Solutions Manual for Java, Java, Java, 3E

Ralph Morelli and Ralph Walde
with Jamie Mazur, Meisha Morelli, and Alicia Morelli

June 9, 2006

Preface

This *Solutions Manual* contains solutions for all the end of chapter exercises in *Java, Java, Java, 3E*, including the complete source code for all programming exercises.

Each of the solutions has been checked for accuracy. The source code has been thoroughly documented and tested. The programming solutions contain a brief explanation of the main design issues considered in developing the solution when appropriate, and some of the solutions include figures showing their graphical interface or the sample output they produce. UML (Unified Modeling Language) diagrams are provided for exercises that require them.

The Prentice Hall Companion Website for instructors of the Java, Java, Java 3E text contains the source code for all the programming solutions. The programs are organized into directories that correspond to the book's chapters. The Instructor's Companion Website for the text also contains a collection of laboratory exercises and solutions coordinated with the chapters of the text. The lab exercises without solutions are also provided on the text's Companion Website.

We would greatly appreciate any comments, suggestions, or corrections you may have on any of the ancillary materials that are available with *Java, Java, Java, 3E*. The best way to reach us is by e-mail at: **ralph.morelli@trincoll.edu** I will try to get back to you as quickly as possible.

Writing and producing a textbook together with its supporting materials is an enormous team effort, and I would like to thank the entire team of professionals at Prentice-Hall. In terms of this solutions manual, I especially want to commend the work done on the earlier editions by my student assistants: Jamie Mazur, Trinity College, 2000, Meisha Morelli (my daughter), Amherst College, 2002, and Alicia Morelli (my daughter), Trinity College, 2006. Jamie and Meisha wrote the first draft of many of the solutions, and Alicia did extensive work on the UML diagrams. Their work comprised the bulk of the solutions for this document which was edited by the authors for this edition.

R. Morelli and R. Walde, June 2006

Contents

Chapter 0

Computers, Objects, and Java

1. Fill in the blanks in each of the following statements.

   (a) Dividing a problem or a task into parts is an example of the _____ principle. **Answer: divide and conquer**

   (b) Designing a class so that it shields certain parts of an object from other objects is an example of the _____ principle. **Answer: information hiding**

   (c) The fact that Java programs can run without change on a wide variety of different kinds of computers is an example of _____. **Answer: platform independence**

   (d) The fact that social security numbers are divided into three parts is an example of the _____ principle. **Answer: abstraction**

   (e) To say that a program is robust means that _____. **Answer: errors in the program don't usually cause system crashes**

   (f) An _____ is a separate module that encapsulates a Java program's attributes and actions. **Answer: object**

2. Explain the difference between each of the following pairs of concepts.

   (a) *hardware* and *software*

   Answer: A computer's *hardware* consists of its physical components, such as its processor and memory. Its *software* includes the programs that control the hardware.

   (b) *systems* and *application* software

   Answer: *Systems* software, such as the operating system, is the software that gives the computer its general usability. *Application* software consists of programs that perform specific tasks, such as word processing.

   (c) *compiler* and *interpreter*

   Answer: A *compiler* translates an entire source code program into object code, which can then be executed. An *interpreter* translates and executes one line of a program at a time.

(d) *machine language* and *high-level language*

Answer: A computer's *machine language* is the instruction set that its processor understands directly. A *high-level language*, such as Java, is a human-readable computer language.

(e) *general-purpose* and *special-purpose* computer

Answer: A *general-purpose* computer can change its control programs so that it can perform a wide variety of tasks. A *special-purpose* computer, such as the processor that controls a digital watch, comes with a fixed program that performs a single task.

(f) *primary* and *secondary* memory

Answer: A computer's *primary* memory, sometimes called RAM for random access memory, temporarily stores programs and data while the computer is running. Its *secondary* memory consists of disk drives and other devices that are designed for long-term and permanent storage of data.

(g) the *CPU* and the *ALU*

Answer: The *CPU*, central processing unit, is responsible for executing all of the computers instructions. Under the direction of a program, it controls the computer's overall behavior. The *ALU*, arithmetic-logic unit, is that part of the CPU that is responsible for arithmetic and logic operations.

(h) the *Internet* and the *WWW*

Answer: The *Internet* is a global network of different networks. The *WWW* is that subset of the Internet that supports multimedia document exchange through the use of the HyperText Transfer Protocol (HTTP).

(i) a *client* and a *server*

Answer: In a networked environment a *server* is a computer that provides a particular service, such as electronic mail or web page access. A *client* computer is one that contains software, such as an email program or web browser, that lets a user access a particular service.

(j) *HTTP* and *HTML*

Answer: *HTTP* stands for HyperText Transfer Protocol, the set of rules that govern how web pages are exchanged and interpreted. *HTML*, which stands for HyperText Markup Language, is the formal language used to codify web pages.

(k) *source* and *object* code

Answer: *Source code* refers to an untranslated high-level language program, such as a Java program. *Object code* refers to the result of translating a source program into a machine-readable format.

3. Fill in the blanks in each of the following statements.

(a) A _____ is a set of instructions that directs a computer's behavior. **Answer: computer program**

(b) A disk drive would be an example of a _____ device. **Answer: secondary memory**

(c) A mouse is an example of an _____ device. **Answer: input**

(d) A monitor is an example of an _____ device. **Answer: output**

(e) The computer's _____ functions like a scratch pad. **Answer: primary memory**

(f) Java is an example of a _____ programming language. **Answer: high-level**

(g) The Internet is a network of _____. **Answer: networks**

(h) The protocol used by the World Wide Web is the _____ protocol. **Answer: HTTP or HyperText Transfer Protocol**

(i) Web documents are written in _____ code. **Answer: HTML or HyperText Markup Language**

(j) A _____ is a networked computer that is used to store data for other computers on the network. **Answer: file server**

4. Identify the component of computer hardware that is responsible for the following functions.

   (a) fetch-execute cycle **Answer: The CPU, central processing unit**

   (b) arithmetic operations **Answer: the ALU, arithmetic-logic unit**

   (c) executing instructions **Answer: the CPU, central processing unit**

   (d) storing programs while they are executing **Answer: primary memory**

   (e) storing programs and data when the computer is off **Answer: secondary memory**

5. Explain why a typical piece of software, such as a word processor, cannot run on both a Macintosh and a Windows machine.

   Answer: Macintosh and Windows computers constitute different hardware and software platforms. A typical piece of software uses features of a particular operating system (such as Windows) and a particular hardware environment (such as Intel). This is known as *platform dependence*.

6. What advantages do you see in platform independence? What disadvantages?

   Answer: Platform independence means that a single program can run without further translation on a wide variety of computers. One advantage is that this would greatly simplify software development. One disadvantage is that it is very difficult to optimize a platform independent program for a particular hardware or software platform. Therefore, such programs may not be as efficient as platform-dependent programs.

7. In what sense is a person's name an *abstraction*? In what sense is any word of the English language an abstraction?

   Answer: Imagine what it would be like if we had to use a description, rather that a name, to refer to someone. That would make our language very unwieldy. Similarly, words name concepts. When we lack a word for a concept we have to describe the concept. When we define a particular word, it can take the place of its definition, thereby streamlining our language.

8. Analyze the process of writing a term paper in terms of the divide-and-conquer and encapsulation principles.

   Answer: A term paper is organized into a collection of sections, each of which has a clear purpose and contributes the overall goal of the paper. The introduction should introduce the topic and, perhaps, provide an overview of how the paper will develop the topic. The conclusion should summarize the paper's conclusions and perhaps review some of the main points. The main body of the paper should be organized into separate sections, each of which has a clearly identified purpose, and each of which contributes to the overall effectiveness of the paper as a whole.

