollingPanel f, JTextArea o )
119      {
120         connection = c;
121         fields = f;
122         output = o;
123      }
124
125      public void actionPerformed( ActionEvent e )
126      {
127         try {
128            if ( !fields.empty() ) {
129               Statement statement = connection.createStatement();
130               String query = "SELECT ISBN, Title, EditionNumber, Copyright,"
131                  + " PublisherID, ImageFile, Price FROM Titles WHERE ";
132               boolean needAnd = false;
133
134               query += fields.title.getText().equals( "" )  ? "" :
135                   "title ='" + fields.title.getText() + "'";
136
137               if ( !fields.title.getText().equals( "" ) )
138                  needAnd = true;
139
140               query += fields.isbn.getText().equals( "" )  ? "" :
```

```
141                ( needAnd ? " AND isbn = '" + fields.isbn.getText() + "'"
142                :"isbn = '" + fields.isbn.getText() + "'" );
143
144             if ( !fields.isbn.getText().equals( "" ) )
145                needAnd = true;
146
147             query += fields.editionNumber.getText().equals( "" ) ? "" :
148                ( needAnd ? " AND EditionNumber='" +
149                fields.editionNumber.getText() + "'" : "EditionNumber='" +
150                fields.editionNumber.getText() + "'" );
151
152             if ( !fields.editionNumber.getText().equals( "" ) )
153                needAnd = true;
154
155             query += fields.copyright.getText().equals( "" ) ? "" :
156                ( needAnd ? " AND Copyright='" +
157                fields.copyright.getText() + "'" : "Copyright='" +
158                fields.copyright.getText() + "'" );
159
160             if ( !fields.copyright.getText().equals( "" ) )
161                needAnd = true;
162
163             query += fields.price.getText().equals( "" )  ? "" :
164                ( needAnd ? " AND Price = '" + fields.price.getText() +
165                "'" : "Price = '" + fields.price.getText() + "'" );
166
167             if ( !fields.price.getText().equals( "" ) )
168                needAnd = true;
169
170             query += fields.imageFile.getText().equals( "" )  ? "" :
171                ( needAnd ? " AND ImageFile = '" +
172                fields.imageFile.getText() + "'" : "ImageFile = '" +
173                fields.imageFile.getText() + "'" );
174
175             if ( !fields.imageFile.getText().equals( "" ) )
176                 needAnd = true;
177
178             query += fields.publisherID.getText().equals( "" )  ? "" :
179                ( needAnd ? " AND PublisherID=" +
180                fields.publisherID.getText() : "PublisherID=" +
181                fields.publisherID.getText() );
182
183             output.append( "\nSending query: " +
184                connection.nativeSQL( query ) + "\n" );
185
186             ResultSet rs = statement.executeQuery( query );
187             display( rs );
188             output.append( "\nQuery successful\n" );
189             statement.close();
190          }
191          else
192             fields.title.setText( "Enter title here then press Find" );
193
194       } // end try
```

```
195
196        catch ( SQLException sqlex ) {
197           sqlex.printStackTrace();
198           output.append( sqlex.toString() );
199        }
200
201     } // end method actionPerformed
202
203     // Display results of query. If rs is null
204     public void display( ResultSet rs )
205     {
206        try {
207           // postition at first record
208           // If there are no records, display a message
209           if ( ! rs.next() ) {
210              output.append( "\nNo record found\n" );
211              return;
212           }
213
214           String isbnNumber = rs.getString( 1 );
215
216           if ( !isbnNumber.equals( "" ) ) {
217              fields.isbn.setText( isbnNumber );
218              fields.title.setText( rs.getString( 2 ) );
219              fields.editionNumber.setText( rs.getString( 3 ) );
220              fields.copyright.setText( rs.getString( 4 ) );
221              fields.publisherID.setText( rs.getString( 5 ) );
222              fields.imageFile.setText( rs.getString( 6 ) );
223              fields.price.setText( rs.getDouble( 7 ) + "" );
224           }
225           else
226              output.append( "\nNo record found\n" );
227
228        } // end try
229
230        catch ( SQLException sqlex ) {
231           sqlex.printStackTrace();
232           output.append( sqlex.toString() );
233        }
234
235     } // end method display
236
237  } // end class FindTitle
238
239  class UpdateTitle implements ActionListener {
240     private TitleScrollingPanel fields;
241     private JTextArea output;
242     private Connection connection;
243
244     public UpdateTitle( Connection c, TitleScrollingPanel f, JTextArea o )
245     {
246        connection = c;
247        fields = f;
248        output = o;
249     }
```

```
250
251    public void actionPerformed( ActionEvent e )
252    {
253       try {
254          Statement statement = connection.createStatement();
255
256          if ( ! fields.isbn.getText().equals( "" ) ) {
257             String query = "UPDATE Titles SET Title='" +
258                fields.title.getText() + "', EditionNumber='" +
259                fields.editionNumber.getText() + "', Copyright='" +
260                fields.copyright.getText() + "', PublisherID='" +
261                fields.publisherID.getText() + "', ImageFile='" +
262                fields.imageFile.getText() + "', Price='" +
263                fields.price.getText() + "' WHERE ISBN='" +
264                fields.isbn.getText() + "'";
265
266             output.append( "\nSending query: " +
267                connection.nativeSQL( query ) + "\n" );
268
269             int result = statement.executeUpdate( query );
270
271             if ( result == 1 )
272                output.append( "\nUpdate successful\n" );
273             else {
274                output.append( "\nUpdate failed\n" );
275                fields.clear();
276             }
277
278             statement.close();
279          }
280          else
281             output.append( "\nYou may only update an existing record. " +
282                "Use Find to locate the record, then modify the " +
283                "information and press Update.\n" );
284
285       } // end try
286
287       catch ( SQLException sqlex ) {
288          sqlex.printStackTrace();
289          output.append( sqlex.toString() );
290       }
291
292    } // end method actionPerformed
293
294 } // end class UpdateTitle
295
296 class TitleHelp implements ActionListener {
297    private JTextArea output;
298
299    public TitleHelp( JTextArea o )
300    {
301       output = o;
302    }
303
```

```
304    public void actionPerformed( ActionEvent e )
305    {
306       output.append( "\nClick Find to locate an Title record.\n" +
307          "Click Add to insert a new Title record.\n" +
308          "Click Update to update the information in an Title record.\n" +
309          "Click Clear to empty the textfields.\n" );
310    }
311
312 } // end class TitleHelp
313
314 class TitleDone implements ActionListener {
315    private EditTitle parent;
316
317    public TitleDone( EditTitle p ) { parent = p ; }
318
319    public void actionPerformed( ActionEvent e )
320    {
321       parent.setVisible( false );
322    }
323
324 } // end class TitleDone
325
326 class EditTitlesControlPanel extends JPanel {
327    private JButton findName, addName, updateName, clear, help, done;
328
329    public EditTitlesControlPanel( Connection c, TitleScrollingPanel s,
330       JTextArea t , EditTitle p )
331    {
332       setLayout( new GridLayout( 1, 6 ) );
333
334       findName = new JButton( "Find" );
335       findName.addActionListener( new FindTitle( c, s, t ) );
336       add( findName );
337
338       addName = new JButton( "Add" );
339       addName.addActionListener( new AddTitle( c, s, t ) );
340       add( addName );
341
342       updateName = new JButton( "Update" );
343       updateName.addActionListener( new UpdateTitle( c, s, t ) );
344       add( updateName );
345
346       clear = new JButton( "Clear" );
347       clear.addActionListener( new ClearTitleFields( s ) );
348       add( clear );
349
350       help = new JButton( "Help" );
351       help.addActionListener( new TitleHelp( t ) );
352       add( help );
353
354       done = new JButton( "Done" );
355       done.addActionListener( new TitleDone( p ) );
356       add( done );
357
358    } // end constructor EditTitlescontrolPanel
```

```
359
360 } // end class EditTitlesControlPanel
361
362 class TitleScrollingPanel extends JPanel {
363    private JPanel labelPanel, fieldsPanel;
364    private String labels[] = { "ISBN:", "Title:", "Edition Number:",
365       "Copyright:", "Price:", "PublisherID:", "Image File Name:" };
366    JTextField isbn, title, editionNumber, copyright, price,
367       publisherID, imageFile;
368
369    public TitleScrollingPanel()
370    {
371       // Label panel
372       labelPanel = new JPanel();
373       labelPanel.setLayout( new GridLayout( labels.length, 1 ) );
374
375       ImageIcon ii = new ImageIcon( "images/icon.jpg" );
376
377       for ( int i = 0; i < labels.length; i++ )
378          labelPanel.add( new JLabel( labels[ i ], ii, 0 ) );
379
380       // TextField panel
381       fieldsPanel = new JPanel();
382       fieldsPanel.setLayout( new GridLayout( labels.length, 1 ) );
383       isbn = new JTextField( 20 );
384       fieldsPanel.add( isbn );
385
386       title = new JTextField( 20 );
387       fieldsPanel.add( title );
388
389       editionNumber = new JTextField( 20 );
390       fieldsPanel.add( editionNumber );
391
392       copyright = new JTextField( 20 );
393       fieldsPanel.add( copyright );
394
395       price = new JTextField( 20 );
396       fieldsPanel.add( price );
397
398       publisherID = new JTextField( 20 );
399       fieldsPanel.add( publisherID );
400
401       imageFile = new JTextField( 20 );
402       fieldsPanel.add( imageFile );
403
404       setLayout( new GridLayout( 1, 2 ) );
405       add( labelPanel );
406       add( fieldsPanel );
407
408    } // end constructor TitleScrollingPanel
409
410    public void clear()
411    {
412       isbn.setText( "" );
```

```
413         title.setText( "" );
414         editionNumber.setText( "" );
415         copyright.setText( "" );
416         price.setText( "" );
417         publisherID.setText( "" );
418         imageFile.setText( "" );
419      }
420
421      public boolean empty()
422      {
423         if ( isbn.getText().equals( "" ) && title.getText().equals( "" ) &&
424              editionNumber.getText().equals( "" ) &&
425              copyright.getText().equals( "" ) &&
426              price.getText().equals( "" ) &&
427              publisherID.getText().equals( "" ) &&
428              imageFile.getText().equals( "" ) )
429            return true;
430
431         return false;
432      }
433
434   } // end class TitleScrollingPanel
435
436   class ClearTitleFields implements ActionListener {
437      private TitleScrollingPanel fields;
438
439      public ClearTitleFields( TitleScrollingPanel f )
440      {
441         fields = f;
442      }
443
444      public void actionPerformed( ActionEvent e )
445      {
446         fields.clear();
447      }
448
449   } // end class ClearTitleFields
```

23.4 In Section 10.7, we introduced an employee-payroll hierarchy to calculate each employee's payroll. In this exercise, we provide a database of employees that corresponds to the employee-payroll hierarchy. (A SQL script to create the employee database is provided with the examples for this chapter on the CD that accompanies this text and on our Web site `www.deitel.com`.) Write an application that allows:

a) Add employees to the Employee table.
b) For each employee added to the table, add payroll to the corresponding table. For example, for a salaried employee add the payroll information to the salariedEmployees table.

ANS:

```
1  // Exercise 23.4 solution: AddEmployees.java
2  import java.sql.*;
3  import java.awt.*;
4  import java.awt.event.*;
```

```
5  import java.util.*;
6  import javax.swing.*;
7
8  public class AddEmployees extends JFrame {
9     private Connection connection;
10    private Statement statement;
11    private ResultSet resultSet;
12    private ResultSetMetaData rsMetaData;
13    private Container container;
14    private JTable table;
15    private JTextField input;
16    private JButton addSalariedEmployee, addCommissionEmployee,
17       addBasePlusCommissionEmployee, addHourlyEmployee;
18
19    public AddEmployees()
20    {
21       super( "Add Employees" );
22
23       // The URL specifying the books database to which this program
24       // connects to using JDBC
25       String url = "jdbc:db2j:employees";
26
27       // Load the driver to allow connection to the database
28       try {
29          Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
30
31          connection = DriverManager.getConnection( url );
32       }
33       catch ( ClassNotFoundException cnfex ) {
34          System.err.println( "Failed to load JDBC driver." );
35          cnfex.printStackTrace();
36          System.exit( 1 );  // terminate program
37       }
38       catch ( SQLException sqlex ) {
39          System.err.println( "Unable to connect" );
40          sqlex.printStackTrace();
41          System.exit( 1 );  // terminate program
42       }
43
44       // if connected to database, set up GUI
45       JPanel topPanel = new JPanel();
46       topPanel.setLayout( new FlowLayout() );
47       topPanel.add( new JLabel( "Enter query to insert employees:" ) );
48
49       input = new JTextField( 50 );
50       topPanel.add( input );
51       input.addActionListener(
52
53          new ActionListener() {
54
55             public void actionPerformed( ActionEvent e )
56             {
57                addEmployee( input.getText() );
58             }
```

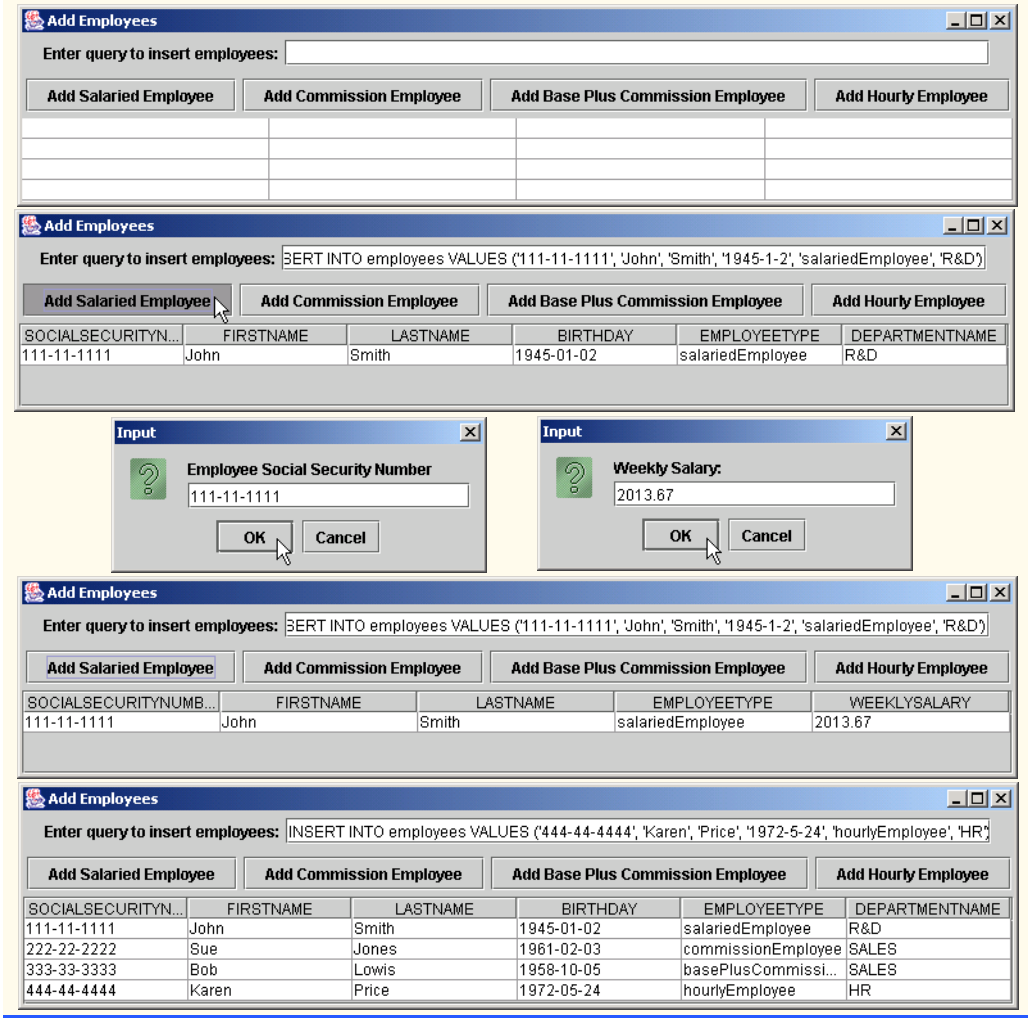
```
59             }
60          );
61
62          // create four buttons that allow user to add specific employee
63          JPanel centerPanel = new JPanel();
64          centerPanel.setLayout( new FlowLayout() );
65
66          addSalariedEmployee = new JButton( "Add Salaried Employee" );
67          addSalariedEmployee.addActionListener( new ButtonHandler() );
68
69          addCommissionEmployee = new JButton( "Add Commission Employee" );
70          addCommissionEmployee.addActionListener( new ButtonHandler() );
71
72          addBasePlusCommissionEmployee =
73             new JButton( "Add Base Plus Commission Employee" );
74          addBasePlusCommissionEmployee.addActionListener(
75             new ButtonHandler() );
76
77          addHourlyEmployee = new JButton( "Add Hourly Employee" );
78          addHourlyEmployee.addActionListener( new ButtonHandler() );
79
80          // add four buttons to centerPanel
81          centerPanel.add( addSalariedEmployee );
82          centerPanel.add( addCommissionEmployee );
83          centerPanel.add( addBasePlusCommissionEmployee );
84          centerPanel.add( addHourlyEmployee );
85
86          JPanel inputPanel = new JPanel();
87          inputPanel.setLayout( new BorderLayout() );
88          inputPanel.add( topPanel, BorderLayout.NORTH );
89          inputPanel.add( centerPanel, BorderLayout.CENTER );
90
91          table = new JTable( 4, 4 );
92
93          container = getContentPane();
94          container.setLayout( new BorderLayout() );
95          container.add( inputPanel, BorderLayout.NORTH );
96          container.add( table, BorderLayout.CENTER );
97
98          getTable();
99
100         setSize( 800, 300 );
101         setVisible( true );
102
103      } // end constructor AddEmployees
104
105      private void getTable()
106      {
107         try {
108            statement = connection.createStatement();
109            resultSet = statement.executeQuery( "SELECT * FROM employees" );
110            displayResultSet( resultSet );
111         }
112         catch ( SQLException sqlex ) {
```