9. Analyze your car in terms of object-oriented design principles. In other words, pick one of your car's systems, such as the braking system, and analyze it in terms of the divide-and-conquer, encapsulation, information, and interface principles.

   Answer: The fact that your car is organized into different systems is an example of the divide-and-conquer principle at work. Each system has a specific, well-identified purpose. The purpose of the braking system is to stop the car. Each module must provide a particular interface other modules that interact with it. For example, the user interface for the braking system consists of a floor pedal, which must be depressed in order to stop the car. The internal details of the braking system are hidden from the everyday driver, who needn't know, for example, whether the car uses disk or drum brakes to use its braking system effectively. All the driver needs to know is what is required to operate the interface.

10. Make an object-oriented analysis of the interaction between a student, a librarian, and a library database when a student checks a book out of a college library.

    Answer: A college library will have a well-defined set of rules on how a student checks out a book that define an interface between the student, librarian, and the library database program. It describes what information the student must provide to the librarian and what actions the librarian must take with some input devises connected to the library database so that the student may borrow the requested book. The library database program uses the information-hiding principle to hide most details of what it does from both the librarian and the student. The librarian may press a couple of keyboard keys and scan the student's ID card and the book with an optical scanner and the database program will report that the book is checked out to the student. The program hides from the librarian and student

what information is being changed in the database and what operations are taken in the program to accomplish this.

Chapter 1

Java Program Development

1. Fill in the blanks in each of the following statements.

   (a) A Java class definition contains an object's _____ and _____ .
   Answer: instance variables, methods

   (b) A stub class is one which contains a proper _____ but an empty _____ .
   Answer: Answer: heading, body

2. Explain the difference between each of the following pairs of concepts.

   (a) *Application* and *applet*.
   Answer: A Java *application* runs in stand-alone mode. A Java *applet* is an embedded program that runs within the context of a WWW browser.

   (b) *Single-line* and *multiline* comment.
   Answer: A *single line comment* begins with // and is placed at the end of a line of code to clarify the line. A *multiline comment* also provides clarification as well as important information about the program. It may extend over several lines and it begins with /* and ends with */.

   (c) *Compiling* and *running* a program.
   Answer: *Compiling* a program translates it from Java language statements into Java bytecode. *Running* a program uses a Java Virtual Machine (JVM) to interpret and execute the bytecode.

   (d) *Source file* and *bytecode* file.
   Answer: A *source* file is the program written in Java language statements. Source code must be translated into a *bytecode* file in order to be understood by the JVM.

   (e) *Syntax* and *semantics*.
   Answer: *Syntax* is the set of rules that determine whether a particular statement is correctly formulated. *Semantics* refers to the meaning of a program statement – that is, what action the statement takes.

(f) *Syntax error* and *semantic error*.

Answer: A *syntax error* is an incorrectly formulated statement that cannot be read by the Java compiler. A *semantic error* is an error in the logic of the program that will cause it to run incorrectly. A semantic error will not be detected by the compiler. The program may still run, but will not produce the desired results.

(g) *Data* and *methods*.

Answer: *Data* are ways of representing information needed to run the program. *Methods* are sections of code that manipulate information and perform particular actions.

(h) *Variable* and *method*.

Answer: A *variable* is a memory location in which a datum, such as an integer or a character, may be stored. A *method* is a section of code that manipulates this information and performs a particular action.

(i) *Algorithm* and *method*.

Answer: An *algorithm* is a step-by-step description of the solution to a problem. A *method* is a named section of code that performs a particular operation.

(j) *Pseudocode* and *Java code*.

Answer: *Pseudocode* is a hybrid between English and Java code that does not pay attention to the Java syntax. Use of a pseudocode makes translating a program into Java code easier.

(k) *Method definition* and *method invocation*.

Answer: *Method definition* is the task a particular method is written to perform. *Method invocation* is the invoking, or calling on, a particular method to perform its designated task.

3. For each of the following, identify it as either a syntax error or a semantic error. Justify your answers.

(a) Write your class header as `public Class MyClass`.

Answer: Syntax Error: The keyword `class` must begin with lowercase, `class`.

(b) Define the `init()` header as `public vid init()`.

Answer: Syntax error: `vid` is not a keyword understood by the compiler.

(c) Print a string of five asterisks by `System.out.println("***");`

Answer: Semantic error: It is correctly formulated but only three asterisks will be printed.

(d) Forget the semicolon at the end of a `println()` statement.

Answer: Syntax error: In Java, statements must be terminated with a semicolon.

(e) Calculate the sum of two numbers as `N - M`.

Answer: Semantic error: It is correctly formulated but the difference of *N* and *M* will be calculated instead of the sum.

4. Suppose you have a Java program stored in a file named `Test.java`. Describe the compilation and execution process for this program, naming any other files that would be created.

   Answer: `Test.java` is first translated by the Java compiler from a text file containing Java language statements into a Java bytecode file named `Test.class`. It will then be loaded into the computer's main memory and interpreted and executed by the Java Virtual Machine.

5. Suppose *N* is 15. What numbers would be output by the following pseudocode algorithm? Suppose *N* is 6. What would be output by the algorithm in that case?

```
0. Print N.
1. If N equals 1, stop.
2. If N is even, divide it by 2.
3. If N is odd, triple it and add 1.
4. Go to step 0.
```

   Answer: This was supposed to be the *3N+1* algorithm. But given the way this algorithm is stated, it will result in an infinite sequence being printed out in each case. For *N=15*, the sequence will be 15, 46, 70, 106, 160, 80, 40, 20, 10, 16, 8, 4, 2, 4, 2, ... For *N=6*, the sequence will be 6, 10, 16, 8, 4, 2, 4, 2, ... Perhaps a better algorithm for this exercise would be modify steps 2 and 3 as follows:

```
2: If N is even, divide it by 2 and go to step 0.
3. If N is odd, triple it and add 1 and go to step 0.
```

   Stated this way, the algorithms would print out the sequences 15, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1 for *N=15* and 6, 3, 10, 5, 16, 8, 4, 2, 1 for *N=6*.

6. Suppose *N* is 5 and *M* is 3. What value would be reported by the following pseudocode algorithm? In general, what quantity does this algorithm calculate?

```
0. Write 0 on a piece of paper.
1. If M equals 0, report what's on the paper and stop.
2. Add N to the quantity written on the paper.
3. Subtract 1 from M
4. Go to step 1.
```

   Answer: This algorithm will report 15. In general this algorithm calculates the quantity $M \times N$.

7. **Puzzle Problem**: You are given two different length strings that have the characteristic that they both take exactly one hour to burn. However, neither string burns at a constant rate. Some sections of the strings burn very fast, other sections burn very slow. All you have to work with is a box of matches and the two strings. Describe an algorithm that uses the strings and the matches to calculate when exactly 45 minutes has elapsed.

   Answer:

```
0. Light both ends of the first string and
   one end of the second string at the same time.

1. When the first string is finished burning,
   light the unlit end of the second string.

2. When the second string is finished burning
   exactly 45 minutes will have passed.
```

8. **Puzzle Problem**: A polar bear that lives right at the North Pole can walk due south for one hour, due east for one hour, and due north for one hour, and end up right back where it started. Is it possible to do this anywhere else on earth? Explain.

   Answer: Yes. There are an infinite number of other locations on earth where this is possible. For example, pick a latitude near the South Pole where the circumference of the earth takes exactly 1/2 hour to traverse. Start at a point that is 1 hour north of this latitude. From this point (and there are an infinite number of such points) you can walk one hour due south, then one hour due east (making 2 revolutions of the earth and arriving back where you turned east), and then one hour due north. You will arrive back at the same spot where you started.