```
113            sqlex.printStackTrace();
114         }
115      }
116
117      private void addEmployee( String query )
118      {
119         try {
120            statement = connection.createStatement();
121            statement.executeUpdate( query );
122            getTable();
123         }
124         catch ( SQLException sqlex ) {
125            sqlex.printStackTrace();
126         }
127      }
128
129      private void displayResultSet( ResultSet rs ) throws SQLException
130      {
131         // position to first record
132         boolean moreRecords = rs.next();
133
134         // if there are no records, display a message
135         if ( !moreRecords ) {
136            JOptionPane.showMessageDialog( this,
137               "ResultSet contained no records" );
138            return;
139         }
140
141         Vector columnHeads = new Vector();
142         Vector rows = new Vector();
143
144         try {
145            // get column heads
146            ResultSetMetaData rsmd = rs.getMetaData();
147
148            for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
149               columnHeads.addElement( rsmd.getColumnName( i ) );
150
151            // get row data
152            do {
153               rows.addElement( getNextRow( rs, rsmd ) );
154            } while ( rs.next() );
155
156            // display table with ResultSet contents
157            table = new JTable( rows, columnHeads );
158            JScrollPane scroller = new JScrollPane( table );
159            container.remove( 1 );
160            container.add( scroller, BorderLayout.CENTER );
161            container.validate();
162
163         } // end try
164
165         catch ( SQLException sqlex ) {
166            sqlex.printStackTrace();
167         }
```

```
168
169    } // end method displayResultSet
170
171    private Vector getNextRow( ResultSet rs,
172       ResultSetMetaData rsmd ) throws SQLException
173    {
174       Vector currentRow = new Vector();
175
176       for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
177          switch( rsmd.getColumnType( i ) ) {
178             case Types.VARCHAR:
179             case Types.LONGVARCHAR:
180                currentRow.addElement( rs.getString( i ) );
181                break;
182             case Types.INTEGER:
183                currentRow.addElement( new Long( rs.getLong( i ) ) );
184                break;
185             case Types.REAL:
186                currentRow.addElement( new Float( rs.getDouble( i ) ) );
187                break;
188             case Types.DATE:
189                currentRow.addElement( rs.getDate( i ) );
190                break;
191             default:
192                System.out.println( "Type was: " +
193                   rsmd.getColumnTypeName( i ) );
194          }
195
196       return currentRow;
197
198    } // end method getNextRow
199
200    public void shutDown()
201    {
202       try {
203          connection.close();
204       }
205       catch ( SQLException sqlex ) {
206          System.err.println( "Unable to disconnect" );
207          sqlex.printStackTrace();
208       }
209    }
210
211    public static void main( String[] args )
212    {
213       final AddEmployees application = new AddEmployees();
214       application.addWindowListener(
215
216          new WindowAdapter() {
217
218             public void windowClosing( WindowEvent e )
219             {
220                application.shutDown();
221                System.exit( 0 );
222             }
```

```
223          }
224       );
225    }
226
227    // inner class ButtonHandler handles button event
228    private class ButtonHandler implements ActionListener {
229
230       public void actionPerformed( ActionEvent event )
231       {
232          String socialSecurityNumber = JOptionPane.showInputDialog(
233             "Employee Social Security Number" );
234          String insertQuery = "", displayQuery = "";
235
236          // add salaried employee to table salariedEmployee
237          if ( event.getSource() == addSalariedEmployee ) {
238             double weeklySalary = Double.parseDouble(
239                JOptionPane.showInputDialog( "Weekly Salary:" ) );
240             insertQuery = "INSERT INTO salariedEmployees VALUES ( '" +
241                socialSecurityNumber + "', '" + weeklySalary + "', '0' )";
242             displayQuery = "SELECT employees.socialSecurityNumber, " +
243                "employees.firstName, employees.lastName, " +
244                "employees.employeeType, salariedEmployees.weeklySalary" +
245                " FROM employees, salariedEmployees WHERE " +
246                "employees.socialSecurityNumber = " +
247                "salariedEmployees.socialSecurityNumber";
248          }
249
250          // add commission employee to table commissionEmployee
251          else if ( event.getSource() == addCommissionEmployee ) {
252             int grossSales = Integer.parseInt(
253                JOptionPane.showInputDialog( "Gross Sales:" ) );
254             double commissionRate = Double.parseDouble(
255                JOptionPane.showInputDialog( "Commission Rate:" ) );
256             insertQuery = "INSERT INTO commissionEmployees VALUES ( '" +
257                socialSecurityNumber + "', '" + grossSales + "', '" +
258                commissionRate + "', '0' )";
259             displayQuery = "SELECT employees.socialSecurityNumber, " +
260                "employees.firstName, employees.lastName, " +
261                "employees.employeeType, commissionEmployees.grossSales," +
262                " commissionEmployees.commissionRate FROM employees, " +
263               "commissionEmployees WHERE employees.socialSecurityNumber="
264                + "commissionEmployees.socialSecurityNumber";
265          }
266
267          // add base plus commission employee to table
268          // basePlusCommissionEmployee
269          else if ( event.getSource() == addBasePlusCommissionEmployee ) {
270             int grossSales = Integer.parseInt(
271                JOptionPane.showInputDialog( "Gross Sales:" ) );
272             double commissionRate = Double.parseDouble(
273                JOptionPane.showInputDialog( "Commission Rate:" ) );
274             double baseSalary = Double.parseDouble(
275                JOptionPane.showInputDialog( "Base Salary:" ) );
276             insertQuery = "INSERT INTO basePlusCommissionEmployees " +
```

```
277                "VALUES ( '" + socialSecurityNumber + "', '" + grossSales +
278                "', '" + commissionRate + "', '" + baseSalary + "', '0' )";
279             displayQuery = "SELECT employees.socialSecurityNumber, " +
280                "employees.firstName, employees.lastName, employees." +
281                "employeeType, basePlusCommissionEmployees.baseSalary, " +
282                "basePlusCommissionEmployees.grossSales, basePlus" +
283                "CommissionEmployees.commissionRate FROM employees, " +
284                "basePlusCommissionEmployees WHERE " +
285                "employees.socialSecurityNumber = " +
286                "basePlusCommissionEmployees.socialSecurityNumber";
287          }
288
289          // add hourly employee to table hourlyEmployee
290          else {
291             int hours = Integer.parseInt(
292                JOptionPane.showInputDialog( "Hours:" ) );
293             double wage = Double.parseDouble(
294                JOptionPane.showInputDialog( "Wage:" ) );
295             insertQuery = "INSERT INTO hourlyEmployees VALUES ( '" +
296                socialSecurityNumber + "', '" + hours + "', '" + wage +
297                "', '0' )";
298             displayQuery = "SELECT employees.socialSecurityNumber, " +
299                "employees.firstName, employees.lastName, " +
300                "employees.employeeType, hourlyEmployees.hours, " +
301                "hourlyEmployees.wage FROM employees, hourlyEmployees " +
302                "WHERE employees.socialSecurityNumber = " +
303                "hourlyEmployees.socialSecurityNumber";
304          }
305
306          // execute insert query and display employee info
307          try {
308             statement = connection.createStatement();
309             statement.executeUpdate( insertQuery );
310
311             // display the employee info
312             statement = connection.createStatement();
313             resultSet = statement.executeQuery( displayQuery );
314             displayResultSet( resultSet );
315          }
316          catch ( SQLException exception ) {
317             exception.printStackTrace();
318          }
319
320       } // end method actionPerformed
321
322    } // end inner class ButtonHandler
323
324 } // end class AddEmployees
```



23.5 Write an application that provides a `JComboBox` and a `JTextArea` to allow the user to perform a query that is either selected from the `JComboBox` or defined in the `JTextArea`. Sample predified querys are:

a) Select all employees working in Department SALES.
b) Select hourly employees working over 30 hours.
c) Select all comission employees in descending order of the comission rate.

ANS:

```
1  // Exercise 23.5 Solution: DisplayQueryResults.java
2  import java.sql.*;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
```

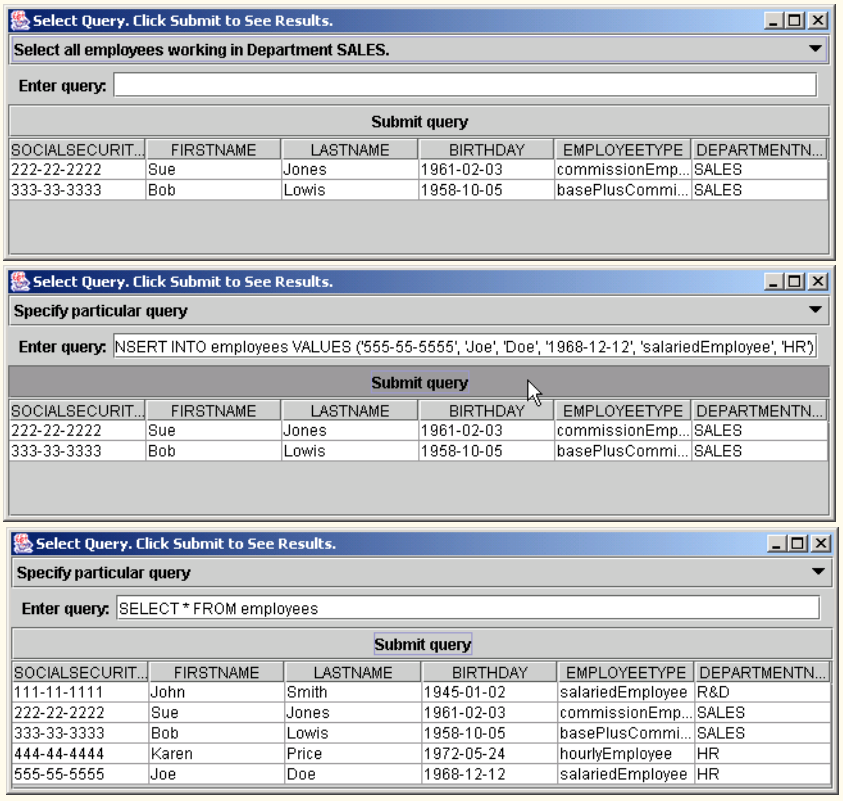
```
 6  import java.util.*;
 7
 8  public class DisplayQueryResults extends JFrame {
 9     private Connection connection;
10     private Statement statement;
11     private ResultSet resultSet;
12     private ResultSetMetaData rsMetaData;
13     private JTable table;
14     private JComboBox inputQuery;
15     private JButton submitQuery;
16     private JTextField input;
17
18     public DisplayQueryResults()
19     {
20        super( "Select Query. Click Submit to See Results." );
21
22        // The URL specifying the employees database to which this program
23        // connects to using JDBC
24        String url = "jdbc:db2j:employees";
25
26        // Load the driver to allow connection to the database
27        try {
28           Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
29
30           connection = DriverManager.getConnection( url );
31        }
32        catch ( ClassNotFoundException cnfex ) {
33           System.err.println( "Failed to load JDBC driver." );
34           cnfex.printStackTrace();
35           System.exit( 1 );  // terminate program
36        }
37        catch ( SQLException sqlex ) {
38           System.err.println( "Unable to connect" );
39           sqlex.printStackTrace();
40           System.exit( 1 );  // terminate program
41        }
42
43        String queries[] = { "Select all employees working in Department " +
44           "SALES.", "Select hourly employees working over 30 hours.",
45           "Select all comission employees in descending order of the " +
46           "comission rate.", "Specify particular query" };
47
48        // If connected to database, set up GUI
49        inputQuery = new JComboBox( queries );
50
51        submitQuery = new JButton( "Submit query" );
52        submitQuery.addActionListener(
53
54           new ActionListener() {
55
56              public void actionPerformed( ActionEvent e )
57              {
58                 getTable();
59              }
```

```
60          }
61       );
62
63       JPanel topPanel = new JPanel();
64       input = new JTextField( 50 );
65       input.addActionListener(
66
67          new ActionListener() {
68
69             public void actionPerformed( ActionEvent e )
70             {
71                try {
72                   String query = input.getText();
73                   statement = connection.createStatement();
74                   resultSet = statement.executeQuery( query );
75                   displayResultSet( resultSet );
76                }
77                catch ( SQLException sqlex ) {
78                   sqlex.printStackTrace();
79                }
80             }
81          }
82       );
83
84       JPanel centerPanel = new JPanel();
85       centerPanel.setLayout( new FlowLayout() );
86       centerPanel.add( new JLabel( "Enter query:" ) );
87       centerPanel.add( input );
88       topPanel.setLayout( new BorderLayout() );
89       topPanel.add( inputQuery, BorderLayout.NORTH );
90       topPanel.add( centerPanel, BorderLayout.CENTER );
91       topPanel.add( submitQuery, BorderLayout.SOUTH );
92
93       table = new JTable( 4, 4 );
94
95       Container c = getContentPane();
96       c.setLayout( new BorderLayout() );
97       c.add( topPanel, BorderLayout.NORTH );
98       c.add( table, BorderLayout.CENTER );
99
100      getTable();
101
102      setSize( 650, 200 );
103      setVisible( true );
104
105   } // end constructor DisplayQueryResult
106
107   private void getTable()
108   {
109      try {
110         int selection = inputQuery.getSelectedIndex();
111         String query = null;
112
113         switch ( selection ) {
```

```
114             case 0:
115                query = "SELECT * FROM employees WHERE " +
116                   "departmentName = 'SALES'";
117                break;
118             case 1:
119                query = "SELECT * FROM hourlyEmployees WHERE hours >= 30";
120                break;
121             case 2:
122                query = "SELECT * FROM commissionEmployees ORDER BY " +
123                   "commissionRate DESC";
124                break;
125             case 3:
126                query = input.getText();
127                   break;
128          }
129
130          statement = connection.createStatement();
131
132          if ( query.substring( 0, 6 ).equals( "SELECT" ) ) {
133             resultSet = statement.executeQuery( query );
134             displayResultSet( resultSet );
135          }
136
137          else statement.executeUpdate( query );
138
139       } // end try
140
141       catch ( SQLException sqlex ) {
142          sqlex.printStackTrace();
143       }
144
145    } // end method getTable
146
147    private void displayResultSet( ResultSet rs ) throws SQLException
148    {
149       // position to first record
150       boolean moreRecords = rs.next();
151
152       // If there are no records, display a message
153       if ( !moreRecords ) {
154          JOptionPane.showMessageDialog( this,
155             "ResultSet contained no records" );
156          setTitle( "No records to display" );
157          return;
158       }
159
160       Vector columnHeads = new Vector();
161       Vector rows = new Vector();
162
163       try {
164          // get column heads
165          ResultSetMetaData rsmd = rs.getMetaData();
166
167          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
168             columnHeads.addElement( rsmd.getColumnName( i ) );
```

```
169
170          // get row data
171          do {
172             rows.addElement( getNextRow( rs, rsmd ) );
173          } while ( rs.next() );
174
175          // display table with ResultSet contents
176          table = new JTable( rows, columnHeads );
177          JScrollPane scroller = new JScrollPane( table );
178          Container c = getContentPane();
179          c.remove( 1 );
180          c.add( scroller, BorderLayout.CENTER );
181          c.validate();
182
183       } // end try
184
185       catch ( SQLException sqlex ) {
186          sqlex.printStackTrace();
187       }
188
189    } // end method displayResultSet
190
191    private Vector getNextRow( ResultSet rs,
192       ResultSetMetaData rsmd ) throws SQLException
193    {
194       Vector currentRow = new Vector();
195
196       for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
197          switch( rsmd.getColumnType( i ) ) {
198             case Types.VARCHAR:
199             case Types.LONGVARCHAR:
200                   currentRow.addElement( rs.getString( i ) );
201                break;
202             case Types.INTEGER:
203                   currentRow.addElement( new Long( rs.getLong( i ) ) );
204                break;
205             case Types.REAL:
206                   currentRow.addElement( new Float( rs.getDouble( i ) ) );
207                break;
208             case Types.DATE:
209                   currentRow.addElement( rs.getDate( i ) );
210               break;
211             default:
212                System.out.println( "Type was: " +
213                   rsmd.getColumnTypeName( i ) );
214          }
215
216       return currentRow;
217
218    } // end method getNextRow
219
220    public void shutDown()
221    {
222       try {
```

```
223            connection.close();
224         }
225         catch ( SQLException sqlex ) {
226            System.err.println( "Unable to disconnect" );
227            sqlex.printStackTrace();
228         }
229      }
230
231      public static void main( String args[] )
232      {
233         final DisplayQueryResults app = new DisplayQueryResults();
234         app.addWindowListener(
235
236            new WindowAdapter() {
237
238               public void windowClosing( WindowEvent e )
239               {
240                  app.shutDown();
241                  System.exit( 0 );
242               }
243            }
244         );
245      }
246
247   } // end class DisplayQueryResults
```



23.6 Modify Exercise 23.5 to perform the following tasks:

a) Increase base salary by 10% for all base plus comission employees.

b) If the employee's birthday is in current month, add $100 bonus.

c) For all comission employee whose gross sales over 10000, add $100 bonus.

ANS:

```
1  // Exercise 23.6 Solution: DisplayQueryResults.java
2  import java.sql.*;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.util.*;
7
8  public class DisplayQueryResults extends JFrame {
9     private Connection connection;
10    private Statement statement;
11    private ResultSet resultSet;
12    private ResultSetMetaData rsMetaData;
13    private JTable table;
14    private JComboBox inputQuery;
15    private JButton submitQuery;
```

```
16    private JTextField input;
17
18    public DisplayQueryResults()
19    {
20       super( "Select Query. Click Submit to See Results." );
21
22       // The URL specifying the employees database to which this program
23       // connects to using JDBC
24       String url = "jdbc:db2j:employees";
25
26       // Load the driver to allow connection to the database
27       try {
28          Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
29
30          connection = DriverManager.getConnection( url );
31       }
32       catch ( ClassNotFoundException cnfex ) {
33          System.err.println( "Failed to load JDBC driver." );
34          cnfex.printStackTrace();
35          System.exit( 1 );  // terminate program
36       }
37       catch ( SQLException sqlex ) {
38          System.err.println( "Unable to connect" );
39          sqlex.printStackTrace();
40          System.exit( 1 );  // terminate program
41       }
42
43       String queries[] = { "Select all employees working in Department " +
44          "SALES.", "Select hourly employees working over 30 hours.",
45          "Select all comission employees in descending order of the " +
46          "comission rate.", "Increase base salary by 10% for all base " +
47          "plus comission employees.", "If the employee's birthday is " +
48          "in current month, add $100 bonus.", "For all comission " +
49          "employee whose gross sales over 10000, add $100 bonus.",
50          "Specify particular query" };
51
52       // If connected to database, set up GUI
53       inputQuery = new JComboBox( queries );
54
55       submitQuery = new JButton( "Submit query" );
56       submitQuery.addActionListener(
57
58          new ActionListener() {
59
60             public void actionPerformed( ActionEvent e )
61             {
62                getTable();
63             }
64          }
65       );
66
67       JPanel topPanel = new JPanel();
68       input = new JTextField( 50 );
69       input.addActionListener(
```