9. **Puzzle Problem**: Lewis Carroll, the author of *Alice in Wonderland*, used the following puzzle to entertain his guests: A captive Queen weighing 195 pounds, her son weighing 90 pounds, and her daughter weighing 165 pounds, were trapped in a very high tower. Outside their window was a pulley and rope with a basket fastened on each end. They managed to escape by using the baskets and a 75-pound weight they found in the tower. How did they do it? The problem is anytime the difference in weight between the two baskets is more than 15 pounds, someone might get killed. Describe an algorithm that gets them down safely.

   Answer:

```
                                                          TOWER, GROUND
0. QDSW are in the tower.                                  QDSW, -
1. Lower the 75 lb. wgt, empty basket comes up. (75/0)     QDS , W
2. Lower the 90 lb. prince as 75 lb. wgt. goes up. (90/75) QD W, S
3. Lower the 75 lb. wgt., empty basket comes up. (75/0)    QD  , SW
4. Lower the daughter, as son & wgt go up. (165/165)       Q SW, D
```

```
 5. Lower the 75 lb. wgt., empty basket comes up. (75/0)      Q S , DW
 6. Lower the son as the wgt. goes up (90/75).                Q  W, DS
 7. Lower queen and wgt., as son and daughter go up.(270/255)  DS , QW
 8. Lower the son as the wgt goes up. (90/75)                  D W, QS
 9. Lower the 75 lb. wgt. (75/0)                               D  , QSW
10. Lower the daughter, as son and wgt. go up. (165/165)        SW, QD
11. Lower the wgt (75/0)                                        S , QDW
12. Lower the son as the wgt. goes up. (90/75)                  - , QDSW
```

Note that this solution allows the occupants to pull on the ropes when the weights are equal (165/165).

10. **Puzzle Problem**: Here's another Carroll favorite: A farmer needs to cross a river with his fox, goose, and a bag of corn. There's a rowboat that will hold the farmer and one other passenger. The problem is that the fox will eat the goose, if they are left alone, and the goose will eat the corn, if they are left alone. Write an algorithm that describes how he got across without losing any of his possessions.

    Answer:

```
                                    Other bank
                                    ----------
 1. Farmer and goose row across     Goose
 2. Farmer rows back
 3. Farmer and fox row across
 4. Farmer and goose row back       Fox
 5. Farmer and corn row across
 6. Farmer rows back                Fox and Corn
 7. Farmer and goose row across     Fox, Corn, Goose, Farmer
```

11. **Puzzle Problem**: Have you heard this one? A farmer lent the mechanic next door a 40-pound weight. Unfortunately the mechanic dropped the weight and it broke into four pieces. The good news is that according to the mechanic, it is still possible to use the four pieces to weigh any quantity between one and 40 pounds on a balance scale. How much did each of the four pieces weigh? (*Hint*: You can weigh a 4-pound object on a balance by putting a 5-pound weight on one side and a 1-pound weight on the other.)

    Answer: The weights were 1 + 3 + 9 + 27 = 40 and here's some of the formulas for weighing various weights. The equation relates the two sides of a balance scale. The unknown is the first value in the equation:

```
X
--
40          = 27 + 9 + 3 + 1
39          = 27 + 9 + 3
38 + 1      = 27 + 9 + 3
```

```
37          = 27 + 9 + 1
36          = 27 + 9
35 + 1      = 27 + 9
...
6  + 3
5  + 3 + 1 = 9
4           = 3 + 1
3           = 3
2  + 1      = 3
1           = 1
```

12. Suppose your little sister asks you to help her calculate her homework average in her science course by showing how to use a pocket calculator. Describe an algorithm that she can use to find the average of 10 homework grades.

    Answer:

```
0. Input the grade for the next homework assignment.
1. Type the plus key.
2. If there are more grades to average, go to 0.
3. Type the equal key to get the total for all the grades.
4. Type the division key.
5. Input 10
6. Type the equal key to obtain the average.
```

13. A Caesar cipher is a secret code in which each letter of the alphabet is shifted by *N* letters to the right, with the letters at the end of the alphabet wrapping around to the beginning. For example, if *N* is 1, "daze" would be written as "ebaf." Describe an algorithm that can be used to create a Caesar encoded message with a shift of 5.

    Answer:

```
Repeat 5 times
   Shift each letter of the message by 1
```

14. Suppose you received the message, "sxccohv duh ixq," which you know to be a Caesar cipher. Figure out what it says and then describe an algorithm that will always find what the message said regardless of the size of the shift that was used.

    Answer:

```
Repeat until the message is readable
   Shift each letter of the encrypted message by 1

   sxccohv duh ixq
   tyddpiw evi jyr
```

```
uzeeqjx fwj kzs
vaffrky gxk lat
wbggslz hyl mbu
xchhtma izm ncv
ydiiunb jan odw
zejjvoc kbo pex
afkkwpd lcp qfy
bgllxqe mdq rgz
chmmyrf ner sha
dinnzsg ofs tib
ejooath pgt ujc
fkppbui qhu vkd
glqqcvj riv wle
hmrrdwk sjw xmf
inssexl tkx yng
jottfym uly zoh
kpuugzn vmz api
lqvvhzo wna bqj
mrwwiap xob crk
nsxxjbq ypc dsl
otyykcr zqd etm
puzzles are fun
```

15. Suppose you're talking to your little brother on the phone and he wants you to calculate his homework average. All you have to work with is a piece of chalk and a very small chalkboard — big enough to write one four-digit number. What's more, although your little brother knows how to read numbers, he doesn't know how to count very well so he can't tell you how many grades there are. All he can do is read the numbers to you. Describe an algorithm that will calculate the correct average under these conditions.

    Answer:

    ```
    0. Divide the chalkboard into two boxes
    1. Write a 0 in each box
    2. Get a grade from your brother
    3. Add it to the number in the first box
    4. Add 1 to the number in the second box
    5. If there are more grades, go to 2.
    6. Divide the number in the first box by the
       number in the second box. That's the average.
    ```

16. Write a *header* for a public applet named `SampleApplet`.

    Answer:

    ```
    public class SampleApplet extends Applet
    ```

17. Write a *header* for a public method named `getName`.

    Answer:

```
public String getName()
```

18. Design a class to represent a geometric rectangle with a given length and width, such that it is capable of calculating the area and the perimeter of the rectangle.

    Answer:

```
Class Name:  Rectangle
Role:  To store length and width and calculate area and perimeter
Attributes
     length:  A variable to store the length of the rectangle
     width:   A variable to store the width of the rectangle
Behaviors
     Rectangle(l,w): A method that creates a Rectange by setting it's
         length and width
     calculateArea(): A method to calculate the area of the rectangle
         using the formula: area = length x width
     calculatePerimeter():   A method to calculate the perimeter of the
         rectangle using the formula: perimeter = 2 x length + 2 x width
```

    A UML diagram for a `Rectangle` class is shown in Figure 1.1.

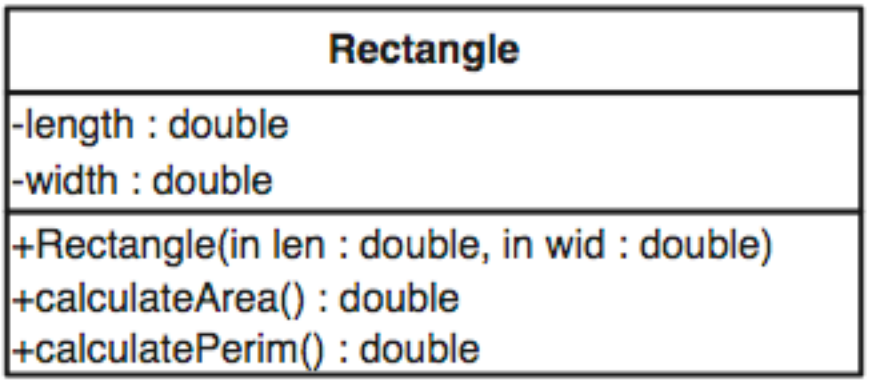


Figure 1.1: UML diagram for the `Rectangle` class.

19. Modify the `OldMacDonald` class to "sing" either *Mary Had a Little Lamb* or your favorite nursery rhyme.

    Answer:

```
/*
```

```
 * File: MaryHadALittleLamb.java
 * Author: Java, Java, Java
 * Description: This class ``sings'' a nursery rhyme.
 */

public class MaryHadALittleLamb
{
    /**
     * main() prints the verse of the nursery rhyme
     */
    public static void main(String argv[]) // Main method
    {
        System.out.println("Mary had a little lamb.");
        System.out.println("Little lamb.");
        System.out.println("Little lamb.");
        System.out.println("Mary had a little lamb.");
        System.out.println("Its fleece was white as snow.");
    } // main()
} // MaryHadALittleLamb
```

20. Define a Java class, called `Patterns`, modeled after `OldMacDonald`, that will print the following patterns of asterisks, one after the other heading down the page:

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

Answer:

```
/*
 * File: Patterns.java
 * Author: Java, Java, Java
 * Description: This class prints a patter of asterisks.
 */

public class Patterns
{
    public static void main(String argv[]) // Main method
    {
        System.out.println("*****");
        System.out.println(" ****");
        System.out.println("  ***");
        System.out.println("   **");
        System.out.println("    *");
        System.out.println();
        System.out.println("*****");
```

```
        System.out.println("*   *");
        System.out.println("*   *");
        System.out.println("*   *");
        System.out.println("*****");
        System.out.println();
        System.out.println("*****");
        System.out.println("* * *");
        System.out.println(" * *");
        System.out.println("* * *");
        System.out.println("*****");
    } // main()
} // Patterns
```

21. Write a Java class that prints your initials as block letters, as shown in the example in the margin.

Fig. 1.2 *The printed initials.*

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

Answer:

```
/*
 * File: Initials.java
 * Author: Java, Java, Java
 * Description: This class prints the block initials RM.
 */

public class Initials
{
    public static void main(String argv[]) // Main method
    {
        System.out.println("******* *       *");
        System.out.println("*     * **     **");
        System.out.println("*     * * *   * *");
        System.out.println("******* *  * *  *");
        System.out.println("**      *   *   *");
        System.out.println("* *     *       *");
        System.out.println("*  *    *       *");
        System.out.println("*   *   *       *");
    } // main()
} // Initials
```

22. **Challenge:** Define a class that represents a `Temperature` object. It should store the current temperature in an instance variable of type `double`, and it should have two `public` methods, `setTemp(double t)`, which assigns `t` to the instance variable, and `getTemp()`, which `return`s the value of the instance variable. Use the `Riddle` class as a model.

Answer: A UML diagram for a `Temperature` class is shown in Figure 13.4.

```
/*
 * File: Temperature.java
 * Author: Java, Java, Java
```

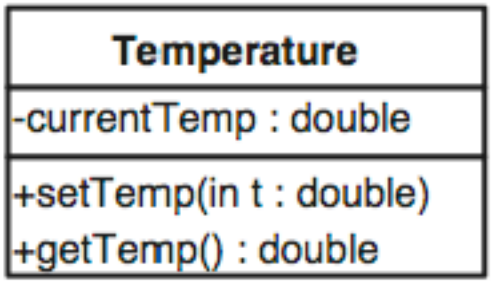


Figure 1.3: UML diagram for the `Temperature` class.

```
 * Description: This class stores the current temperature.
 *  It contains access methods to set and get the temperature.
 */

public class Temperature
{
    private double currentTemp;

    /**
     * setTemp() sets the temperature to the given value
     * @param t -- the given value
     */
    public void setTemp(double t)
    {
        currentTemp = t;
    }

    /**
     * getTemp() returns the current temperature
     */
    public double getTemp()
    {
        return currentTemp;
    }
} // Temperature
```

23. **Challenge:** Define a class named `TaxWhiz` that computes the sales tax for a purchase. It should store the current tax rate as an instance variable. Following the model of the `Riddle` class, you can initialize the rate using a `TaxWhiz()` method. This class should have one `public` method, `calcTax(double purchase)`, which `returns` a `double`, whose value is `purchases` times the tax rate. For example, if the tax rate is 4 percent, 0.04, and the purchase is $100, the `calcTax()` should return 4.0.

    Answer: A UML diagram for a `TaxWhiz` class is shown in Figure 1.4.

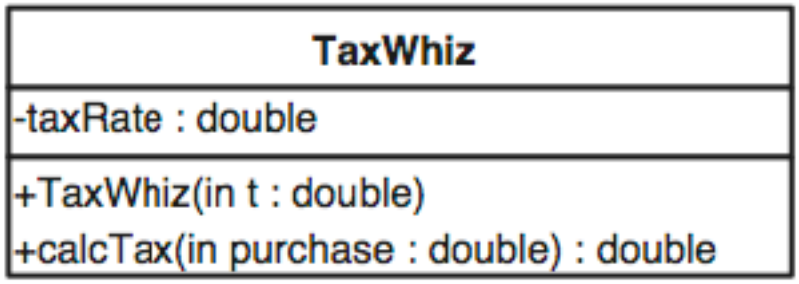


Figure 1.4: UML diagram for the `TaxWhiz` class.

```
/*
 * File: TaxWhiz.java
 * Author: Java, Java, Java
 * Description: This class stores a tax rate and
 *  computes the tax for a given purchase price.
 */

public class TaxWhiz
{
    private double taxRate;

    /**
     * TaxWhiz() constructor creates an object with
     *  a given tax rate
     * @param t -- the given tax rate
     */
    public TaxWhiz(double t)
    {
        taxRate = t;
    }

    /**
     * calcTax() returns the tax for a given purchase
     * @param purchase -- the given purchase price
     */
    public double calcTax(double purchase)
    {
        return taxRate * purchase;
    }
} // TaxWhiz
```

24. What is stored in the variables `num1` and `num2` after the following statements are executed?

```
int num1 = 5;
```

```
int  num2  =  8;
num1  =  num1  +  num2;
num2  =  num1  +  num2;
```

Answer: `num1` contains the value 13 and `num2` contains the value 21.

25. Write a series of statements that will declare a variable of type `int` and store in it the difference between 61 and 51.

    Answer:

```
int  num;
num  =  61 - 51;
```

UML Exercises

26. Modify the UML diagram of the `Riddle` class to contain a method named `getRiddle()` that would return the riddle's question and answer.

    Answer: A UML diagram for the `Riddle` class is shown in Figure 1.5.

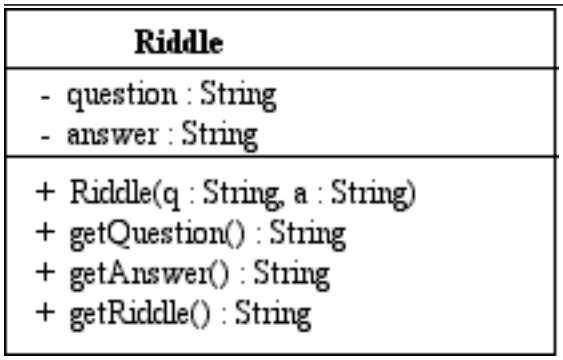


Figure 1.5: The UML diagram for the `Riddle` class.

27. Draw a UML class diagram representing the following class: The name of the class is `Circle`. It has one attribute, a `radius` which is represented by a `double` value. It has one operation, `calculateArea()`, which returns a `double`. Its attributes should be designated as private and its method as public.