```
70
71          new ActionListener() {
72
73             public void actionPerformed( ActionEvent e )
74             {
75                try {
76                   String query = input.getText();
77                   statement = connection.createStatement();
78                   resultSet = statement.executeQuery( query );
79                   displayResultSet( resultSet );
80                }
81                catch ( SQLException sqlex ) {
82                   sqlex.printStackTrace();
83                }
84             }
85          }
86       );
87
88       JPanel centerPanel = new JPanel();
89       centerPanel.setLayout( new FlowLayout() );
90       centerPanel.add( new JLabel( "Enter query:" ) );
91       centerPanel.add( input );
92       topPanel.setLayout( new BorderLayout() );
93       topPanel.add( inputQuery, BorderLayout.NORTH );
94       topPanel.add( centerPanel, BorderLayout.CENTER );
95       topPanel.add( submitQuery, BorderLayout.SOUTH );
96
97       table = new JTable( 4, 4 );
98
99       Container c = getContentPane();
100      c.setLayout( new BorderLayout() );
101      c.add( topPanel, BorderLayout.NORTH );
102      c.add( table, BorderLayout.CENTER );
103
104      getTable();
105
106      setSize( 650, 200 );
107      setVisible( true );
108
109   } // end constructor DisplayQueryResult
110
111   private void getTable()
112   {
113      try {
114         int selection = inputQuery.getSelectedIndex();
115         String query = null;
116
117         switch ( selection ) {
118            case 0:
119               query = "SELECT * FROM employees WHERE " +
120                  "departmentName = 'SALES'";
121               break;
122            case 1:
123               query = "SELECT * FROM hourlyEmployees WHERE hours >= 30";
```

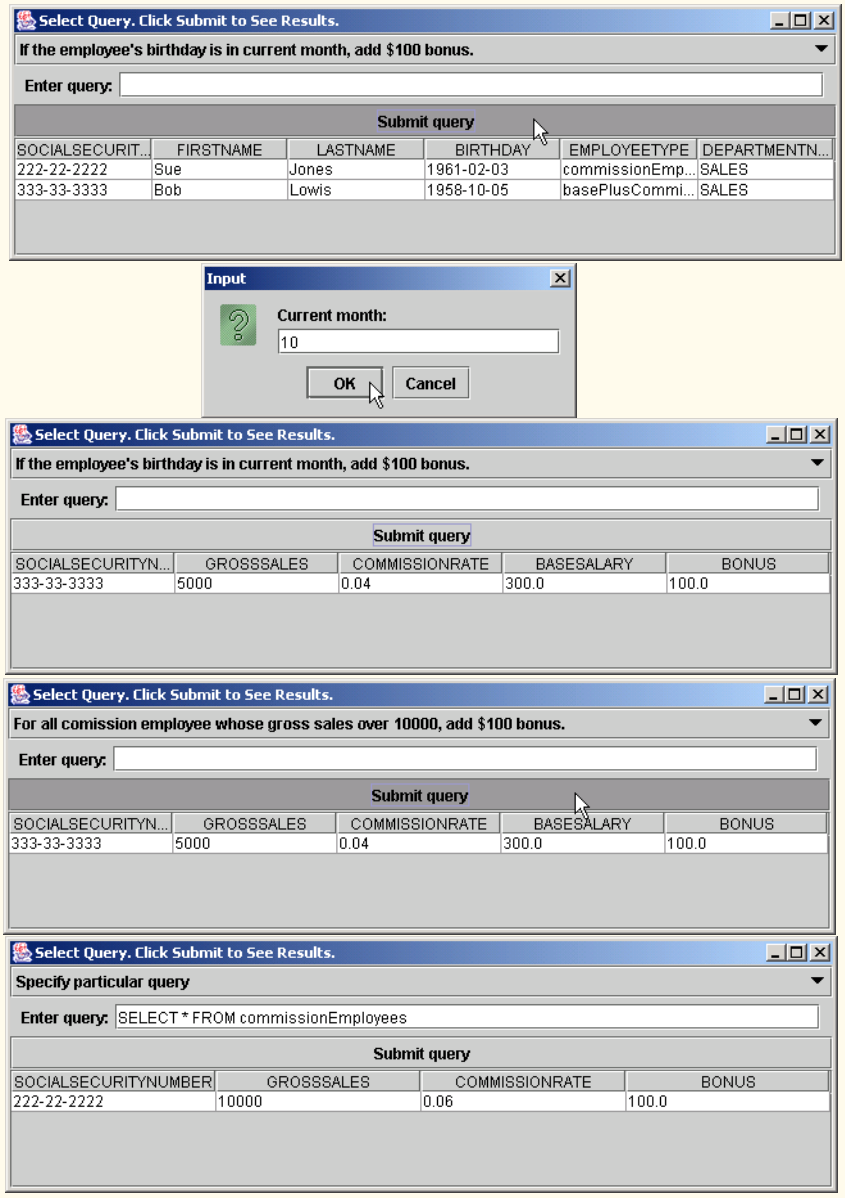
```
124                break;
125             case 2:
126                query = "SELECT * FROM commissionEmployees ORDER BY " +
127                   "commissionRate DESC";
128                break;
129             case 3:
130                query = "UPDATE basePlusCommissionEmployees SET " +
131                   "baseSalary = baseSalary * 1.1";
132                break;
133             case 4:
134                addBirthdayBonus();
135                break;
136             case 5:
137                query = "UPDATE commissionEmployees SET " +
138                   "bonus = bonus + 100.00 WHERE grossSales >= 10000";
139                break;
140             case 6:
141                query = input.getText();
142                break;
143
144          } // end switch
145
146          statement = connection.createStatement();
147
148          // execute query if not null
149          if ( query != null ) {
150
151             // select query
152             if ( query.substring( 0, 6 ).equals( "SELECT" ) ) {
153                resultSet = statement.executeQuery( query );
154                displayResultSet( resultSet );
155             }
156             else  // other queries
157                statement.executeUpdate( query );
158          }
159
160       } // end try
161
162       catch ( SQLException sqlex ) {
163          sqlex.printStackTrace();
164       }
165
166    } // end method getTable
167
168    private void addBirthdayBonus()
169    {
170       // get current month
171       int currentMonth = Integer.parseInt(
172          JOptionPane.showInputDialog( "Current month: " ) );
173
174       // validate current month
175       while ( !( currentMonth >= 1 && currentMonth <= 12 ) )
176          currentMonth = Integer.parseInt(
177             JOptionPane.showInputDialog( "Current month: " ) );
```

```
178
179       // add $100 bonus to employee whose birthday matches current month
180       try {
181          String getEmployees = "SELECT * FROM employees";
182          statement = connection.createStatement();
183          resultSet = statement.executeQuery( getEmployees );
184          String birthday;
185          Vector birthdayList = new Vector();
186
187          // find employee whose birthday match current month
188          while ( resultSet.next() ) {
189             birthday = resultSet.getDate( "birthday" ).toString();
190             int month = Integer.parseInt( birthday.substring( 5, 7 ) );
191
192             if ( month == currentMonth ) {
193                birthdayList.add(
194                   resultSet.getString( "socialSecurityNumber" ) );
195                birthdayList.add(
196                   resultSet.getString( "employeeType" ) + "s" );
197 }
198          }
199
200          addBonus( birthdayList );
201
202       } // end try
203
204       catch ( SQLException exception ) {
205          exception.printStackTrace();
206       }
207
208    }  // end addBirthdayBonus
209
210    private void addBonus( Vector vector )
211    {
212       String socialSecurityNumber, employeeType;
213
214       // add bonus to all employees in the vector
215       try {
216
217          // add $100 to each employee listed in the vector
218          for ( int i = 0; i < vector.size() / 2; i++ ) {
219             socialSecurityNumber = ( String ) vector.elementAt( i * 2 );
220             employeeType = ( String ) vector.elementAt( i * 2 + 1 );
221
222             // add $100 bonus
223             statement = connection.createStatement();
224             statement.executeUpdate( "UPDATE " + employeeType +
225                " SET bonus = bonus + 100.00 WHERE socialSecurityNumber " +
226                "= '" + socialSecurityNumber + "'" );
227
228             // display after update
229             statement = connection.createStatement();
230             ResultSet afterUpdate = statement.executeQuery(
231                "SELECT * FROM " + employeeType );
```

```
232             displayResultSet( afterUpdate );
233          }
234
235       } // end try
236
237       catch ( SQLException exception ) {
238          exception.printStackTrace();
239       }
240
241    } // end method addBonus
242
243    private void displayResultSet( ResultSet rs ) throws SQLException
244    {
245       // position to first record
246       boolean moreRecords = rs.next();
247
248       // If there are no records, display a message
249       if ( !moreRecords ) {
250          JOptionPane.showMessageDialog( this,
251             "ResultSet contained no records" );
252          setTitle( "No records to display" );
253          return;
254       }
255
256       Vector columnHeads = new Vector();
257       Vector rows = new Vector();
258
259       try {
260          // get column heads
261          ResultSetMetaData rsmd = rs.getMetaData();
262
263          for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
264             columnHeads.addElement( rsmd.getColumnName( i ) );
265
266          // get row data
267          do {
268             rows.addElement( getNextRow( rs, rsmd ) );
269          } while ( rs.next() );
270
271          // display table with ResultSet contents
272          table = new JTable( rows, columnHeads );
273          JScrollPane scroller = new JScrollPane( table );
274          Container c = getContentPane();
275          c.remove( 1 );
276          c.add( scroller, BorderLayout.CENTER );
277          c.validate();
278
279       } // end try
280
281       catch ( SQLException sqlex ) {
282          sqlex.printStackTrace();
283       }
284
285    } // end method displayResultSet
```

```
286
287    private Vector getNextRow( ResultSet rs,
288       ResultSetMetaData rsmd ) throws SQLException
289    {
290       Vector currentRow = new Vector();
291
292       for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
293          switch( rsmd.getColumnType( i ) ) {
294             case Types.VARCHAR:
295             case Types.LONGVARCHAR:
296                   currentRow.addElement( rs.getString( i ) );
297                break;
298             case Types.INTEGER:
299                   currentRow.addElement( new Long( rs.getLong( i ) ) );
300                break;
301             case Types.REAL:
302                   currentRow.addElement( new Float( rs.getDouble( i ) ) );
303                break;
304             case Types.DATE:
305                   currentRow.addElement( rs.getDate( i ) );
306               break;
307             default:
308                System.out.println( "Type was: " +
309                   rsmd.getColumnTypeName( i ) );
310          }
311
312       return currentRow;
313
314    } // end method getNextRow
315
316    public void shutDown()
317    {
318       try {
319          connection.close();
320       }
321       catch ( SQLException sqlex ) {
322          System.err.println( "Unable to disconnect" );
323          sqlex.printStackTrace();
324       }
325    }
326
327    public static void main( String args[] )
328    {
329       final DisplayQueryResults app = new DisplayQueryResults();
330       app.addWindowListener(
331
332          new WindowAdapter() {
333
334             public void windowClosing( WindowEvent e )
335             {
336                app.shutDown();
337                System.exit( 0 );
338             }
339          }
340       );
```

```
341      }
342
343   } // end class DisplayQueryResults
```



24

Servlets

Objectives

- To execute servlets with the Apache Tomcat server.
- To be able to respond to HTTP requests from an `HttpServlet`.
- To be able to redirect requests to static and dynamic Web resources.

A fair request should be followed by the deed in silence.
Dante Alighieri

The longest part of the journey is said to be the passing of the gate.
Marcus Terentius Varro

If nominated, I will not accept; if elected, I will not serve.
General William T. Sherman

Friends share all things.
Pythagoras

SELF-REVIEW EXERCISES

24.1 Fill in the blanks in each of the following statements:

a) Classes `HttpServlet` and `GenericServlet` implement the _____ interface.

ANS: `Servlet`

b) Class `HttpServlet` defines the methods _____ and _____ to respond to `get` and `post` requests from a client.

ANS: `doGet`, `doPost`

c) `HttpServletResponse` method _____ obtains a character-based output stream that enables text data to be sent to the client.

ANS: `getWriter`

d) The `form` attribute _____ specifies the server-side *form handler*, i.e., the program that handles the request.

ANS: `action`

e) _____ is the well-known host name that refers to your own computer.

ANS: `localhost`

24.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) Servlets usually are used on the client side of a networking application.

ANS: False. Servlets are usually used on the server side.

b) Servlet methods are executed by the servlet container.

ANS: True.

c) The two most common HTTP requests are `get` and `put`.

ANS: False. The two most common HTTP request types are `get` and `post`.

d) The well-known port number for Web requests is 55.

ANS: False. The well-known port number for Web requests is 80.

EXERCISES

24.3 Create a Web application for dynamic FAQs. The application should obtain the information to create the dynamic FAQ Web page from a database that consists of a `Topics` table and an `FAQ` table. The `Topics` table should have two fields—a unique integer ID for each topic (`topicID`) and a name for each topic (`topicName`). The `FAQ` table should have three fields—the `topicID` (a foreign key), a string representing the question (`question`) and the answer to the question (`answer`). When the servlet is invoked, it should read the data from the database and return a dynamically created Web page containing each question and answer, sorted by topic.

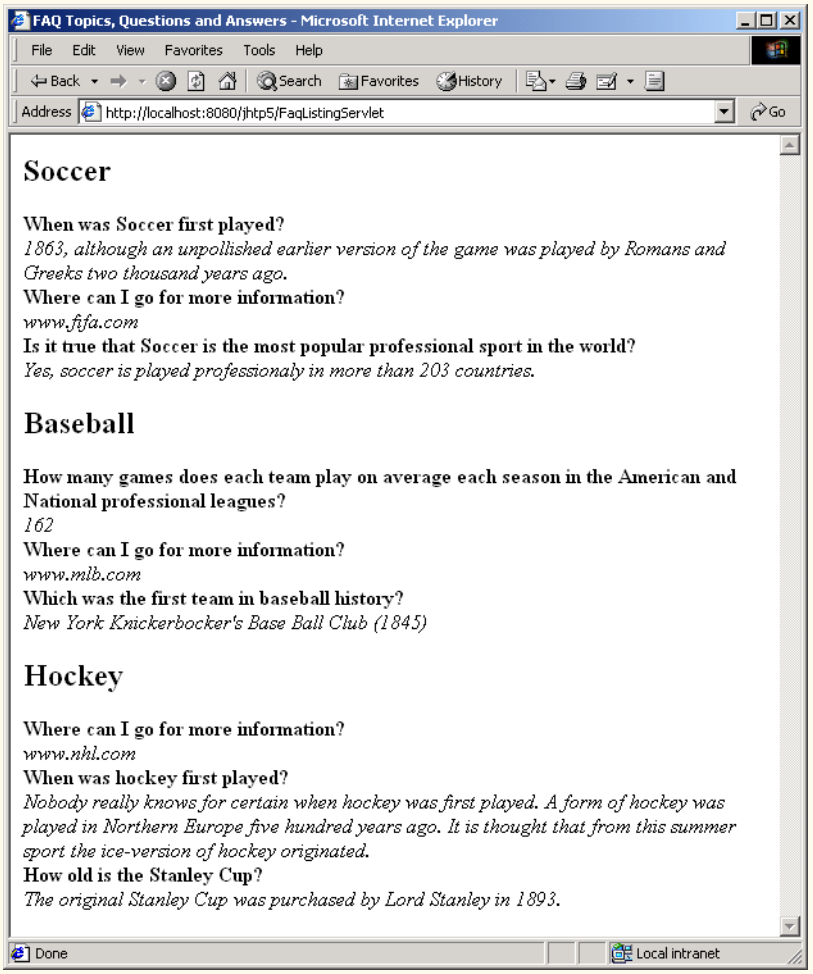
ANS:

```
1  // Exercise 24.3 solution: FaqListingServlet.java
2  // Displays all FAQs sorted by topic.
3  package com.deitel.jhtp5.servlets;
4
5  import java.sql.*;
6  import java.io.*;
7  import javax.servlet.http.*;
8  import javax.servlet.*;
9
10 public class FaqListingServlet extends HttpServlet {
11    private Connection connection;
12    private Statement statement;
13
```

```
14    // set up database connection and prepare SQL statements
15    public void init( ServletConfig config ) throws ServletException
16    {
17       // attempt database connection and create PreparedStatements
18       try {
19
20          System.setProperty( "db2j.system.home",
21             config.getInitParameter( "databaseLocation" ) );
22
23          Class.forName( config.getInitParameter( "databaseDriver" ) );
24          connection = DriverManager.getConnection(
25             config.getInitParameter( "databaseName" ) );
26
27          // create Statement to query database
28          statement = connection.createStatement();
29       }
30
31       // for any exception throw an UnavailableException to
32       // indicate that the servlet is not currently available
33       catch ( Exception exception ) {
34          exception.printStackTrace();
35          throw new UnavailableException(exception.getMessage());
36       }
37
38    }  // end of init method
39
40    // display all FAQs sorted by topic
41    public void doGet( HttpServletRequest request,
42       HttpServletResponse response )
43    {
44       String query;
45
46       // set response content type
47       response.setContentType( "text/html" );
48
49       // get output stream
50       PrintWriter out = null;
51       try {
52          out = response.getWriter();
53       } catch ( Exception exception ) {
54          exception.printStackTrace();
55          return;
56       }
57
58       // start XHTML document
59       out.println( "<?xml version = \"1.0\"?>" );
60
61       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
62          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
63          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
64
65       out.println(
66          "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
67
```

```
68          // head section of document
69          out.println( "<head>" );
70          out.println( "<title>FAQ Topics, Questions and Answers" +
71                       "</title>" );
72          out.println( "</head>" );
73
74          // body section of document
75          out.println( "<body>" );
76
77          // attempt to retrieve information from database
78          ResultSet baseballResults = null;
79          ResultSet soccerResults = null;
80          ResultSet hockeyResults = null;
81
82          // display information from database
83          try {
84
85             // display soccer information
86             out.println( "<h2> Soccer </h2> <p>" );
87
88             query = "SELECT TOPICNAME, QUESTION, ANSWER FROM FAQ, TOPICS " +
89                "WHERE FAQ.TOPICID = TOPICS.TOPICID AND FAQ.TOPICID = '1'";
90             soccerResults = statement.executeQuery( query );
91
92             while ( soccerResults.next() ) {
93
94                // display question
95                out.println( "<b>" + soccerResults.getString(
96                   "QUESTION" ) + "</b> <br>" );
97
98                // display answer
99                out.println( "<i>" + soccerResults.getString(
100                  "ANSWER" ) + "</i> <br>" );
101            }
102
103            // release ResultSet
104            soccerResults.close();
105
106            // display baseball information
107            out.println( "<h2> Baseball </h2> <p>" );
108            query = "SELECT TOPICNAME, QUESTION, ANSWER FROM FAQ, TOPICS " +
109              "WHERE FAQ.TOPICID = TOPICS.TOPICID AND FAQ.TOPICID = '2'";
110            baseballResults = statement.executeQuery( query );
111
112            while ( baseballResults.next() ) {
113
114               // display question
115               out.println( "<b>" + baseballResults.getString(
116                  "QUESTION" ) + "</b> <br>" );
117
118               // display answer
119               out.println( "<i>" + baseballResults.getString(
120                  "ANSWER" ) + "</i> <br>" );
121            }
```

```
122
123          // release ResultSet
124          baseballResults.close();
125
126          // display hockey information
127          out.println( "<h2> Hockey </h2> <p>" );
128          query = "SELECT TOPICNAME, QUESTION, ANSWER FROM FAQ, TOPICS " +
129            "WHERE FAQ.TOPICID = TOPICS.TOPICID AND FAQ.TOPICID = '3'";
130          hockeyResults = statement.executeQuery( query );
131
132          while ( hockeyResults.next() ) {
133
134             // display question
135             out.println( "<b>" + hockeyResults.getString(
136                "QUESTION" ) + "</b> <br>" );
137
138             // display answer
139             out.println( "<i>" + hockeyResults.getString(
140                "ANSWER" ) + "</i> <br>" );
141          }
142
143          // release ResultSet
144          hockeyResults.close();
145       }
146
147       // catch SQLExceptions
148       catch( SQLException exception ) {
149          exception.printStackTrace();
150          out.println( "<p><h1>... ERROR: could not retrieve" +
151                       " information from database</h1>" );
152          out.println( "</body></html>" );
153          return;
154       }
155
156       // finish html document structure
157       out.println( "</body>" );
158       out.println( "</html>" );
159
160    } // end method doGet
161
162 } // end class FaqListingServlet
```



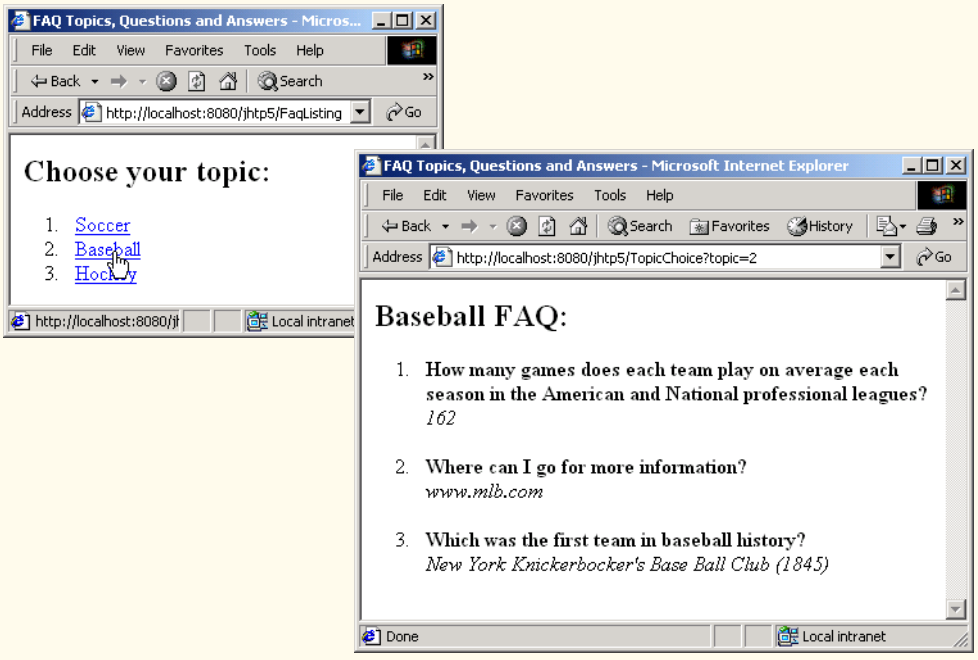
24.4 Modify the Web application of Exercise 24.3 so that the initial request to the servlet returns a Web page of topics in the FAQ database. Then, the user can hyperlink to another servlet that returns only the frequently asked questions for a particular topic.

ANS:

```
// Exercise 24.4 solution: FaqListingServlet.java
// Displays all FAQs sorted by topic.
package com.deitel.jhtp5.servlets;

import java.sql.*;
import java.io.*;
import javax.servlet.http.*;
import javax.servlet.*;

public class FaqListingServlet extends HttpServlet {
   private Connection connection;
   private Statement statement;

   // set up database connection and prepare SQL statements
   public void init( ServletConfig config ) throws ServletException
   {
      // attempt database connection and create PreparedStatements
      try {
         System.setProperty( "db2j.system.home",
            config.getInitParameter( "databaseLocation" ) );

         Class.forName( config.getInitParameter( "databaseDriver" ) );
         connection = DriverManager.getConnection(
            config.getInitParameter( "databaseName" ) );

         // create Statement to query database
         statement = connection.createStatement();
      }

      // for any exception throw an UnavailableException to
      // indicate that the servlet is not currently available
      catch ( Exception exception ) {
         exception.printStackTrace();
         throw new UnavailableException(exception.getMessage());
      }

   }  // end of init method

   // display all FAQs sorted by topic
   public void doGet( HttpServletRequest request,
                      HttpServletResponse response )
   {

      // set response content type
      response.setContentType( "text/html" );

      // get output stream
      PrintWriter out = null;

```

```
50          try {
51             out = response.getWriter();
52          } catch (Exception exception) {
53             exception.printStackTrace();
54             return;
55          }
56
57          // start XHTML document
58          out.println( "<?xml version = \"1.0\"?>" );
59
60          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
61             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
62             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
63
64          out.println(
65             "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
66
67          // head section of document
68          out.println( "<head>" );
69          out.println( "<title>FAQ Topics, Questions and Answers"
70                       + "</title>" );
71          out.println( "</head>" );
72
73          // body section of document
74          out.println( "<body>" );
75
76          ResultSet topicResults = null;
77
78          // attempt to retrieve information from database
79          try {
80
81             // display information from database
82             out.println( "<h2> Choose your topic: </h2> <p>" );
83
84             // obtain ResultSet
85             String query = "SELECT TOPICNAME, TOPICID FROM TOPICS";
86             topicResults = statement.executeQuery( query );
87
88             // display available topics
89             out.println( "<ol>" );
90             while ( topicResults.next() ) {
91
92                // create link information
93                String topicID = topicResults.getString( "TOPICID" );
94                String link = "<a href=\"/jhtp5/TopicChoice?topic=" +
95                   topicID + "\">" + topicResults.getString( "TOPICNAME" ) +
96                   "</a>";
97
98                // display link
99                out.println( "<li>" + link + "<br>" );
100            }
101            out.println( "</ol>" );
102
```

```
103          // release ResultSet
104          topicResults.close();
105       }
106
107       // catch SQLExceptions
108       catch( SQLException exception ) {
109          exception.printStackTrace();
110          out.println( "<p><h1>... ERROR: could not retrieve " +
111                       "information from database</h1>" );
112          out.println( "</body></html>" );
113          return;
114       }
115
116       // finish html document structure
117       out.println( "</body>" );
118       out.println( "</html>" );
119
120    } // end method doGet
121
122 } // end class FaqListingServlet
```




```
1  // Exercise 24.4 solution: TopicChoiceServlet.java
2  // Displays all FAQs sorted by topic.
3  package com.deitel.jhtp5.servlets;
4
5  import java.sql.*;
```

```
6  import java.io.*;
7  import javax.servlet.http.*;
8  import javax.servlet.*;
9
10 public class TopicChoiceServlet extends HttpServlet {
11    private Connection connection;
12
13    // set up database connection and prepare SQL statements
14    public void init( ServletConfig config ) throws ServletException
15    {
16
17       // attempt database connection and create PreparedStatements
18       try {
19          System.setProperty( "db2j.system.home",
20             config.getInitParameter( "databaseLocation" ) );
21
22          Class.forName( config.getInitParameter( "databaseDriver" ) );
23          connection = DriverManager.getConnection(
24             config.getInitParameter( "databaseName" ) );
25       }
26
27       // for any exception throw an UnavailableException to
28       // indicate that the servlet is not currently available
29       catch ( Exception exception ) {
30          exception.printStackTrace();
31          throw new UnavailableException(exception.getMessage() );
32       }
33
34    }  // end of init method
35
36    // display all FAQs sorted by topic
37    public void doGet( HttpServletRequest request,
38                       HttpServletResponse response )
39    {
40       // set response content type
41       response.setContentType( "text/html" );
42
43       // get output stream
44       PrintWriter out = null;
45       try {
46          out = response.getWriter();
47       } catch (Exception exception) {
48          exception.printStackTrace();
49          return;
50       }
51
52       // start XHTML document
53       out.println( "<?xml version = \"1.0\"?>" );
54
55       out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
56          "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
57          "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
58
59       out.println(
60          "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
```

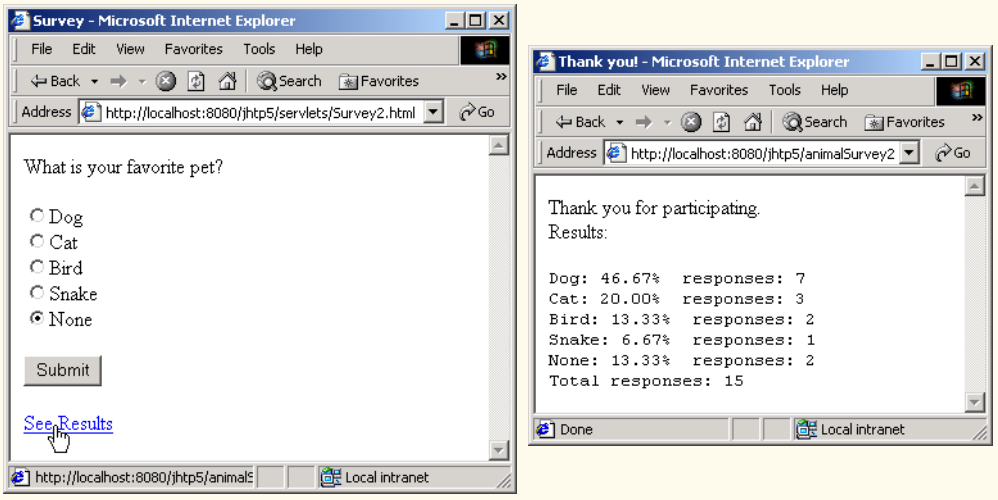
```
61
62          // head section of document
63          out.println( "<head>" );
64          out.println( "<title>FAQ Topics, Questions and Answers</title>" );
65          out.println( "</head>" );
66
67          // body section of document
68          out.println( "<body>" );
69
70          // attempt to retrieve information from database
71          ResultSet topicResults = null;
72          try {
73
74             // get selected topic
75             String topicID = request.getParameter( "topic" );
76             Statement statement = connection.createStatement();
77             String sqlQuery = "SELECT TOPICNAME, QUESTION, ANSWER FROM " +
78                "FAQ, TOPICS WHERE FAQ.TOPICID = TOPICS.TOPICID " +
79                "AND FAQ.TOPICID = '" + topicID + "'";
80
81             // get ResultSet
82             ResultSet topicResultSet = statement.executeQuery( sqlQuery );
83
84             // get first row
85             topicResultSet.next();
86
87             // display first row's information
88             String topicName = topicResultSet.getString( "TOPICNAME" );
89
90             // display information from database
91             out.println( "<h2>" + topicName + " FAQ: </h2> <p>" );
92
93             // display database information
94             out.println( "<ol>" );
95             do {
96
97                // obtain question and answer for current row
98                String question = topicResultSet.getString( "QUESTION" );
99                String answer = topicResultSet.getString( "ANSWER" );
100
101               // display link
102               out.println( "<li> <b>" + question + "</b> <br>" );
103               out.println( "<i>" + answer + "</i> <p>" );
104            } while ( topicResultSet.next() );
105            out.println( "</ol>" );
106
107            // release ResultSet
108            topicResultSet.close();
109         }
110
111         // catch SQLExceptions
112         catch( SQLException exception ) {
113            exception.printStackTrace();
114            out.println( "<p><h1>... ERROR: could not retrieve " +
115                         "information from database</h1>" );
```

```
116          out.println( "</body></html>" );
117          return;
118       }
119
120       // finish html document structure
121       out.println( "</body>" );
122       out.println( "</html>" );
123
124    } // end method doGet
125
126 } // end class TopicChoiceServlet
```

24.5 Modify the Web application of Fig. 24.20 to allow the user to see the survey results without responding to the survey.

ANS:

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Exercise 24.5: Survey2.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Survey</title>
10    </head>
11
12    <body>
13       <form method = "post" action = "/jhtp5/animalSurvey2">
14
15          <p>What is your favorite pet?</p>
16
17          <p>
18             <input type = "radio" name = "animal"
19                value = "1" />Dog<br />
20             <input type = "radio" name = "animal"
21                value = "2" />Cat<br />
22             <input type = "radio" name = "animal"
23                value = "3" />Bird<br />
24             <input type = "radio" name = "animal"
25                value = "4" />Snake<br />
26             <input type = "radio" name = "animal"
27                value = "5" checked = "checked" />None
28          </p>
29
30          <p><input type = "submit" value = "Submit" /></p>
31
32          <p><a href = "/jhtp5/animalSurvey2">See Results</a></p>
33
34       </form>
35    </body>
36 </html>
```




```
1  // Exercise 24.5 solution: SurveyServlet2.java
2  // A Web-based survey that uses JDBC from a servlet.
3  package com.deitel.jhtp5.servlets;
4
5  import java.io.*;
6  import java.text.*;
7  import java.sql.*;
8  import javax.servlet.*;
9  import javax.servlet.http.*;
10
11 public class SurveyServlet2 extends HttpServlet {
12    private Connection connection;
13    private Statement statement;
14
15    // set up database connection and create SQL statement
16    public void init( ServletConfig config ) throws ServletException
17    {
18       // attempt database connection and create Statement
19       try {
20          System.setProperty( "db2j.system.home",
21             config.getInitParameter( "databaseLocation" ) );
22
23          Class.forName( config.getInitParameter( "databaseDriver" ) );
24          connection = DriverManager.getConnection(
25             config.getInitParameter( "databaseName" ) );
26
27          // create Statement to query database
28          statement = connection.createStatement();
29       }
30
31       // for any exception throw an UnavailableException to
32       // indicate that the servlet is not currently available
33       catch ( Exception exception ) {
34          exception.printStackTrace();
```

```
35             throw new UnavailableException( exception.getMessage() );
36          }
37
38       }  // end of init method
39
40       // process survey response
41       protected void doPost( HttpServletRequest request,
42          HttpServletResponse response ) throws ServletException, IOException
43       {
44          PrintWriter out = response.getWriter();
45
46          // read current survey response
47          int value =
48             Integer.parseInt( request.getParameter( "animal" ) );
49          String query;
50
51          // attempt to process a vote and display current results
52          try {
53
54             // update total for current survey response
55             query = "UPDATE surveyresults SET votes = votes + 1 " +
56                   "WHERE id = " + value;
57             statement.executeUpdate( query );
58
59             displayResults( response );
60
61          } // end try
62
63          // if database exception occurs, return error page
64          catch ( SQLException sqlException ) {
65             sqlException.printStackTrace();
66             out.println( "<title>Error</title>" );
67             out.println( "</head>" );
68             out.println( "<body><p>Database error occurred. " );
69             out.println( "Try again later.</p></body></html>" );
70             out.close();
71          }
72
73       }  // end of doPost method
74
75       // process "get" request from client
76       protected void doGet( HttpServletRequest request,
77          HttpServletResponse response ) throws ServletException, IOException
78       {
79          displayResults( response );
80
81       }  // end method doGet
82
83       // display results
84       public void displayResults( HttpServletResponse response )
85             throws ServletException, IOException
86       {
87          String query;
88
```

```
89          // set up response to client
90          response.setContentType( "text/html" );
91          PrintWriter out = response.getWriter();
92          DecimalFormat twoDigits = new DecimalFormat( "0.00" );
93
94          // start XHTML document
95          out.println( "<?xml version = \"1.0\"?>" );
96
97          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
98             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
99             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
100
101         out.println(
102            "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
103
104         // head section of document
105         out.println( "<head>" );
106
107         // attempt to process a vote and display current results
108         try {
109
110            // get total of all survey responses
111            query = "SELECT sum( votes ) FROM surveyresults";
112            ResultSet totalRS = statement.executeQuery( query );
113            totalRS.next();
114            int total = totalRS.getInt( 1 );
115
116            // get results
117            query = "SELECT surveyoption, votes, id FROM surveyresults " +
118               "ORDER BY id";
119            ResultSet resultsRS = statement.executeQuery( query );
120            out.println( "<title>Thank you!</title>" );
121            out.println( "</head>" );
122
123            out.println( "<body>" );
124            out.println( "<p>Thank you for participating." );
125            out.println( "<br />Results:</p><pre>" );
126
127            // process results
128            int votes;
129
130            while ( resultsRS.next() ) {
131               out.print( resultsRS.getString( 1 ) );
132               out.print( ": " );
133               votes = resultsRS.getInt( 2 );
134               out.print( twoDigits.format(
135                  ( double ) votes / total * 100 ) );
136               out.print( "%  responses: " );
137               out.println( votes );
138            }
139
140            resultsRS.close();
141
142            out.print( "Total responses: " );
```

```
143          out.print( total );
144
145          // end XHTML document
146          out.println( "</pre></body></html>" );
147          out.close();
148
149       } // end try
150
151       // if database exception occurs, return error page
152       catch ( SQLException sqlException ) {
153          sqlException.printStackTrace();
154          out.println( "<title>Error</title>" );
155          out.println( "</head>" );
156          out.println( "<body><p>Database error occurred. " );
157          out.println( "Try again later.</p></body></html>" );
158          out.close();
159       }
160
161    } // end method displayResults
162
163    // close SQL statements and database when servlet terminates
164    public void destroy()
165    {
166       // attempt to close statements and database connection
167       try {
168          statement.close();
169          connection.close();
170       }
171
172       // handle database exceptions by returning error to client
173       catch( SQLException sqlException ) {
174          sqlException.printStackTrace();
175       }
176    }
177
178 } // end class SurveyServlet2
```

24.6 Modify the Web application of Fig. 24.20 to make it generic for use with any survey of the appropriate form. Use servlet parameters (as discussed in Section 24.7) to specify the survey options. When the user requests the survey, dynamically generate a `form` containing the survey options. Deploy this Web application twice using different context roots. *Note*: You may need to modify the database in this example so that it can store multiple surveys at once.