    Answer: A UML diagram for a `Circle` class is shown in Figure 1.6.

28. To represent a triangle we need attributes for each of its three sides and operations to create a triangle, calculate its area, and calculate its perimeter. Draw a UML diagram to represent this triangle.

    Answer: A UML diagram for a `Triangle` class is shown in Figure 2.3.

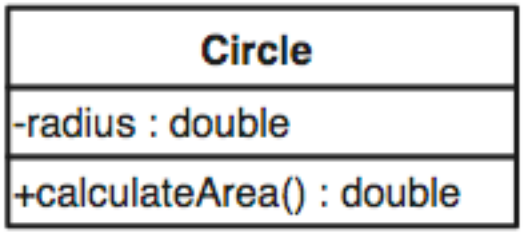


Figure 1.6: UML diagram for the `Circle` class.

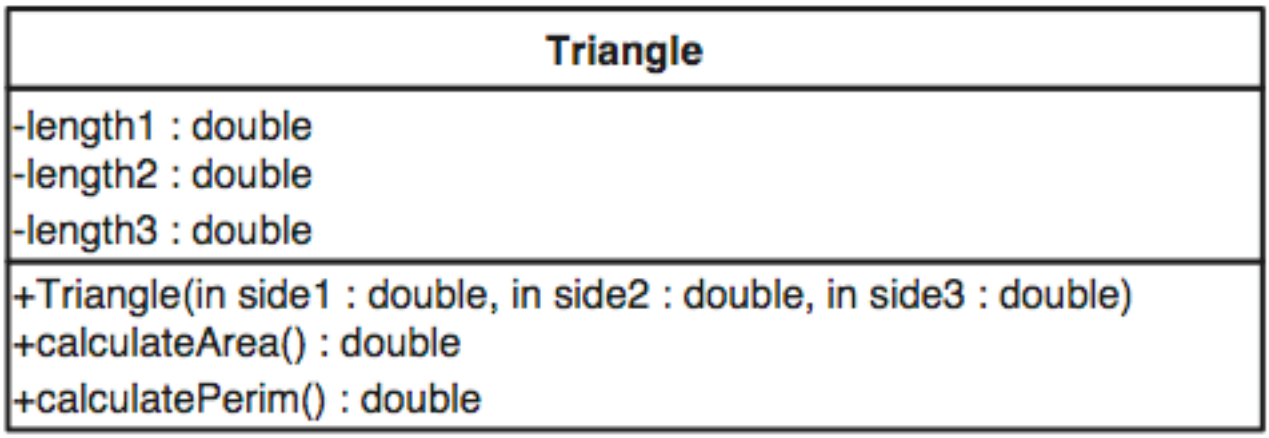


Figure 1.7: UML diagram for the `Triangle class`.

29. Try to give the Java class definition for the class described in the UML diagram shown in Figure 1.8.

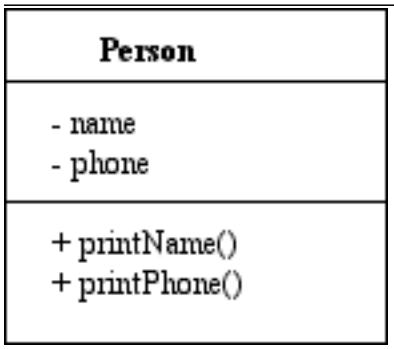


Figure 1.8: The `Person` class

Answer:

```
/*  File: Person.java
 * Author: Java, Java, Java
 * Description: This class represents a person.
 */

public class Person
{
    private String name;
    private String phone;

    /**
     * printName() prints the person's name.
     */
    public void printName()
    {
        System.out.println(name);
    }

    /**
     * printPhone() prints the person's name.
     */
    public void printPhone()
    {
        System.out.println(phone);
    }
} // Person
```

Chapter 2

Objects: Defining, Creating, and Using

1. Consider the transaction of asking your professor for your grade in your computer science course. Identify the objects in this program and the type of messages that would be passed among them.

   Answer: The objects involved in this program are the student and the professor. The messages passed between them would be a request for the course grade by the student and the reporting of the grade to the student by the professor.

2. Now suppose the professor in the previous exercise decides to automate the transaction of looking up a student's grade and has asked you to design a program to perform this task. The program should let a student type in his or her name and ID number and should then display his or her grades for the semester, with a final average. Suppose there are five quiz grades, three exams, and two program grades. Identify the objects in this program and the type of messages that would be passed among them. (*Hint*: The grades themselves are just data values, not objects.)

   Answer: The objects would be a student-record object and the computer program that allows the actual student to obtain his or her grade. The grades would be data contained in the student-record object along with name and ID. The actual student would first input his or her name and ID number and then the program would give the student his or her grade. The program would mediate between the actual student and the student-record object, passing a request from the student to the record and a result from the record to the actual student.

3. In the `RiddleUser` class (Fig. 2.15), give two examples of object instantiation and explain what is being done.

   Answer: The two examples of object instantiation are:

```
Riddle riddle1 =  new Riddle(
    "What is black and white and red all over?",
```

```
      "An embarrassed zebra.");
   Riddle riddle2 =  new Riddle(
      "What is black and white and read all over?",
      "A newspaper.");
```

These two statements create two `Riddle` objects. They reserve memory locations for storing the two riddles and stores `String` values to the `question` and `answer` instance variables for the objects.

4. Explain the difference between a *method definition* and a *method call*. Give an example of each from the `Riddle` and `RiddleUser` discussed in this chapter.

   Answer: Method definition is the part of the code that contains the executable statements that the method performs. Method call is when the method is actually called on to perform the task it is meant to.

```
// Example of a method definition in the Riddle class.
public String getQuestion()
{   return question;
}  // getQuestion()

// Example of a call to the above method in the RiddleUser class.
System.out.println(riddle1.getQuestion());
```

5. In the `RiddleUser` class (Fig 2.15), identify three examples of method calls and explain what is being done.

   Answer:

```
System.out.println("");
        //Prints whatever is in the quotes.
riddle1.getQuestion();
        // Returns the question stored in riddle1.
riddle1.getAnswer();
        // Returns the answer stored in riddle1.
```

6. Describe how the slogan "define, create, use" applies to the `Riddle` example.

   Answer: First the methods and variables within a class are defined (for example `getQuestion()`, and `getAnswer()` are defined in the `Riddle` class), and then instances are created (`riddle1` and `riddle2`) with the `new` operator, and finally the methods are invoked (`riddle1.getQuestion()` and `riddle1.getAnswer()`) and used to use the objects (instances).

7. An *identifier* is the name for a _____, _____, or a _____.

   Answer: class, method, variable

8. Which of the following would be valid *identifiers*?

```
int  74ElmStreet  Big_N     L$&%#   boolean  Boolean  _number
Int  public       Private   Joe     j1       2*K      big numb
```

Answer: `Int, Private, Big_N, Joe, j1`

9. Explain the difference between a `class variable` and `instance variable`.

   Answer: A `class variable` describes a variable associated with the class which exists whether or not objects of the class are created. An `instance variable` describes a variable associated with each object of the class; a separate copy of the variable is created for each object of the class created.