ANS:

```
1  // Exercise 24.6 solution: SurveyServlet.java
2  // A Web-based survey that uses JDBC from a servlet.
3  package com.deitel.jhtp5.servlets;
4
5  import java.io.*;
6  import java.text.*;
7  import java.sql.*;
8  import java.util.*;
```

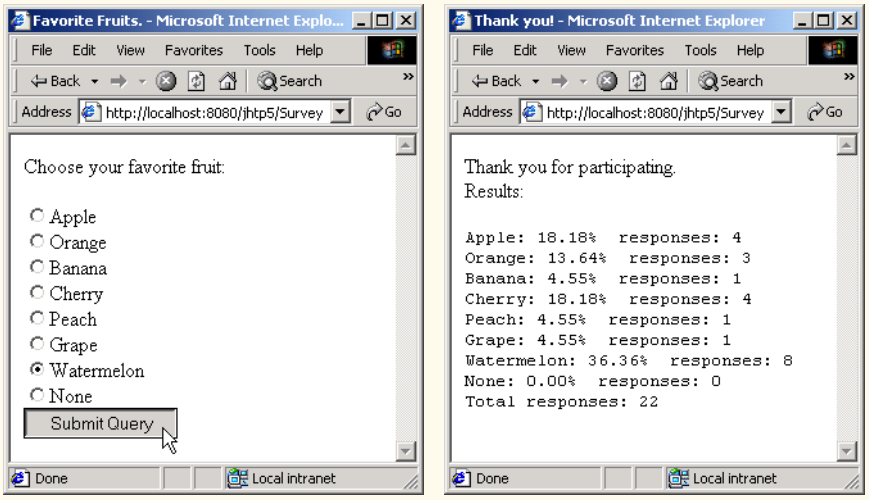
```
 9  import javax.servlet.*;
10  import javax.servlet.http.*;
11
12  public class SurveyServlet extends HttpServlet {
13     private Connection connection;
14
15     // servlet attributes
16     private String databaseName;
17     private String tableName;
18     private String surveyText;
19     private List surveyOptions = new ArrayList();
20     private String submissionIdentifier;
21     private String pageTitle;
22
23     // set up database connection and prepare SQL statements
24     public void init( ServletConfig config )
25        throws ServletException
26     {
27        // attempt database connection and create PreparedStatements
28        try {
29
30           // obtain initialization parameters
31           databaseName = config.getInitParameter( "databaseName" );
32           tableName = config.getInitParameter( "tableName" );
33           surveyText = config.getInitParameter( "surveyText" );
34           submissionIdentifier = config.getInitParameter(
35              "submissionIdentifier" );
36           pageTitle = config.getInitParameter( "pageTitle" );
37
38           // obtain column names
39           int index = 1;
40           String column = config.getInitParameter( "" + index++ );
41
42           while ( column != null ) {
43
44              // add columns to List
45              surveyOptions.add( column );
46              column = config.getInitParameter( "" + index++ );
47           }
48
49           // specify database location
50           System.setProperty( "db2j.system.home",
51              config.getInitParameter( "databaseLocation" ) );
52
53           // obtain driver reference
54           Class.forName( config.getInitParameter( "databaseDriver" ) );
55
56           // create connection to database
57           connection = DriverManager.getConnection(
58              config.getInitParameter( "databaseName" ) );
59        }
60
61        // for any exception throw an UnavailableException to
62        // indicate that the servlet is not currently available
63        catch ( Exception exception ) {
```

```
64             exception.printStackTrace();
65             throw new UnavailableException(exception.getMessage());
66          }
67
68       }  // end of init method
69
70       // display survey
71       protected void doGet( HttpServletRequest request,
72          HttpServletResponse response ) throws ServletException, IOException
73       {
74          // set up response to client
75          response.setContentType( "text/html" );
76          PrintWriter out = response.getWriter();
77
78          // start XHTML document
79          out.println( "<?xml version = \"1.0\"?>" );
80
81          out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
82             "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
83             "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
84
85          out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
86
87          // head section of document
88          out.println( "<head>" );
89
90          // set document title
91          out.println( "<title>" + pageTitle + "</title>" );
92
93          out.println( "</head>" );
94
95          // display survey
96          out.println( "<body>" );
97
98          // display survey information
99          out.println( surveyText );
100
101         // display form
102         out.println( "<form method=POST action=\"/jhtp5/Survey\">" );
103
104         // display survey elements
105         for (int i = 0; i < surveyOptions.size(); i++ ) {
106            String element = (String) surveyOptions.get( i );
107            out.println( "<input type=\"radio\" name=\"" +
108               submissionIdentifier + "\"value=\"" + ( i + 1 ) +
109               "\">" + element + "<br>" );
110         }
111
112         // display button
113         out.println( "<input type=\"Submit\">" );
114
115         // close form
116         out.println( "</form>" );
117
```

```
118         // finalize page html
119         out.println( "</body> </html>" );
120
121      } // end method doGet
122
123      // process survey response
124      protected void doPost( HttpServletRequest request,
125         HttpServletResponse response ) throws ServletException, IOException
126      {
127         // set up response to client
128         response.setContentType( "text/html" );
129         PrintWriter out = response.getWriter();
130         DecimalFormat twoDigits = new DecimalFormat( "0.00" );
131
132         // start XHTML document
133         out.println( "<?xml version = \"1.0\"?>" );
134
135         out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
136            "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
137            "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
138
139         out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
140
141         // head section of document
142         out.println( "<head>" );
143
144         // read current survey response
145         int value =
146            Integer.parseInt( request.getParameter( submissionIdentifier ) );
147
148         // attempt to process a vote and display current results
149         try {
150
151            // update total for current survey response
152            String updateVotesSQL = "UPDATE " + tableName +
153               " SET votes = votes + 1 " + "WHERE id =" + value;
154            Statement updateVotes = connection.createStatement();
155            updateVotes.executeUpdate( updateVotesSQL );
156
157            // get total of all survey responses
158            Statement totalVotes = connection.createStatement();
159            String totalVotesSQL = "SELECT sum( votes ) FROM " + tableName;
160            ResultSet totalRS = totalVotes.executeQuery( totalVotesSQL );
161
162            totalRS.next();
163            int total = totalRS.getInt( 1 );
164
165            // get results
166            // PreparedStatement to obtain surveyoption table's data
167            Statement results = connection.createStatement();
168            String resultsSQL = "SELECT surveyoption, votes, id " +
169               "FROM " + tableName + " ORDER BY id";
170            ResultSet resultsRS = results.executeQuery( resultsSQL );
171
```

```
172          // display contents XHTML
173          out.println( "<title>Thank you!</title>" );
174          out.println( "</head>" );
175
176          out.println( "<body>" );
177          out.println( "<p>Thank you for participating." );
178          out.println( "<br />Results:</p><pre>" );
179
180          // process results
181          int votes;
182
183          while ( resultsRS.next() ) {
184             out.print( resultsRS.getString( 1 ) );
185             out.print( ": " );
186             votes = resultsRS.getInt( 2 );
187             out.print( twoDigits.format(
188                ( double ) votes / total * 100 ) );
189             out.print( "%  responses: " );
190             out.println( votes );
191          }
192
193          resultsRS.close();
194
195          out.print( "Total responses: " );
196          out.print( total );
197
198          // end XHTML document
199          out.println( "</pre></body></html>" );
200          out.close();
201
202       } // end try
203
204       // if database exception occurs, return error page
205       catch ( SQLException sqlException ) {
206          sqlException.printStackTrace();
207          out.println( "<title>Error</title>" );
208          out.println( "</head>" );
209          out.println( "<body><p>Database error occurred. " );
210          out.println( "Try again later.</p></body></html>" );
211          out.close();
212       }
213
214    }  // end of doPost method
215
216    // close SQL statements and database when servlet terminates
217    public void destroy()
218    {
219       // attempt to close statements and database connection
220       try {
221          connection.close();
222       }
223
224       // handle database exceptions by returning error to client
225       catch( SQLException sqlException ) {
```

```
226             sqlException.printStackTrace();
227          }
228       }  // end of destroy method
229
230 } // end class SurveyServlet
```

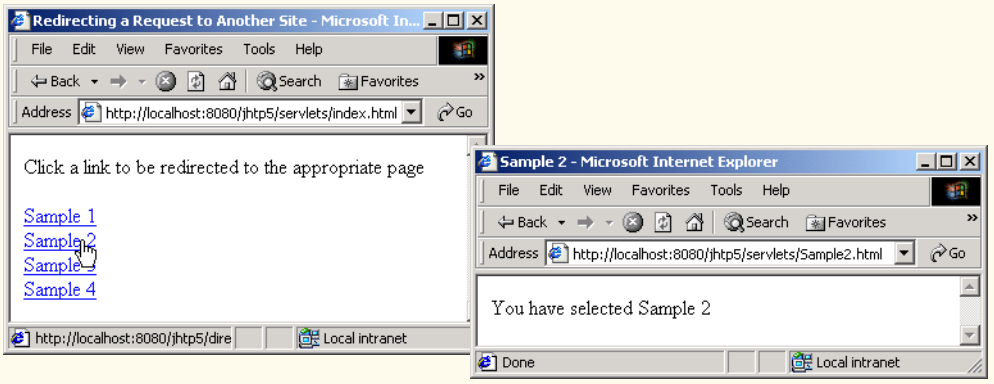


24.7 Write a Web application that consists of a servlet (`DirectoryServlet`) and several Web documents. Document `index.html` should be the first document the user sees. In that document, you should have a series of hyperlinks for other Web pages in your site. When clicked, each hyperlink should invoke the servlet with a `get` request that contains a `page` parameter. The servlet should obtain parameter `page` and redirect the request to the appropriate document.

ANS:

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Exercise 24.7: index.html -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Redirecting a Request to Another Site</title>
10    </head>
11
12    <body>
13       <p>
14          Click a link to be redirected to the appropriate page
15       </p>
16       <p>
17          <a href = "/jhtp5/directory?page=sample1">
18             Sample 1</a><br />
19          <a href = "/jhtp5/directory?page=sample2">
20             Sample 2</a><br />
```

```
21            <a href = "/jhtp5/directory?page=sample3">
22               Sample 3</a><br />
23            <a href = "/jhtp5/directory?page=sample4">
24               Sample 4</a>
25         </p>
26      </body>
27   </html>
```




```
1  // Exercise 24.7: DirectoryServlet.java
2  // Program demonstrates redirection.
3  package com.deitel.jhtp5.servlets;
4
5  import javax.servlet.*;
6  import javax.servlet.http.*;
7  import java.io.*;
8
9  public class DirectoryServlet extends HttpServlet {
10
11    // process "get" request from client
12    protected void doGet( HttpServletRequest request,
13       HttpServletResponse response ) throws ServletException, IOException
14    {
15       String location = request.getParameter( "page" );
16
17       if ( location != null )
18
19          if ( location.equals( "sample1" ) )
20             response.sendRedirect( "servlets/Sample1.html" );
21
22          else if ( location.equals( "sample2" ) )
23             response.sendRedirect( "servlets/Sample2.html" );
24
25          else if ( location.equals( "sample3" ) )
26             response.sendRedirect( "servlets/Sample3.html" );
27
28          else if ( location.equals( "sample4" ) )
29             response.sendRedirect( "servlets/Sample4.html" );
30
```

```
31         // code that executes only if this servlet
32         // does not redirect the user to another page
33         response.setContentType( "text/html" );
34         PrintWriter out = response.getWriter();
35         out.println( "<?xml version = \"1.0\"?>" );
36         out.println( "<!DOCTYPE html PUBLIC \"-//W3C//DTD " +
37            "XHTML 1.0 Strict//EN\" \"http://www.w3.org" +
38            "/TR/xhtml1/DTD/xhtml1-strict.dtd\">" );
39         out.println( "<html xmlns = \"http://www.w3.org/1999/xhtml\">" );
40         out.println( "<head>" );
41         out.println( "<title>Invalid Page</title>" );
42         out.println( "</head>" );
43         out.println( "<p><a href = \"servlets/DirectoryServlet.html\">" );
44         out.println( "Click here to choose again</a></p>" );
45         out.println( "</body>" );
46         out.println( "</html>" );
47         out.close();
48
49      }  // end method doGet
50
51   }  // end class DirectoryServlet
```

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 24.7: Sample1.html -->
6
7   <html xmlns = "http://www.w3.org/1999/xhtml">
8   <head>
9      <title>Sample 1</title>
10  </head>
11
12  <body>
13  <form method = "post" action = "/jhtp5/sample1">
14
15     <p>You have selected Sample 1</p>
16
17  </form>
18
19  </body>
20  </html>
```

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 24.7: Sample2.html -->
6
7   <html xmlns = "http://www.w3.org/1999/xhtml">
8   <head>
9      <title>Sample 2</title>
```

```
10   </head>
11
12   <body>
13   <form method = "post" action = "/jhtp5/sample2">
14
15      <p>You have selected Sample 2</p>
16
17   </form>
18
19   </body>
20   </html>
```

```
 1   <?xml version = "1.0"?>
 2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4
 5   <!-- Exercise 24.7: Sample3.html -->
 6
 7   <html xmlns = "http://www.w3.org/1999/xhtml">
 8   <head>
 9      <title>Sample 3</title>
10   </head>
11
12   <body>
13   <form method = "post" action = "/jhtp5/sample3">
14
15      <p>You have selected Sample 3</p>
16
17   </form>
18
19   </body>
20   </html>
```

```
 1   <?xml version = "1.0"?>
 2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
 4
 5   <!-- Exercise 24.7: Sample4.html -->
 6
 7   <html xmlns = "http://www.w3.org/1999/xhtml">
 8   <head>
 9      <title>Sample 4</title>
10   </head>
11
12   <body>
13   <form method = "post" action = "/jhtp5/sample4">
14
15      <p>You have selected Sample 4</p>
16
17   </form>
18
19   </body>
20   </html>
```

25

JavaServer Pages (JSP)

Objectives

- To be able to create and deploy JavaServer Pages.
- To use JSP's implicit objects and scriptlets to create dynamic Web pages.
- To specify global JSP information with directives.
- To use actions to manipulate JavaBeans in a JSP, to include resources dynamically and to forward requests to other JSPs.

A tomato does not communicate with a tomato, we believe. We could be wrong.
Gustav Eckstein

A donkey appears to me like a horse translated into Dutch.
Georg Christoph Licthtenberg

Talent is a question of quantity. Talent does not write one page: it writes three hundred.
Jules Renard

Every action must be due to one or other of seven causes: chance, nature, compulsion, habit, reasoning, anger, or appetite.
Aristotle

SELF-REVIEW EXERCISES

25.1 Fill in the blanks in each of the following statements:

a) Action _____ has the ability to match request parameters to properties of the same name in a bean by specifying `"*"` for attribute `property`.

ANS: `<jsp:setProperty>`

b) There are four key components to JSPs: _____, _____, _____ and _____.

ANS: directives, actions, scriptlets, tag libraries

c) The implicit objects have four scopes: _____, _____, _____ and _____.

ANS: application, page, request and session

d) The _____ directive is processed once, at JSP translation time and causes content to be copied into the JSP.

ANS: `include`

e) Classes and interfaces specific to JavaServer Pages programming are located in packages _____ and _____.

ANS: `avax.servlet.jsp`, `javax.servlet.jsp.tagext`

f) JSPs normally execute as part of a Web server that is referred to as the _____.

ANS: JSP container

g) JSP scripting components include _____, _____, _____, _____ and _____.

ANS: scriptlets, comments, expressions, declarations, escape sequences.

25.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) An object in page scope exists in every JSP of a particular Web application.

ANS: False. Objects in page scope exist only as part of the page in which they are used.

b) Directives specify global information that is not associated with a particular JSP request.

ANS: True.

c) Action `<jsp:include>` is evaluated once at page translation time.

ANS: False. Action `<jsp:include>` enables dynamic content to be included in a JavaServer Page.

d) Like XHTML comments, JSP comments and script language comments appear in the response to the client.

ANS: False. JSP comments and script language comments are ignored and do not appear in the response.

e) Objects in application scope are part of a particular Web application.

ANS: False. Objects in application scope are part of the JSP container application.

f) Each page has its own instances of the page-scope implicit objects.

ANS: True.

g) Action `<jsp:setProperty>` has the ability to match request parameters to properties of the same name in a bean by specifying `"*"` for attribute `property`.

ANS: True.

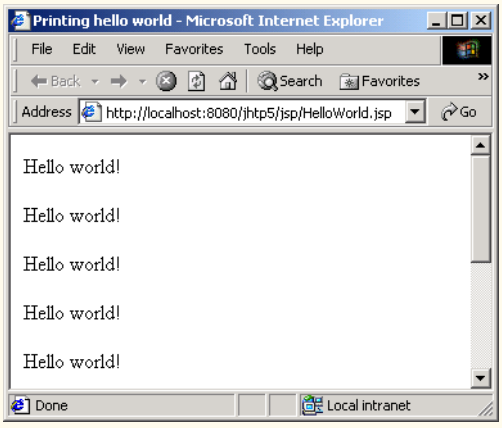
h) Objects in session scope exist for the client's entire browsing session.

ANS: True.

EXERCISES

25.3 Write a JSP page to output the string "Hello world!" ten times.
ANS:

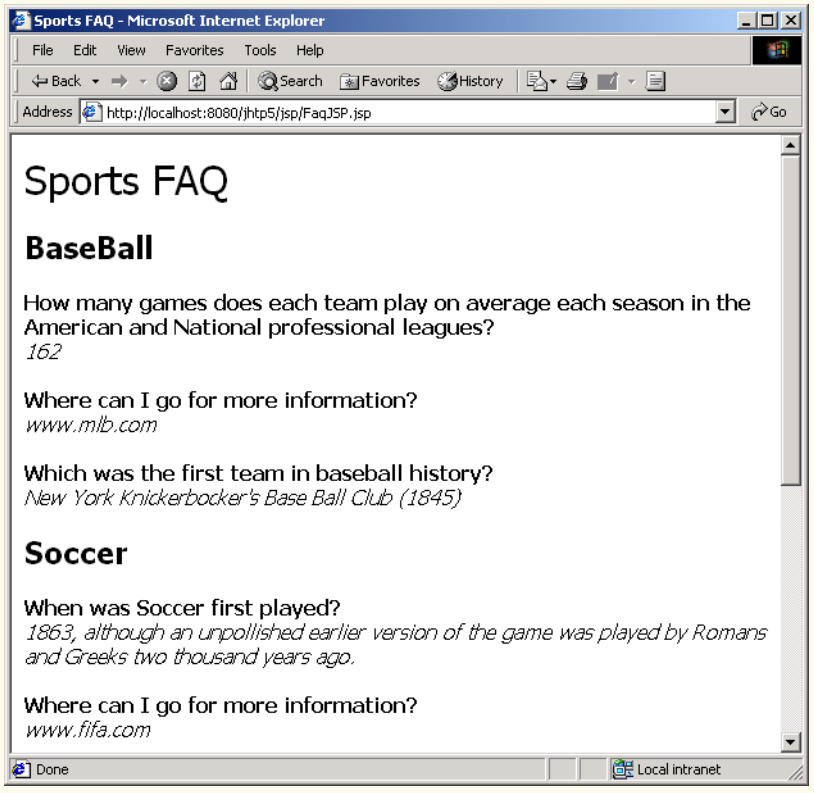
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Exercise 25.3 solution: HelloWorld.jsp -->
6
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8
9  <head>
10    <title>Printing hello world</title>
11 </head>
12
13 <body>
14 <% for( int i = 0; i < 10; i++ ) { %>
15
16    <p>Hello world!</p>
17
18 <% } %>
19 </body>
20 </html>
```



25.4 Modify Exercise 24.4 to run as a JSP page.

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Excercise 25.4: FaqJSP.jsp -->
<%-- page settings --%>
<%@ page import = "java.util.*" %>
<%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>

<%-- beans used in this JSP --%>
<jsp:useBean id = "faqData" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.FaqDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <title>Sports FAQ</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td, th {
            text-align: center;
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <p style = "font-size: 2em;">Sports FAQ</p>

      <!-- get Baseball FAQs -->
      <h2>BaseBall</h2><p>

<% // start scriptlet

   List faqList = faqData.getQuestions( "baseball" );
   Iterator faqListIterator = faqList.iterator();
   FaqBean faq;

   while ( faqListIterator.hasNext() ) {
      faq = ( FaqBean ) faqListIterator.next();

%> <%-- end scriptlet; insert fixed template data --%>

      <b><%= faq.getQuestion() %></b><br />
```

```
52          <i><%= faq.getAnswer() %></i><br /><br />
53
54    <% // continue scriptlet
55
56       } // end while
57
58    %> <%-- end scriptlet --%>
59
60          <!-- get Soccer FAQs -->
61          <h2>Soccer</h2><p>
62
63    <% // start scriptlet
64
65       faqList = faqData.getQuestions( "soccer" );
66       faqListIterator = faqList.iterator();
67
68       while ( faqListIterator.hasNext() ) {
69          faq = ( FaqBean ) faqListIterator.next();
70
71    %> <%-- end scriptlet; insert fixed template data --%>
72
73          <b><%= faq.getQuestion() %></b><br />
74          <i><%= faq.getAnswer() %></i><br /><br />
75
76    <% // continue scriptlet
77
78       } // end while
79
80    %> <%-- end scriptlet --%>
81
82          <!-- get Hockey FAQs -->
83          <h2>Hockey</h2><p>
84
85    <% // start scriptlet
86
87       faqList = faqData.getQuestions( "hockey" );
88       faqListIterator = faqList.iterator();
89
90       while ( faqListIterator.hasNext() ) {
91          faq = ( FaqBean ) faqListIterator.next();
92
93    %> <%-- end scriptlet; insert fixed template data --%>
94
95          <b><%= faq.getQuestion() %></b><br />
96          <i><%= faq.getAnswer() %></i><br /><br />
97
98    <% // continue scriptlet
99
100      } // end while
101
102   %> <%-- end scriptlet --%>
103
104      </body>
105   </html>
```




```
1  // Excercise 25.4 solution: FaqBean.java
2  // Class FaqBean contains the nececary information
3  // for a FAQ, and supports get and set methods.
4  package com.deitel.jhtp5.jsp.beans;
5
6  public class FaqBean {
7     String topicName;
8     String question;
9     String answer;
10
11    // sets topic name
12    public void setTopicName( String topicName )
13    {
14       this.topicName = topicName;
15    }
16
17    // gets topic name
18    public String getTopicName()
19    {
20       return topicName;
21    }
22
```

```
23     // sets question
24     public void setQuestion( String question )
25     {
26        this.question = question;
27     }
28
29     // gets question
30     public String getQuestion()
31     {
32        return question;
33     }
34
35     // sets answer
36     public void setAnswer( String answer )
37     {
38        this.answer = answer;
39     }
40
41     // gets answer
42     public String getAnswer()
43     {
44        return answer;
45     }
46
47  } // end class FaqBean
```

```
 1  // Excercise 25.4 solution: FaqDataBean.java
 2  // Class FaqDataBean makes a database connection and supports
 3  // retrieving FAQ data from the database.
 4  package com.deitel.jhtp5.jsp.beans;
 5
 6  import java.io.*;
 7  import java.sql.*;
 8  import java.util.*;
 9
10  public class FaqDataBean {
11     private Connection connection;
12     private Statement statement;
13     private String baseballQuery, soccerQuery, hockeyQuery;
14
15     // construct FaqDataBean object
16     public FaqDataBean() throws Exception
17     {
18         // specify database location
19        System.setProperty( "db2j.system.home", "C:/CloudScape_5.0" );
20
21        // load the Cloudscape driver
22        Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
23
24        // connect to the database
25        connection = DriverManager.getConnection( "jdbc:db2j:faq" );
26
```

```
27          // create Statement to query database
28          statement = connection.createStatement();
29
30          // define soccer query
31          soccerQuery = "SELECT TOPICNAME, QUESTION, ANSWER FROM " +
32             "FAQ, TOPICS WHERE FAQ.TOPICID = TOPICS.TOPICID " +
33             "AND FAQ.TOPICID = '1'";
34
35          // define baseball query
36          baseballQuery = "SELECT TOPICNAME, QUESTION, ANSWER FROM " +
37             "FAQ, TOPICS WHERE FAQ.TOPICID = TOPICS.TOPICID " +
38             "AND FAQ.TOPICID = '2'";
39
40          // define hockey query
41          hockeyQuery = "SELECT TOPICNAME, QUESTION, ANSWER FROM " +
42             "FAQ, TOPICS WHERE FAQ.TOPICID = TOPICS.TOPICID " +
43             "AND FAQ.TOPICID = '3'";
44       }
45
46       // return an ArrayList of Questions
47       public List getQuestions( String type ) throws SQLException
48       {
49          List faqList = new ArrayList();
50          ResultSet results = null;
51
52          // determine sport and excecute query
53          if ( type.equals( "baseball" ) )
54              results = statement.executeQuery( baseballQuery );
55          else if ( type.equals( "soccer" ) )
56             results = statement.executeQuery( soccerQuery );
57          else if ( type.equals( "hockey" ) )
58              results = statement.executeQuery( hockeyQuery );
59
60          // get row data
61          while ( results.next() ) {
62             FaqBean faq = new FaqBean();
63
64             faq.setTopicName( results.getString( 1 ) );
65             faq.setQuestion( results.getString( 2 ) );
66             faq.setAnswer( results.getString( 3 ) );
67
68             faqList.add( faq );
69          }
70
71          return faqList;
72
73       } // end method getQuestions
74
75
76       // close statements and terminate database connection
77       protected void finalize()
78       {
79          // attempt to close database connection
80          try {
```

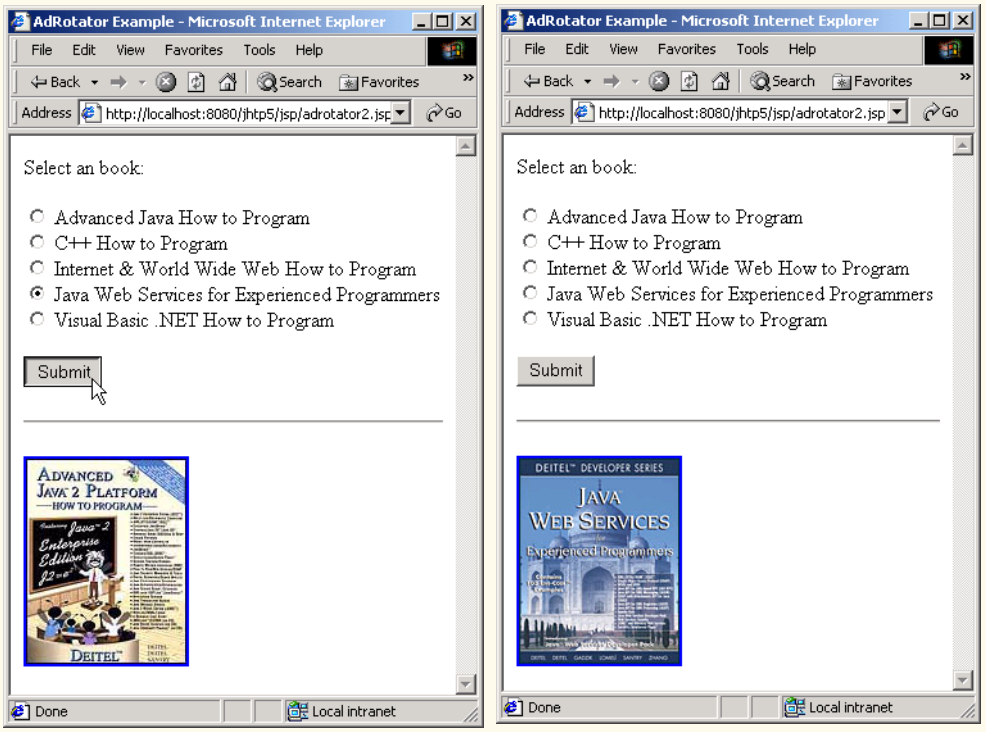
```
81             statement.close();
82             connection.close();
83          }
84
85          // process SQLException on close operation
86          catch ( SQLException sqlException ) {
87             sqlException.printStackTrace();
88          }
89       }
90
91    } // end class FaqDataBean
```

25.5 Rewrite Figure 25.15 to allow users to select the image. Use a JSP expression instead of the `getProperty` JSP tag.

ANS:

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 25.5 solution: adrotator2.jsp -->
6
7   <jsp:useBean id = "rotator" scope = "application"
8      class = "com.deitel.jhtp5.jsp.Rotator" />
9
10  <html xmlns = "http://www.w3.org/1999/xhtml">
11
12     <head>
13        <title>AdRotator Example</title>
14
15        <style type = "text/css">
16           .big { font-family: helvetica, arial, sans-serif;
17                  font-weight: bold;
18                  font-size: 2em }
19        </style>
20     </head>
21
22     <body>
23
24        <form method = "post" action = "/jhtp5/jsp/adrotator2.jsp">
25
26           <p>Select an book:</p>
27
28           <p>
29              <input type = "radio" name = "book" value = "0" />
30                 Advanced Java How to Program<br />
31              <input type = "radio" name = "book" value = "1" />
32                 C++ How to Program<br />
33              <input type = "radio" name = "book" value = "2" />
34                 Internet &amp; World Wide Web How to Program<br />
35              <input type = "radio" name = "book" value = "3" />
36                 Java Web Services for Experienced Programmers<br />
```

```
37                <input type = "radio" name = "book" value = "4" />
38                   Visual Basic .NET How to Program<br />
39             </p>
40
41             <p><input type = "submit" value = "Submit" /></p>
42
43          </form>
44
45          <hr />
46
47          <p>
48             <% if ( request.getParameter( "book" ) != null )
49                   rotator.setLink( Integer.parseInt(
50                      request.getParameter( "book" ) ) );
51             %>
52
53             <a href = <%= rotator.getLink() %> >
54
55                <img src = <%= rotator.getImage() %> alt = "advertisement" />
56             </a>
57          </p>
58       </body>
59    </html>
```



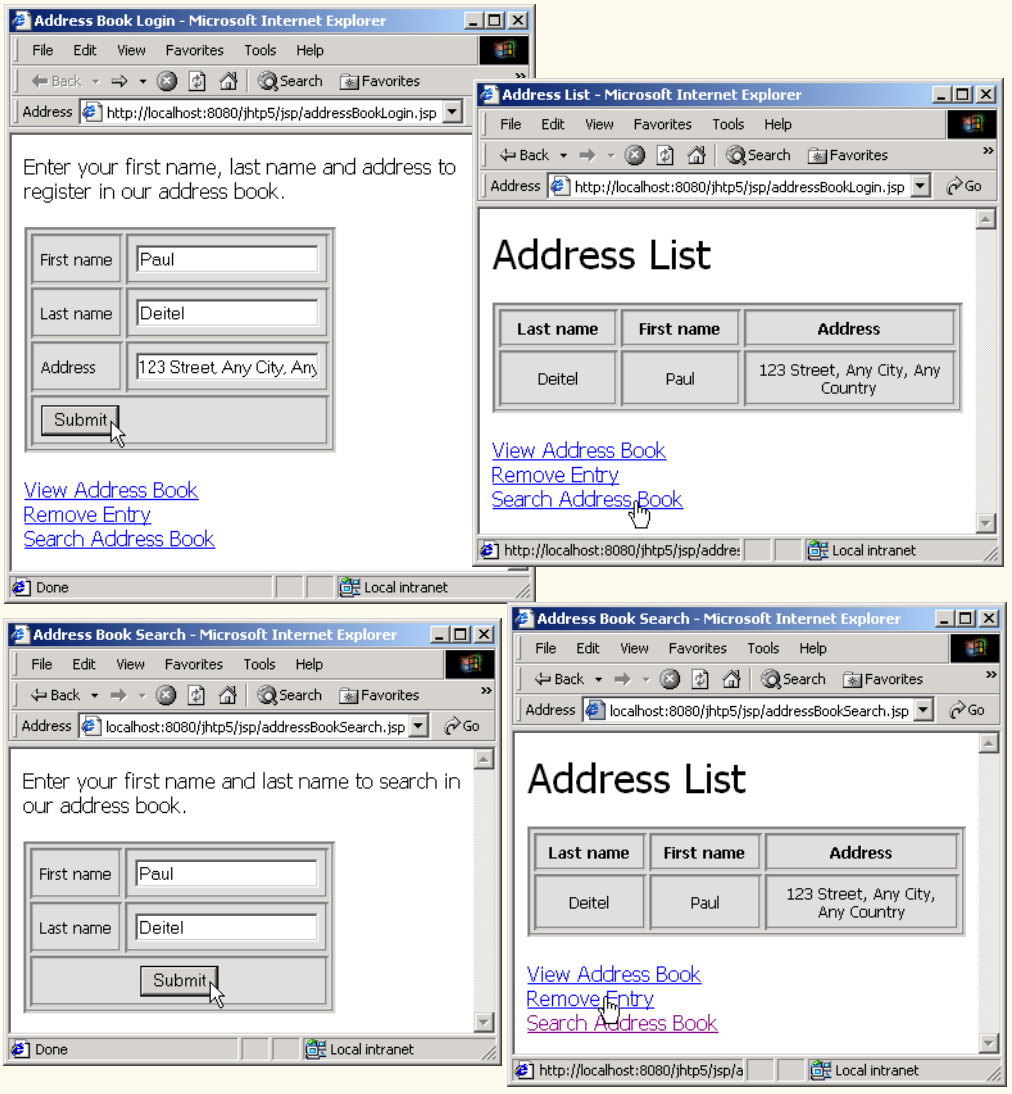
25.6 Create a JSP and JDBC-based address book. Use the guest book example of Fig. 25.20 through Fig. 25.24 as a guide. Your address book should allow one to insert entries, delete entries and search for entries.

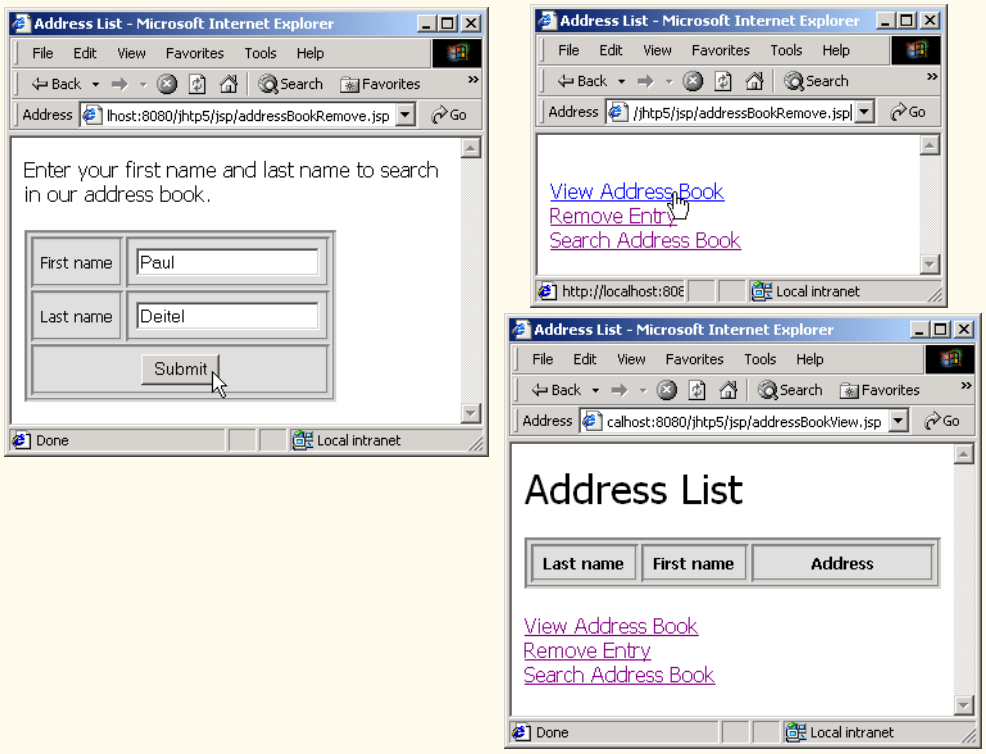
ANS:

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Exercise 25.6 solution: addressBookLogin.jsp -->

<%-- page settings --%>
<%@ page errorPage = "addressBookErrorPage.jsp" %>

<%-- beans used in this JSP --%>
<jsp:useBean id = "address" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.AddressBean" />
<jsp:useBean id = "addressData" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.AddressDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <title>Address Book Login</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td {
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <jsp:setProperty name = "address" property = "*" />

<% // start scriptlet

   if ( address.getFirstName() == null ||
      address.getLastName() == null ||
      address.getAddress() == null ) {

%> <%-- end scriptlet to insert fixed template data --%>

         <form method = "post" action = "addressBookLogin.jsp">

            <p>Enter your first name, last name and
               address to register in our address book.</p>
```

```
50
51             <table>
52                <tr>
53                   <td>First name</td>
54
55                   <td>
56                      <input type = "text" name = "firstName" />
57                   </td>
58                </tr>
59
60                <tr>
61                   <td>Last name</td>
62
63                   <td>
64                      <input type = "text" name = "lastName" />
65                   </td>
66                </tr>
67
68                <tr>
69                   <td>Address</td>
70
71                   <td>
72                      <input type = "text" name = "address" />
73                   </td>
74                </tr>
75
76                <tr>
77                   <td colspan = "2">
78                      <input type = "submit" value = "Submit" />
79                   </td>
80                </tr>
81             </table>
82          </form>
83
84          <!-- links to view, delete and search address book -->
85          <a href = "addressBookView.jsp">View Address Book</a><br />
86          <a href = "addressBookRemove.jsp">Remove Entry</a><br />
87          <a href = "addressBookSearch.jsp">Search Address Book</a>
88
89    <% // continue scriptlet
90
91       }  // end if
92       else {
93          addressData.addEntry( address );
94
95    %>
96          <jsp:forward page = "addressBookView.jsp" />
97
98    <%
99       }
100
101   %> <%-- end scriptlet to add links --%>
102
103      </body>
```