10. Identify the syntax error (if any) in each declaration. Remember that some parts of a field declaration are optional.

```
                // Answers are given as comments
(a)   public boolean isEven ;
                // OK
(b)   Private boolean isEven ;
                // the P in Private should be lowercase
(c)   private boolean isOdd
        // there should be a semicolon at the end of the line
(d)   public boolean is Odd ;
                // there should not be a space in is Odd
(e)   string S ;
                // OK
(f)   public String boolean ;
                // OK
(g)   private boolean even = 0;
        // 0 is not a valid value for a boolean variable
(h)   private String s = helloWorld ;
                // helloWorld should be in quotes
```

11. Write declarations for each of the following instance variables.

```
(a) A private boolean variable named bool that has
an initial value of true.

    private  boolean  bool  =  true;    // Answer

(b) A public string variable named str has an initial
value of ``hello.''

    public  String  str  =  "hello";    // Answer

(c) A private int variable named nEmployees that is
not assigned an initial value.

    private  int  nEmployees ;        // Answer
```

12. Identify the syntax error (if any) in each method header:

```
                    // Answers given as comments.
(a)   public String boolean()
         // boolean is the the name of a primitive data
         //   type and cannot be used as the name of a method
(b)   private void String()
         // OK even though String is a class name
(c)   private void myMethod
         // There should be a set of parentheses after myMethod
(d)   private myMethod()
         // The ResultType is not defined
(e)   public static void Main (String argv[])
         // This is OK but it's a different
         // method than main() with lowercase m.
```

13. Identify the syntax error (if any) in each assignment statement. Assume that the following variables have been declared:

```
  public int m;
 public boolean b;
  public String s;

        // Answers are given as comments.
(a)  m =  "86" ;
        // "86" is not a valid value for an integer variable.
(b)  m =  86 ;
        // OK
(c)  m =  true ;
        // true is not a valid value for an int variable.
(d)  s =  1295 ;
        // The string 1295 needs quotation marks around it.
(e)  s =  "1295" ;
        // OK
(f)   b =  "true" ;
        //  true should not have quotation marks around it.
(g)   b =  false
        // OK
```

14. Given the definition of the `NumberAdder` class, add statements to its `main()` method to create two instances of this class, named `adder1` and `adder2`. Then add statements to set `adder1`'s numbers to 10 and 15, and `adder2`'s numbers to 100 and 200. Then add statements to print their respective sums.

```
/*
 * File: NumberAdder.java
 * Author: Java, Java, Java
 * Description: This class adds its two instance variables
```

```
 *  when its getSum() method is called.
 */
public class NumberAdder
{
    private double num1;
    private double num2;

    /**
     * setNums() sets the two instance variables from
     * the given parameters.
     * @param n1 -- one of the given numbers
     * @param n2 -- the second given number
     */
    public void setNums(double n1, double n2)
    {
        num1 = n1;
        num2 = n2;
    } // setNums()

    /**
     * getSum() returns the sum of the two instance
     * variables.
     */
    public double getSum()
    {
        return num1 + num2 ;
    } // getSum()

    /**
     * main() creates two instances of this class and
     * tests its setNums() and getSum() methods.
     */
    public static void main(String args[])
    {       // Create two instances
        NumberAdder adder1 = new NumberAdder();
        NumberAdder adder2 = new NumberAdder();
            // Set the instances' values
        adder1.setNums(10,15);
        adder2.setNums(100,200);
            // Print the values
        System.out.println("Sum of adder1 " +
                                   adder1.getSum());
        System.out.println("Sum of adder2 " +
                                   adder2.getSum());
    } // main()
} // NumberAdder
```

15. For the `NumberAdder` class in the previous exercise, what are the names of its instance variables and instance methods? Identify three expressions that occur in the program and explain what they do. Identify two assignment statements and

explain what they do.

```
Instance variables = num1, num2
Instance methods = setNums(), getSums()

private double num1;
   // This is an expression defining the variable num1 as
   // double and private
System.out.println("Sum of adder1 " + adder1.getSum());
   // This is an expression that reports the sum of adder1
   // by printing what is in the quotation marks

num1 + num2
   // This is an expression that evaluates to the sum of num1
   // and num2
num1 = n1;
   // This is an assignment statement that sets the value of
   // num1 to n1
num2 = n2;
   // This is an assignment statement that sets the value of
   // num1 to n1
```

16. Explain the difference between each of the following pairs of concepts.

    (a) A *method definition* and a *method invocation*.

    Answer: *Method definition* is the part of the code that contains the executable statements that the method performs. *Method invocation* is the act of calling the method to perform the task it is meant to perform.

    (b) Declaring a reference variable and creating an instance.

    Answer: A reference variable (such as `pet1`) refers to an object. An object is an *instance* of a class and must be created (instantiated) with the `new` operator.

    (c) A variable of reference type and a variable of primitive type.

    Answer: *A variable of reference type* is a variable whose type is a class name. *A variable of primitive type* is a variable of type `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, or `short`.

17. Define a Java class named `NumberCruncher` which has a single `double` value as its only instance variable. Then define methods that perform the following operations on its number: get, double, triple, square, and cube. Set the initial value of the number by following the way the `length` and `width` variables are set in the `Rectangle` class.

    Answer: See the answer to the next problem.

18. Write a `main()` method and add it to the `NumberCruncher` class defined in the previous problem. Use it to create a `NumberCruncher` instance, with a certain initial value, and then get it to report its double, triple, square, and cube.

```
/*
 * File: NumberCruncher.java
 * Author: Java, Java, Java
 * Description: This class stores a number and contains
 *  methods to calculate the number's double, triple, and so on.
 */

public class NumberCruncher
{
    private double num;    // The instance variable

    /**
     * NumberCruncher() constructor creates an instance that
     *  stores the number given as its parameter
     * @param number -- the number that will be stored
     */
    public NumberCruncher(double number)
    {
        num = number;
    }

    /**
     * getNum() returns the object's number
     */
    public double getNum()
    {
        return num;
    }

    /**
     * doubleNum() returns the object's number times 2
     */
    public double doubleNum()
    {
        return num * 2;
    }

    /**
     * tripleNum() returns the object's number times 3
     */
    public double tripleNum()
    {
        return num * 3;
    }

    /**
```

```
     * squareNum() returns the square of the object's number
     */
    public double squareNum()
    {
        return num * num;
    }

    /**
     * cubeNum() returns the cube of the object's number
     */
    public double cubeNum()
    {
        return num * num * num;
    }

    /**
     * main() creates an instance of this class and tests its
       various methods
     */
    public static void main( String args[] )
    {
        NumberCruncher cruncher1 = new NumberCruncher(10);
        System.out.println("Value of num is " + cruncher1.getNum());
        System.out.println("num doubled is " +  cruncher1.doubleNum());
        System.out.println("num tripled is " + cruncher1.tripleNum());
        System.out.println("num squared is " + cruncher1.squareNum());
        System.out.println("num cubed is " + cruncher1.cubeNum());
    } // main()
} // NumberCruncher
```

19. Write a Java class definition for a `Cube` object. The object should be capable of reporting its surface area and volume. The surface area of a cube is six times the area of any side. The volume is calculated by cubing the side.

```
/*
 * File: Cube.java
 * Author: Java, Java, Java
 * Description: This class represents a geometric cube.
 */
public class Cube
{
    private double length;

    /**
     * Cube() constructor creates an instance with length l
     * @param l -- the length of the cube's side
     */
    public Cube(double l)
    {
```

```
        length = l;
    }

    /**
     * getLength() returns this cube's length
     */
    public double getLength()
    {
        return length;
    }

    /**
     * calculateSurfaceArea() returns this cube's surface area
     */
    public double calculateSurfaceArea()
    {
        return 6 * length;
    }

    /**
     * calculateVolume() returns this cube's volume
     */
    public double calculateVolume()
    {
        return length * length * length;
    }
} // Cube
```

20. Write a java class definition for a `CubeUser` object that will use the `Cube` object defined in the previous exercise. This class should create three `Cube` instances, each with a different side, and then report their respective surface areas and volumes.

```
/*
 * File: CubeUser.java
 * Author: Java, Java, Java
 * Description: This class creates 3 instances of the Cube
 *  class and prints their areas and volumes.
 */
public class CubeUser
{
    /**
     * main() -- creates instance of Cube and tests them
     */
    public static void main(String argv[])
    {
        Cube cube1;
        cube1 = new Cube(5);
        Cube cube2;
```

```
        cube2 = new Cube(10);
        Cube cube3;
        cube3 = new Cube(15);
        System.out.println("Side length of cube1 is " +
                                          cube1.getLength());
        System.out.println("Surface Area of cube1 is " +
                              cube1.calculateSurfaceArea ());
        System.out.println("Volume of cube1 is " +
                                          cube1.calculateVolume());
        System.out.println("Side length of cube2 is " +
                                          cube2.getLength());
        System.out.println("Surface Area of cube2 is " +
                              cube2.calculateSurfaceArea ());
        System.out.println("Volume of cube2 is " +
                                          cube2.calculateVolume());
        System.out.println("Side length of cube3 is " +
                                          cube3.getLength());
        System.out.println("Surface Area of cube3 is " +
                              cube3.calculateSurfaceArea ());
        System.out.println("Volume of cube3 is " +
                                          cube3.calculateVolume());
    } // main()
} // CubeUser
```

21. **Challenge:** Modify your solution to the previous exercise so that it lets the user input the side of the cube. Follow the example shown in section that shows how to perform numeric input.

```
/** File: CubeUser2.java
 * Author: Java, Java, Java
 * Description: This class creates 3 instances of the Cube
 *  class and prints their areas and volumes. This version
 *  lets the user input the cubes' sizes.
 */
import java.util.Scanner;
import java.io.*;

public class CubeUser2
{
    /**
     * main() -- creates instance of Cube and tests them. This
     *  version uses a Scanner object to read the user's
     *  keyboard input.
     */
    public static void main(String args[]) throws IOException
    {
        Scanner input = new Scanner(System.in);
        int length;
```

```
        System.out.println("Input the side length of cube1: ");
        length = input.nextInt();
        Cube cube1 = new Cube(length);
        System.out.println("Side length of cube1 is " +
                                              cube1.getLength());
        System.out.println("Surface Area of cube1 is " +
                                cube1.calculateSurfaceArea ());
        System.out.println("Volume of cube1 is " +
                                     cube1.calculateVolume());

        System.out.println("Input the side length of cube2: ");
        length = input.nextInt();
        Cube cube2 = new Cube(length);
        System.out.println("Side length of cube2 is " +
                                              cube2.getLength());
        System.out.println("Surface Area of cube2 is " +
                                cube2.calculateSurfaceArea ());
        System.out.println("Volume of cube2 is " +
                                     cube2.calculateVolume());

        System.out.println("Input the side length of cube3: ");
        length = input.nextInt();
        Cube cube3 = new Cube(length);
        System.out.println("Side length of cube3 is " +
                                            cube3.getLength());
        System.out.println("Surface Area of cube3 is " +
                               cube3.calculateSurfaceArea ());
        System.out.println("Volume of cube3 is " +
                                    cube3.calculateVolume());
    } // main()
} // CubeUser2
```

22. **Challenge:** Define a Java class that represents an address book entry, `Entry`, which consists of a name, address, and phone number, all represented as `String`s. For the class's interface, define methods to set and get the values of each of its instance variables. Thus, for the `name` variable, it should have a `setName()` and a `getName()` method.

```
/*
 * File: Entry.java
 * Author: Java, Java, Java
 * Description: This class represents an address book entry, with
 *  name, address and phone number.
 */
public class Entry
{
    private String name;          // The entry's data
    private String address;
    private String phoneNumber;
```

```
/**
 * setName() sets the name from the given String
 * @param nameIn -- the string giving the name
 */
public void setName(String nameIn)
{
    name = nameIn;
}

/**
 * getName() returns the entry's name
 */
public String getName()
{
    return name;
}

/**
 * setAddress() sets the address from the given String
 * @param nameIn -- the string giving the address
 */
public void setAddress(String addressIn)
{
    address = addressIn;
}

/**
 * getAddress() returns the entry's address
 */
public String getAddress()
{
    return address;
}

/**
 * setPhoneNumber() sets the phone number from the given String
 * @param nameIn -- the string giving the phone number
 */
public void setPhoneNumber(String phoneIn)
{
    phoneNumber = phoneIn;
}

/**
 * getPhoneNumber() returns the entry's phone number
 */
public String getPhoneNumber()
{
    return phoneNumber;
```

```
    }

    /**
     * main() creates an Entry
     */
    public static void main(String args[])
    {
        Entry entry = new Entry();
        entry.setName("Lilly");
        entry.setAddress("Dog House Lane");
        entry.setPhoneNumber("123-456-7890");
        System.out.println("Entry is " + entry.getName() + " "
                                       + entry.getAddress() + " "
                                       + entry.getPhoneNumber());
    } // main()
} // Entry
```

UML Exercises

23. Draw a UML class diagram to represent the following class hierarchy: There are two types of languages, natural languages and programming languages. The natural languages include Chinese, English, French, and German. The programming languages include Java, Smalltalk and C++, which are object-oriented languages, Fortran, Cobol, Pascal, and C, which are imperative languages, Lisp and ML, which are functional languages, and Prolog, which is a logic language.

    Answer: A UML diagram for some different programming languages is shown in Figure 2.1.

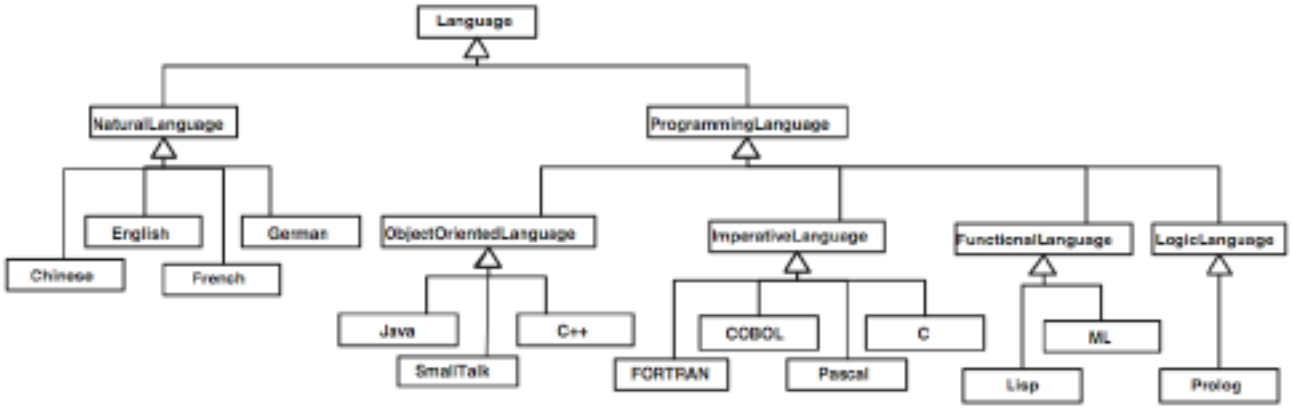


Figure 2.1: UML diagram for some programming languages.

24. Draw a UML class diagram to represent different kinds of automobiles, including trucks, sedans, wagons, SUVs, and the names and manufacturers of some popular models in each category.

    Answer: A UML diagram for some different kinds of automobile types is shown in Figure 2.2.