```
104 </html>
```






```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Exercise 25.6 solution: addressBookView.jsp -->
6
7  <%-- page settings --%>
8  <%@ page errorPage = "addressBookErrorPage.jsp" %>
9  <%@ page import = "java.util.*" %>
10 <%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>
11
12 <%-- AddressDataBean to obtain address list --%>
13 <jsp:useBean id = "addressData" scope = "request"
14    class = "com.deitel.jhtp5.jsp.beans.AddressDataBean" />
15
16 <html xmlns = "http://www.w3.org/1999/xhtml">
17    <head>
18       <title>Address List</title>
19
20       <style type = "text/css">
21          body {
22             font-family: tahoma, helvetica, arial, sans-serif;
23          }
```

```
24
25          table, tr, td, th {
26             text-align: center;
27             font-size: .9em;
28             border: 3px groove;
29             padding: 5px;
30             background-color: #dddddd;
31          }
32       </style>
33    </head>
34
35    <body>
36       <p style = "font-size: 2em;">Address List</p>
37
38       <table>
39          <thead>
40             <tr>
41                <th style = "width: 100px;">Last name</th>
42                <th style = "width: 100px;">First name</th>
43                <th style = "width: 200px;">Address</th>
44             </tr>
45          </thead>
46
47          <tbody>
48
49 <% // start scriptlet
50
51    List addressList = addressData.getAddressList();
52    Iterator addressListIterator = addressList.iterator();
53    AddressBean address;
54
55    while ( addressListIterator.hasNext() ) {
56       address = ( AddressBean ) addressListIterator.next();
57
58 %> <%-- end scriptlet; insert fixed template data --%>
59
60                <tr>
61                   <td><%= address.getLastName() %></td>
62
63                   <td><%= address.getFirstName() %></td>
64
65                   <td><%= address.getAddress() %></td>
66                </tr>
67
68 <% // continue scriptlet
69
70    } // end while
71
72 %> <%-- end scriptlet --%>
73
74          </tbody>
75       </table>
76
77       <!-- links to view, delete and search address book -->
78       <br /><a href = "addressBookView.jsp">View Address Book</a>
```

```
79          <br /><a href = "addressBookRemove.jsp">Remove Entry</a>
80          <br /><a href = "addressBookSearch.jsp">Search Address Book</a>
81
82       </body>
83    </html>
```

```
1     <?xml version = "1.0"?>
2     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5     <!-- Exercise 25.6 solution: addressBookRemove.jsp -->
6
7     <%-- page settings --%>
8     <%@ page errorPage = "addressBookErrorPage.jsp" %>
9     <%@ page import = "java.util.*" %>
10    <%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>
11
12    <%-- beans used in this JSP --%>
13    <jsp:useBean id = "address" scope = "request"
14       class = "com.deitel.jhtp5.jsp.beans.AddressBean" />
15    <jsp:useBean id = "addressData" scope = "request"
16       class = "com.deitel.jhtp5.jsp.beans.AddressDataBean" />
17
18    <html xmlns = "http://www.w3.org/1999/xhtml">
19       <head>
20          <title>Address List</title>
21
22          <style type = "text/css">
23             body {
24                font-family: tahoma, helvetica, arial, sans-serif;
25             }
26
27             table, tr, td, th {
28                text-align: center;
29                font-size: .9em;
30                border: 3px groove;
31                padding: 5px;
32                background-color: #dddddd;
33             }
34          </style>
35       </head>
36
37       <body>
38          <jsp:setProperty name = "address" property = "*" />
39
40    <% // start scriptlet
41
42       if ( address.getFirstName() == null ||
43          address.getLastName() == null ) {
44
45    %> <%-- end scriptlet to insert fixed template data --%>
46
```

```
47          <form method = "post" action = "addressBookRemove.jsp">
48
49             <p>Enter your first name and last name to search
50                in our address book.</p>
51
52             <table>
53                <tr>
54                   <td>First name</td>
55
56                   <td>
57                      <input type = "text" name = "firstName" />
58                   </td>
59                </tr>
60
61                <tr>
62                   <td>Last name</td>
63
64                   <td>
65                      <input type = "text" name = "lastName" />
66                   </td>
67                </tr>
68
69                <tr>
70                   <td colspan = "2">
71                      <input type = "submit"
72                         value = "Submit" />
73                   </td>
74                </tr>
75             </table>
76          </form>
77
78    <% // continue scriptlet
79
80       }  // end if
81       else {
82
83          addressData.removeEntry( address );
84
85    %>
86
87          <!-- links to view, delete and search address book -->
88          <br /><a href = "addressBookView.jsp">View Address Book</a>
89          <br /><a href = "addressBookRemove.jsp">Remove Entry</a>
90          <br /><a href = "addressBookSearch.jsp">Search Address Book</a>
91
92    <% // continue scriptlet
93
94       }  // end else
95
96    %> <%-- end scriptlet --%>
97
98       </body>
99    </html>
```

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Exercise 25.6 solution: addressBookSearch.jsp -->

<%-- page settings --%>
<%@ page errorPage = "addressBookErrorPage.jsp" %>
<%@ page import = "java.util.*" %>
<%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>

<%-- beans used in this JSP --%>
<jsp:useBean id = "address" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.AddressBean" />
<jsp:useBean id = "addressData" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.AddressDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">

   <head>
      <title>Address Book Search</title>

      <style type = "text/css">
         body {
            font-family: tahoma, helvetica, arial, sans-serif;
         }

         table, tr, td, th {
            text-align: center;
            font-size: .9em;
            border: 3px groove;
            padding: 5px;
            background-color: #dddddd;
         }
      </style>
   </head>

   <body>
      <jsp:setProperty name = "address" property = "*" />

<% // start scriptlet

   if ( address.getFirstName() == null ||
      address.getLastName() == null ) {

%> <%-- end scriptlet to insert fixed template data --%>

         <form method = "post" action = "addressBookSearch.jsp">

            <p>Enter your first name and last name to search
               in our address book.</p>

            <table>
               <tr>
```

```
55                      <td>First name</td>
56
57                      <td>
58                         <input type = "text" name = "firstName" />
59                      </td>
60                   </tr>
61
62                   <tr>
63                      <td>Last name</td>
64
65                      <td>
66                         <input type = "text" name = "lastName" />
67                      </td>
68                   </tr>
69
70                   <tr>
71                      <td colspan = "2">
72                         <input type = "submit"
73                            value = "Submit" />
74                      </td>
75                   </tr>
76                </table>
77             </form>
78
79   <% // continue scriptlet
80
81      }  // end if
82      else {
83
84   %>
85         <p style = "font-size: 2em;">Address List</p>
86
87          <table>
88             <thead>
89                <tr>
90                   <th style = "width: 100px;">Last name</th>
91                   <th style = "width: 100px;">First name</th>
92                   <th style = "width: 200px;">Address</th>
93                </tr>
94             </thead>
95
96             <tbody>
97   <%
98          List addressList = addressData.searchEntry( address );
99          Iterator addressListIterator = addressList.iterator();
100         AddressBean addressBean;
101
102         while ( addressListIterator.hasNext() ) {
103            addressBean = ( AddressBean ) addressListIterator.next();
104
105  %> <%-- end scriptlet; insert fixed template data --%>
106
107                  <tr>
108                     <td><%= addressBean.getLastName() %></td>
```

```
109
110                     <td><%= addressBean.getFirstName() %></td>
111
112                     <td><%= addressBean.getAddress() %></td>
113                  </tr>
114
115   <% // continue scriptlet
116
117         } // end while
118
119   %> <%-- end scriptlet --%>
120
121            </tbody>
122         </table>
123
124         <!-- links to view, delete and search address book -->
125         <br /><a href = "addressBookView.jsp">View Address Book</a>
126         <br /><a href = "addressBookRemove.jsp">Remove Entry</a>
127         <br /><a href = "addressBookSearch.jsp">Search Address Book</a>
128
129   <% // continue scriptlet
130
131      }  // end else
132
133   %> <%-- end scriptlet --%>
134
135      </body>
136   </html>
```

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 25.6 solution: addressBookErrorPage.jsp -->
6
7   <%-- page settings --%>
8   <%@ page isErrorPage = "true" %>
9   <%@ page import = "java.util.*" %>
10  <%@ page import = "java.sql.*" %>
11
12  <html xmlns = "http://www.w3.org/1999/xhtml">
13
14     <head>
15        <title>Error!</title>
16
17        <style type = "text/css">
18           .bigRed {
19              font-size: 2em;
20              color: red;
21              font-weight: bold;
22           }
23        </style>
24     </head>
```

```
25
26     <body>
27        <p class = "bigRed">
28
29        <% // scriptlet to determine exception type
30           // and output beginning of error message
31           if ( exception instanceof SQLException )
32        %>
33
34              An SQLException
35
36        <%
37           else if ( exception instanceof ClassNotFoundException )
38        %>
39
40              A ClassNotFoundException
41
42        <%
43           else
44        %>
45
46              An exception
47
48        <%-- end scriptlet to insert fixed template data --%>
49
50           <%-- continue error message output --%>
51           occurred while interacting with the addressbook database.
52        </p>
53
54        <p class = "bigRed">
55           The error message was:<br />
56           <%= exception.getMessage() %>
57        </p>
58
59        <p class = "bigRed">Please try again later</p>
60     </body>
61
62  </html>
```

```
1  // Exercise 25.6 solution: AddressBean.java
2  // JavaBean to store address data in the address book.
3  package com.deitel.jhtp5.jsp.beans;
4
5  public class AddressBean {
6     private String firstName, lastName, address;
7
8     // set the entry's first name
9     public void setFirstName( String name )
10    {
11       firstName = name;
12    }
13
```

```
14      // get the entry's first name
15      public String getFirstName()
16      {
17         return firstName;
18      }
19
20      // set the entry's last name
21      public void setLastName( String name )
22      {
23         lastName = name;
24      }
25
26      // get the entry's last name
27      public String getLastName()
28      {
29         return lastName;
30      }
31
32      // set the antry's email address
33      public void setAddress( String theAddress )
34      {
35         address = theAddress;
36      }
37
38      // get the guest's email address
39      public String getAddress()
40      {
41         return address;
42      }
43
44   }  // end class AddressBean
```

```
1   // Exercise 25.6 solution: AddressDataBean.java
2   // Class AddressDataBean makes a database connection and supports
3   // inserting and retrieving data from the database.
4   package com.deitel.jhtp5.jsp.beans;
5
6   import java.io.*;
7   import java.sql.*;
8   import java.util.*;
9
10  public class AddressDataBean {
11     private Connection connection;
12     private Statement statement;
13
14     // constructor
15     public AddressDataBean() throws Exception
16     {
17        // specify database location
18        System.setProperty( "db2j.system.home", "C:/CloudScape_5.0" );
19
20        // load the Cloudscape driver
21        Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
```

```
22
23          // connect to the database
24          connection = DriverManager.getConnection( "jdbc:db2j:addressbook" );
25
26          // create Statement to query database
27          statement = connection.createStatement();
28       }
29
30       // return an ArrayList of AddressBeans
31       public List getAddressList() throws SQLException
32       {
33          List addressList = new ArrayList();
34          String query = "SELECT firstName, lastName, address FROM guests";
35
36          // obtain list of records
37          ResultSet results = statement.executeQuery( query );
38
39          // get row data
40          while ( results.next() ) {
41             AddressBean address = new AddressBean();
42
43             address.setFirstName( results.getString( 1 ) );
44             address.setLastName( results.getString( 2 ) );
45             address.setAddress( results.getString( 3 ) );
46
47             addressList.add( address );
48          }
49
50          return addressList;
51       }
52
53       // insert an entry in addressbook database
54       public void addEntry( AddressBean address ) throws SQLException
55       {
56          String query = "INSERT INTO guests (firstName, lastName, address)" +
57             " VALUES ('" + address.getFirstName() + "', '" +
58             address.getLastName() + "', '" + address.getAddress() + "')";
59
60          statement.executeUpdate( query );
61       }
62
63       // delete an entry from addressbook database
64       public void removeEntry( AddressBean address ) throws SQLException
65       {
66          String query = "DELETE FROM guests WHERE firstName = '" +
67             address.getFirstName() + "' AND lastName = '" +
68             address.getLastName() + "'";
69
70          statement.executeUpdate( query );
71       }
72
73       // search entries from addressbook database
74       public List searchEntry( AddressBean address ) throws SQLException
75       {
```

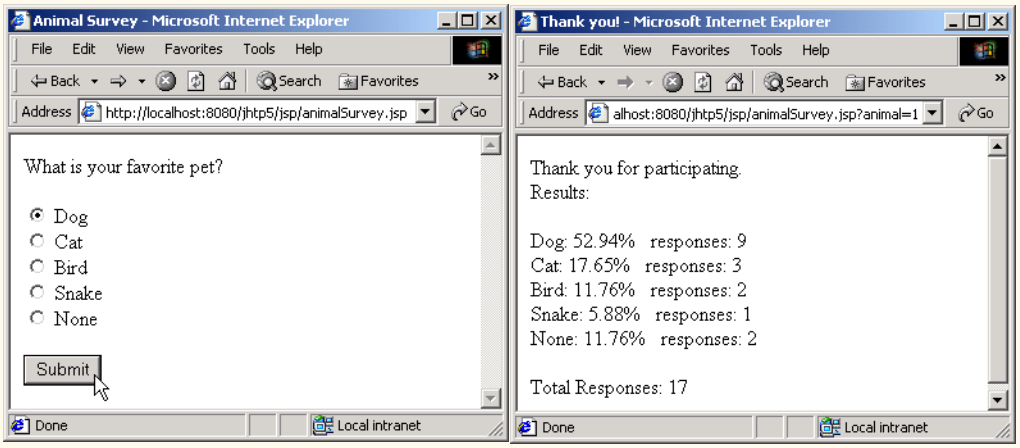
```
76          String query = "SELECT firstName, lastName, address FROM guests " +
77             "WHERE firstName = '" + address.getFirstName() +
78             "' AND lastName = '" + address.getLastName() + "'";
79
80          List searchList = new ArrayList();
81
82          // obtain list of records
83          ResultSet results = statement.executeQuery( query );
84
85          // get row data
86          while ( results.next() ) {
87             AddressBean addressBean = new AddressBean();
88
89             addressBean.setFirstName( results.getString( 1 ) );
90             addressBean.setLastName( results.getString( 2 ) );
91             addressBean.setAddress( results.getString( 3 ) );
92
93             searchList.add( addressBean );
94          }
95
96          return searchList;
97       }
98
99       // close statements and terminate database connection
100      protected void finalize()
101      {
102         // attempt to close database connection
103         try {
104            statement.close();
105            connection.close();
106         }
107
108         // process SQLException on close operation
109         catch ( SQLException sqlException ) {
110            sqlException.printStackTrace();
111         }
112      }
113
114   }  // end class AddressDataBean
```

25.7 Reimplement the Web application of Fig. 24.20 (favorite animal survey) using JSPs.
ANS:

```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 25.7 solution: animalSurvey.jsp -->
6
7   <%-- page settings --%>
8   <%@ page errorPage = "animalSurveyErrorPage.jsp" %>
9   <%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>
10
```

```
<%-- beans used in this JSP --%>
<jsp:useBean id = "voteData" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.VoteDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <title>Animal Survey</title>
</head>

<body>
   <% // begin scriptlet

      String animalType = request.getParameter( "animal" );

      if ( animalType != null ) {

         voteData.addVote( Integer.parseInt( animalType ) );

   %> <%-- end scriptlet to insert fixed template data --%>

         <jsp:forward page = "animalSurveyResults.jsp" />

   <% // continue scriptlet

      }  // end if

      else {

   %> <%-- end scriptlet to insert fixed template data --%>

         <form action = "animalSurvey.jsp" method = "get">
            <p>What is your favorite pet?</p>

            <p>
           <input type = "radio" name = "animal"
              value = "1" /> Dog<br />
           <input type = "radio" name = "animal"
              value = "2" /> Cat<br />
           <input type = "radio" name = "animal"
              value = "3" /> Bird<br />
           <input type = "radio" name = "animal"
              value = "4" /> Snake<br />
           <input type = "radio" name = "animal"
              value = "5" checked = "checked" /> None
        </p>

        <p><input type = "submit" value = "Submit" /></p>
         </form>

   <%  // continue scriptlet

      }  // end else

```

```
65       %> <%-- end scriptlet --%>
66    </body>
67
68    </html>  <!-- end XHTML document -->
```




```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Exercise 25.7 solution: animalSurveyErrorPage.jsp -->
6
7   <%-- page settings --%>
8   <%@ page isErrorPage = "true" %>
9   <%@ page import = "java.util.*" %>
10  <%@ page import = "java.sql.*" %>
11
12  <html xmlns = "http://www.w3.org/1999/xhtml">
13
14     <head>
15        <title>Error!</title>
16
17        <style type = "text/css">
18           .bigRed {
19              font-size: 2em;
20              color: red;
21              font-weight: bold;
22           }
23        </style>
24     </head>
25
26     <body>
27        <p class = "bigRed">
28
29        <% // scriptlet to determine exception type
30           // and output beginning of error message
```

```
31             if ( exception instanceof SQLException )
32          %>
33
34                An SQLException
35
36          <%
37             else if ( exception instanceof ClassNotFoundException )
38          %>
39
40                A ClassNotFoundException
41
42          <%
43             else
44          %>
45
46                An exception
47
48          <%-- end scriptlet to insert fixed template data --%>
49
50             <%-- continue error message output --%>
51             occurred while interacting with the database.
52          </p>
53
54          <p class = "bigRed">
55             The error message was:<br /><%= exception.getMessage() %>
56          </p>
57
58          <p class = "bigRed">Please try again later</p>
59       </body>
60
61    </html>
```

```
1     <?xml version = "1.0"?>
2     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5     <!-- Exercise 25.7 solution: animalSurveyResults.jsp -->
6
7     <%-- page settings --%>
8     <%@ page errorPage = "animalSurveyErrorPage.jsp" %>
9     <%@ page import = "java.util.*" %>
10    <%@ page import = "java.text.*" %>
11    <%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>
12
13    <%-- VoteDataBean to obtain  --%>
14    <jsp:useBean id = "voteData" scope = "request"
15       class = "com.deitel.jhtp5.jsp.beans.VoteDataBean" />
16
17    <html xmlns = "http://www.w3.org/1999/xhtml">
18
19       <head><title>Thank you!</title></head>
20
21       <body>
```

```
22      <p>Thank you for participating.<br />Results:</p>
23
24      <% // start scriptlet
25         DecimalFormat twoDigits = new DecimalFormat( "0.00" );
26         List voteList = voteData.getAnimalVotes();
27         Iterator voteListIterator = voteList.iterator();
28         AnimalBean animal;
29
30         while ( voteListIterator.hasNext() ) {
31            animal = ( AnimalBean ) voteListIterator.next();
32            int votes = voteData.getTotalVotes();
33            String percentage = twoDigits.format(
34               ( double ) animal.getVotes() / votes * 100 );
35
36      %> <%-- end scriptlet; insert fixed template data --%>
37
38            <%= animal.getAnimal() %>:
39            <%= percentage %>%   responses:
40            <%= animal.getVotes() %> <br />
41
42      <% // continue scriptlet
43
44         }  // end while
45
46      %> <%-- end scriptlet --%>
47
48      <p>Total Responses: <%= voteData.getTotalVotes() %></p>
49
50      </body>
51
52   </html>
```

```
1   // Exercise 25.7 solution: AnimalBean.java
2   // JavaBean to store data for animal votes.
3   package com.deitel.jhtp5.jsp.beans;
4
5   public class AnimalBean {
6      private String animal;
7      private int votes;
8
9      // set animal name
10     public void setAnimal( String name )
11     {
12        animal = name;
13     }
14
15     // get animal name
16     public String getAnimal()
17     {
18        return animal;
19     }
20
```

```
21      // set number of votes
22      public void setVotes( int number)
23      {
24         votes = number;
25      }
26
27      // get the guest's last name
28      public int getVotes()
29      {
30         return votes;
31      }
32
33   }  // end class AnimalBean
```

```
1   // Exercise 25.7 solution: VoteDataBean.java
2   // Class VoteDataBean makes a database connection and supports
3   // inserting and retrieving from the database.
4   package com.deitel.jhtp5.jsp.beans;
5
6   import java.io.*;
7   import java.sql.*;
8   import java.util.*;
9
10  public class VoteDataBean {
11     private Connection connection;
12     private Statement statement;
13
14     // set up database connection and prepare SQL statements
15     public VoteDataBean() throws Exception
16     {
17        // specify database location
18        System.setProperty( "db2j.system.home", "C:/CloudScape_5.0" );
19
20        // load the Cloudscape driver
21        Class.forName( "com.ibm.db2j.jdbc.DB2jDriver" );
22
23        // connect to the database
24        connection = DriverManager.getConnection(
25           "jdbc:db2j:animalsurvey" );
26
27        // create Statement to query database
28        statement = connection.createStatement();
29     }
30
31     // update votes
32     public void addVote( int value ) throws SQLException
33     {
34        // update total for current survey response
35        String query = "UPDATE surveyresults SET votes = votes + 1 " +
36           "WHERE id = " + value;
37        statement.executeUpdate( query );
38     }
39
```

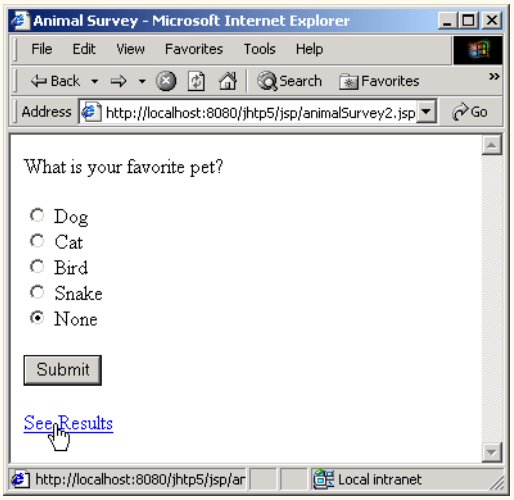
```
40      // return an ArrayList of AnimalBeans
41      public ArrayList getAnimalVotes() throws SQLException
42      {
43         ArrayList voteList = new ArrayList();
44
45         // get results
46         String query = "SELECT surveyoption, votes, id FROM surveyresults" +
47               " ORDER BY id";
48         ResultSet resultsRS = statement.executeQuery( query );
49
50         while ( resultsRS.next() ) {
51            AnimalBean animal = new AnimalBean();
52
53            animal.setAnimal( resultsRS.getString( 1 ) );
54            animal.setVotes( resultsRS.getInt( 2 ) );
55
56            voteList.add( animal );
57         }
58
59         return voteList;
60      }
61
62      // get total of all survey responses
63      public int getTotalVotes() throws SQLException
64      {
65         // get total of all survey responses
66         String query = "SELECT sum( votes ) FROM surveyresults";
67         ResultSet totalRS = statement.executeQuery( query );
68         totalRS.next();
69
70         return totalRS.getInt( 1 );
71      }
72
73      // close SQL statements and database connection
74      public void finalize()
75      {
76         // attempt to close statements and database connection
77         try {
78            statement.close();
79            connection.close();
80         }
81
82         // handle database exceptions by returning error to client
83         catch( SQLException sqlException ) {
84            sqlException.printStackTrace();
85         }
86      }  // end of finalize method
87
88   }  // end class VoteDataBean
```

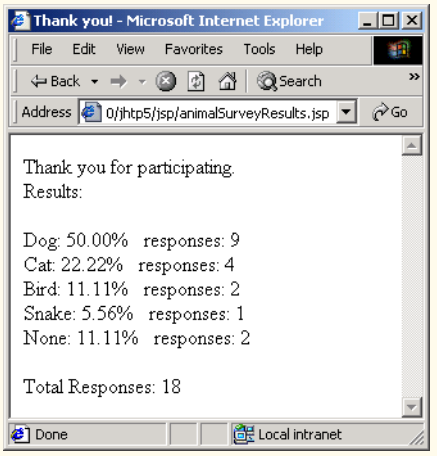
25.8 Modify your solution to Exercise 25.7 to allow the user to see the survey results without responding to the survey.

ANS:

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Exercise 25.8 solution: animalSurvey2.jsp -->

<%-- page settings --%>
<%@ page errorPage = "animalSurveyErrorPage.jsp" %>
<%@ page import = "com.deitel.jhtp5.jsp.beans.*" %>

<%-- beans used in this JSP --%>
<jsp:useBean id = "voteData" scope = "request"
   class = "com.deitel.jhtp5.jsp.beans.VoteDataBean" />

<html xmlns = "http://www.w3.org/1999/xhtml">

<head>
   <title>Animal Survey</title>
</head>

<body>
   <% // begin scriptlet

      String animalType = request.getParameter( "animal" );

      if ( animalType != null ) {

         voteData.addVote( Integer.parseInt( animalType ) );

   %> <%-- end scriptlet to insert fixed template data --%>

         <jsp:forward page = "animalSurveyResults.jsp" />

   <% // continue scriptlet

      }  // end if

      else {

   %> <%-- end scriptlet to insert fixed template data --%>

         <form action = "animalSurvey2.jsp" method = "get">
            <p>What is your favorite pet?</p>

            <p>
            <input type = "radio" name = "animal"
               value = "1" /> Dog<br />
            <input type = "radio" name = "animal"
               value = "2" /> Cat<br />
            <input type = "radio" name = "animal"
               value = "3" /> Bird<br />
            <input type = "radio" name = "animal"
               value = "4" /> Snake<br />
```

```
54                <input type = "radio" name = "animal"
55                   value = "5" checked = "checked" /> None
56                </p>
57
58                <p><input type = "submit" value = "Submit" /></p>
59
60                <p><a href = "animalSurveyResults.jsp">See Results</a></p>
61
62             </form>
63
64       <%  // continue scriptlet
65
66          }  // end else
67
68       %> <%-- end scriptlet --%>
69    </body>
70
71    </html>  <!-- end XHTML document -->
```





Number Systems:

SELF-REVIEW EXERCISES

C.1 The bases of the decimal, binary, octal, and hexadecimal number systems are _____, _____, _____ and _____ respectively.

ANS: 10, 2, 8, 16.

C.2 In general, the decimal, octal and hexadecimal representations of a given binary number contain (more/fewer) digits than the binary number contains.

ANS: Fewer.

C.3 (True/False) A popular reason for using the decimal number system is that it forms a convenient notation for abbreviating binary numbers simply by substituting one decimal digit per group of four binary bits.

ANS: False.

C.4 The (octal / hexadecimal / decimal) representation of a large binary value is the most concise (of the given alternatives).

ANS: Hexadecimal.

C.5 (True/False) The highest digit in any base is one more than the base.

ANS: False. The highest digit in any base is one less than the base.

C.6 (True/False) The lowest digit in any base is one less than the base.

ANS: False. The lowest digit in any base is zero.

C.7 The positional value of the rightmost digit of any number in either binary, octal, decimal, or hexadecimal is always _____.

ANS: 1 (the base raised to the zero power).

C.8 The positional value of the digit to the left of the rightmost digit of any number in binary, octal, decimal, or hexadecimal is always equal to _____.

ANS: The base of the number system.

C.9 Fill in the missing values in this chart of positional values for the rightmost four positions in each of the indicated number systems:

decimal	1000	100	10	1
hexadecimal	...	256	...	...
binary	...	...	...	...
octal	512	...	8	...

ANS:

decimal	1000	100	10	1
hexadecimal	4096	256	16	1
binary	8	4	2	1
octal	512	64	8	1

C.10 Convert binary 110101011000 to octal and to hexadecimal.

ANS: Octal 6530; Hexadecimal D58.

C.11 Convert hexadecimal FACE to binary.

ANS: Binary 1111 1010 1100 1110.

C.12 Convert octal 7316 to binary.

ANS: Binary 111 011 001 110.

C.13 Convert hexadecimal 4FEC to octal. (Hint: First convert 4FEC to binary then convert that binary number to octal.)

ANS: Binary 0 100 111 111 101 100; Octal 47754.

C.14 Convert binary 1101110 to decimal.
ANS: Decimal 2+4+8+32+64=110.

C.15 Convert octal 317 to decimal.
ANS: Decimal 7+1*8+3*64=7+8+192=207.

C.16 Convert hexadecimal EFD4 to decimal.
ANS: Decimal 4+13*16+15*256+14*4096=61396.

C.17 Convert decimal 177 to binary, to octal, and to hexadecimal.
ANS:
Decimal 177
to binary:

```
256 128 64 32 16 8 4 2 1
128 64 32 16 8 4 2 1
(1*128)+(0*64)+(1*32)+(1*16)+(0*8)+(0*4)+(0*2)+(1*1)
10110001
```

to octal:

```
512 64 8 1
64 8 1
(2*64)+(6*8)+(1*1)
261
```

to hexadecimal:

```
256 16 1
16 1
(11*16)+(1*1)
(B*16)+(1*1)
B1
```

C.18 Show the binary representation of decimal 417. Then show the one's complement of 417, and the two's complement of 417.
ANS:
Binary:

```
512 256 128 64 32 16 8 4 2 1
256 128 64 32 16 8 4 2 1
(1*256)+(1*128)+(0*64)+(1*32)+(0*16)+(0*8)+(0*4)+(0*2)+
(1*1)
110100001

One's complement: 001011110
Two's complement: 001011111
Check: Original binary number + its two's complement

110100001
001011111
---------
000000000
```

C.19 What is the result when the one's complement of a number is added to itself?
ANS: Zero.

EXERCISES

C.20 Some people argue that many of our calculations would be easier in the base 12 number system because 12 is divisible by so many more numbers than 10 (for base 10). What is the lowest digit in base 12? What might the highest symbol for the digit in base 12 be? What are the positional values of the rightmost four positions of any number in the base 12 number system?

ANS: The lowest digit is 1. The highest symbol is C. 1728, 144, 12, 1.

C.21 How is the highest symbol value in the number systems we discussed related to the positional value of the first digit to the left of the rightmost digit of any number in these number systems?

ANS:

C.22 Complete the following chart of positional values for the rightmost four positions in each of the indicated number systems:

```
decimal    1000    100     10      1
base 6     ...     ...      6     ...
base 13    ...     169     ...    ...
base 3     27      ...     ...    ...
```

ANS:

```
decimal    1000     100    10    1
base 6      216      36     6    1
base 13    2197     169    13    1
base 3       27       9     3    1
```

C.23 Convert binary 100101111010 to octal and to hexadecimal.

ANS: 4572, 97A.

C.24 Convert hexadecimal 3A7D to binary.

ANS: 11101001111101.

C.25 Convert hexadecimal 765F to octal. (Hint: First convert 765F to binary, then convert that binary number to octal.)

ANS: 73137.

C.26 Convert binary 1011110 to decimal.

ANS: 94.

C.27 Convert octal 426 to decimal.

ANS: 278.

C.28 Convert hexadecimal FFFF to decimal.

ANS: 65535.

C.29 Convert decimal 299 to binary, to octal, and to hexadecimal.

ANS: 100101011, 453, 12B.

C.30 Show the binary representation of decimal 779. Then show the one's complement of 779, and the two's complement of 779.

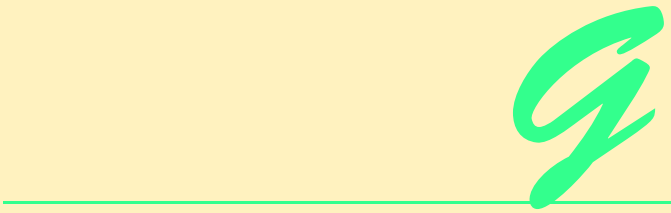
ANS: 8FF.

C.31 What is the result when the two's complement of a number is added to itself?

ANS: 110000101. One's complement: 001111010. Two's complement: 001111011.

C.32 Show the two's complement of integer value -1 on a machine with 32-bit integers.

ANS:



Unicode

SELF-REVIEW EXERCISES

G.1 Fill in the blanks in each of the following.

a) Global software developers had to _____ their products to a specific market before distribution.

ANS: localize.

b) The Unicode Standard is a(n) _____ standard that facilitates the uniform production and distribution of software products.

ANS: encoding.

c) The four design basis that constitute the Unicode Standard are: _____, _____, _____ and _____.

ANS: universal, efficient, uniform, unambiguous.

d) A(n) _____ is the smallest written component the can be represented with a numeric value.

ANS: character.

e) Software that can execute on different operating systems is said to be _____.

ANS: portable.

G.2 State whether each of the following is *true* or *false*. If *false*, explain why.

a) The Unicode Standard encompasses all the world's characters.

ANS: False. It encompasses the majority of the world's characters.

b) A Unicode code value is represented as U+*yyyy*, where *yyyy* represents a number in binary notation.

ANS: False. The yyyy represents a hexadecimal number.

c) A diacritic is a character with a special mark that emphasizes an accent.

ANS: False. A diacritic is a special mark added to a character to distinguish it from another letter or to indicate an accent.

d) Unicode is portable.

ANS: True.

e) When designing Java programs, the escape sequence is denoted by**/u***yyyy*.

ANS: False. The escape sequence is denoted by \u*yyyy*.

EXERCISES

G.3 Navigate to the Unicode Consortium Web site (`www.unicode.org`) and write the hexadecimal code values for the following characters. In which block were they located?

a) Latin letter 'Z.'

ANS: 005A (Basic Latin).

b) Latin letter 'n' with the 'tilde (~).'

ANS: 00F1 (Latin-1 Supplement).

c) Greek letter 'delta.'

ANS: 0394 (Greek).

d) Mathematical operator 'less than or equal to.'

ANS: 2264 (Mathematical Operators).

e) Punctuation symbol 'open quote (").'

ANS: 2036 (General Punctuation).

G.4 Describe the Unicode Standard design basis.

ANS: The Unicode Standard design basis is envisioned to be universal, efficient, uniform and unambiguous. A universal encoding system encompasses all commonly used characters. An efficient encoding system allows text files to be parsed easily. A uniform encoding

system assigns fixed values to all characters. An unambiguous encoding system represents a given character in a consistent manner regardless of where it occurs, or in what context.

G.5 Define the following terms:

a) code value.

ANS: Code value is a bit combination that represents an encoded character. The notation for a code value is U+yyyy in which U+ indicates that the encoding conforms to the Unicode Standard. The yyyy represents the hexadecimal number.

b) surrogates.

ANS: Surrogates are extension mechanisms that allow the Unicode Standard to incorporate additional characters. Surrogates are 16-bit integers in the range D800 through DFFF, which are used solely for the purpose of "escaping" into higher numbered characters. Approximately one million characters can be expressed in this manner. It is used with the UTF-16 encoding form.

c) Unicode Standard.

ANS: The Unicode Standard is an encoding standard that facilitates the uniform production and distribution of software. It outlines a specification to produce the consistent encoding of characters and symbols. Characters and symbols are assigned a unique four-digit hexadecimal code value, which differentiate them from other characters.

G.6 Define the following terms:

a) UTF-8.

ANS: UTF-8, a variable width encoding form, requires one to four bytes to express each Unicode character. UTF-8 data consists of 8-bit bytes (sequences of one, two, three or four bytes depending on the character being encoded) and is well suited for ASCII-based systems when there is a predominance of one-byte characters (ASCII represents characters as one-byte).

b) UTF-16.

ANS: The variable width UTF-16 encoding form expresses Unicode characters in units of 16 bits. Most characters of Unicode are expressed in a single 16-bit unit. However, characters with values above FFFF hexadecimal are expressed with an ordered pair of 16-bit units called surrogates.

c) UTF-32.

ANS: UTF-32 is a 32-bit, fixed-width encoding form that usually requires twice as much memory as UTF-16 encoded characters. The major advantage of the UTF-32 encoding form is that it uniformly expresses all characters, so it is easy to handle in arrays.

G.7 Describe a scenario where it is optimal to store your data in UTF-16 format.

ANS: A situation where it is optimal to use UTF-16 is in documents that require the double-byte character set to encode characters. For instance, the Asian ideographs require 16 bits for representation.

G.8 Using the Unicode Standard code values, write a Java program that prints your first and last name. The program should print your name in all uppercase letters and in all lowercase letters. If you know other languages, print your first and last name in those languages as well.

ANS:

```
1  // Exercise G.8 solution: UnicodeName.java
2  // Display name using Unicode code value.
3  import javax.swing.*;
4
```

```
5  public class UnicodeName {
6
7     public static void main( String args[] )
8     {
9        String upperCaseName =
10          "\u0054\u0045\u004D\u0020\u004E\u0049\u0045\u0054\u004F";
11       String lowerCaseName =
12          "\u0074\u0065\u006D\u0020\u006E\u0069\u0065\u0074\u006F";
13       String output = "Uppercase Name: " + upperCaseName +
14          "\nLowercase Name: " + lowerCaseName;
15
16       JOptionPane.showMessageDialog( null, output );
17
18       System.exit( 0 );
19    }
20
21 }  // end class Unicode
```