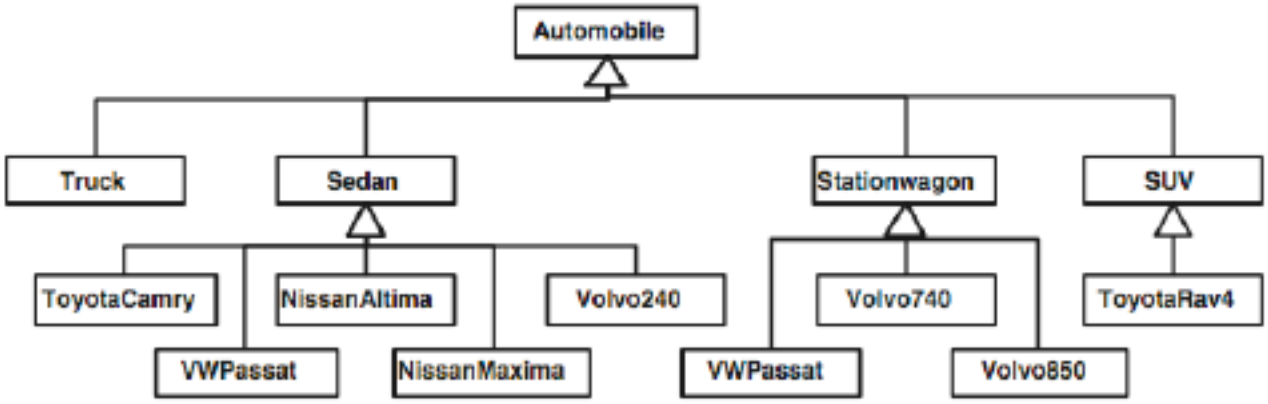


Figure 2.2: UML diagram for some different kinds of automobiles.

25. Draw a UML object diagram of a triangle with attributes for three sides, containing the values 3, 4, and 5.

    Answer: A UML diagram for the triangle is shown in Figure 2.3.

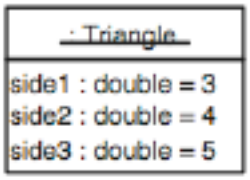


Figure 2.3: UML diagram for a triangle with sides 3, 4, and 5.

26. Suppose you are writing a Java program to implement an electronic address book. Your design is to have two classes, one to represent the user interface and one to represent the address book. Draw a UML diagram to depict this relationship. See Text Figure 2.14.

    Answer: A UML diagram for an address book is shown in Figure 2.4.

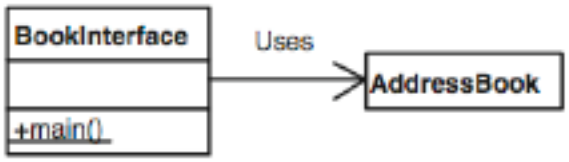


Figure 2.4: UML diagram for an address book.

27. Draw an UML object diagram to depict the relationship between an applet, that serves as a user interface, and three `Triangles`, named `t1`, `t2` and `t3`.

    Answer: A UML diagram for an applet with three triangles is shown in Figure 2.5.

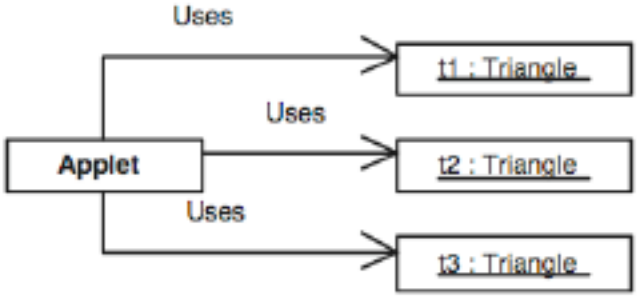


Figure 2.5: UML diagram for an applet with three triangles.

Chapter 3

Methods: Communicating with Objects

1. Fill in the blanks in each of the following sentences:

   (a) When two different methods have the same name, this is an example of _____. **Answer: method overloading**

   (b) Methods with the same name are distinguished by their _____. **Answer: signatures**

   (c) A method that is invoked when an object is created is known as a _____ method. **Answer: constructor**

   (d) A method whose purpose is to provide access to an object's instance variables is known as an _____ method. **Answer: access**

   (e) A `boolean` value is an example of a _____ type. **Answer: primitive**

   (f) A `OneRowNim` variable is an example of a _____ type. **Answer: reference**

   (g) A method's parameters have _____ scope. **Answer: local**

   (h) A class's instance variables have _____ scope. **Answer: class**

   (i) Generally, a class's instance variables should have _____ access. **Answer: private**

   (j) The methods that make up an object's interface should have _____ access. **Answer: public**

   (k) A method that returns no value should be declared _____. **Answer: void**

   (l) Java's if statement and if-else statement are both examples of _____ control structures. **Answer: selection**

   (m) An expression that evaluates to either `true` or `false` is known as a _____. **Answer: boolean expression**

   (n) In an if-else statement, an else clause matches _____. **Answer: the nearest previous unmatched if clause**

(o) The ability to use a superclass method in a subclass is due Java's _____ mechanism. **Answer: inheritance**

(p) The process of redefining a superclass method in a subclass is known as _____ the method. **Answer: overriding**

2. Explain the difference between the following pairs of concepts:

(a) *Parameter* and *argument*.

Answer: A *parameter* is a place holder variable in a method declaration. It is used to store a value that is passed to the method. An *argument* is an actual value that is passed to the method in the method invocation statement.

(b) *Method definition* and *method invocation*.

Answer: A *method definition* is the part of the code that contains the executable statements that the method performs. A *method invocation* is the statement that calls the method to perform the task it is meant to perform.

(c) *Local scope* and *class scope*.

Answer: *Local scope* means the identifier can be used only in the method in which it was declared. *Class scope* means it can be used throughout the class.

(d) *Primitive type* and *reference type*.

Answer: A *primitive type* stores a value of a specific type. A *reference type* stores the address of an object instead of the actual object itself.

(e) *Access method* and *constructor method*.

Answer: A *constructor method* is invoked when an object is created, and it creates an instance of an object. An *access method* provides access to an object's instance variables.

3. Translate each of the following into Java code.

(a) If b1 is true then print "one" otherwise print "two".
Answer:

```
if (b1)                          // Or: if (b1 == true)
    System.out.println("one");
else
    System.out.println("two");
```

(b) If b1 is false then if b2 is true then print "one" otherwise print "two".
Answer:

```
if (b1 == false)
    if (b2)
        System.out.println("one");
    else
        System.out.println("two");
```

(c) If b1 is false then if b2 is true then print ”one” otherwise print ”two” otherwise print ”three.” (with typos fixed).
Answer:

```
if (b1 == false)
    if (b2)
        System.out.println("one");
    else
        System.out.println("two");
else
    System.out.println("three");
```

4. Identify and fix the syntax errors in each of the following.

(a)
```
if ( isWalking == true ) ; //Error: Should be no semicolon
   System.out.println("Walking");
else
   System.out.println("Not walking");
```

(b)
```
if (isWalking)
    System.out.println("Walking") // Error: Semicolon missing
else
    System.out.println("Not walking");
```

(c)
```
if (isWalking)
    System.out.println("Walking");
else
    System.out.println("Not walking") //Error: Semicolon missing
```

(d)
```
if (isWalking = false)  // Error: = is not equality operator
    System.out.println("Walking");
else
    System.out.println("Not walking");
```

5. For each of the following, suppose that `isWalking` is `true` and `isTalking` is `false`. First draw a flowchart for each statement and then determine what would be printed by each statement.

(a)
```
if (isWalking == false)
    System.out.println("One");
    System.out.println("Two");
```

(b)
```
if (isWalking == true)
    System.out.println("One");
    System.out.println("Two");
```

(c)
```
if (isWalking == false)
{
    System.out.println("One");
    System.out.println("Two");
}
```

(d)
```
if (isWalking == false)
    if (isTalking == true)
        System.out.println("One");
    else
        System.out.println("Two");
else
    System.out.println("Three");
```

Answer: Flowcharts for the statements are shown in Figure 3.1.

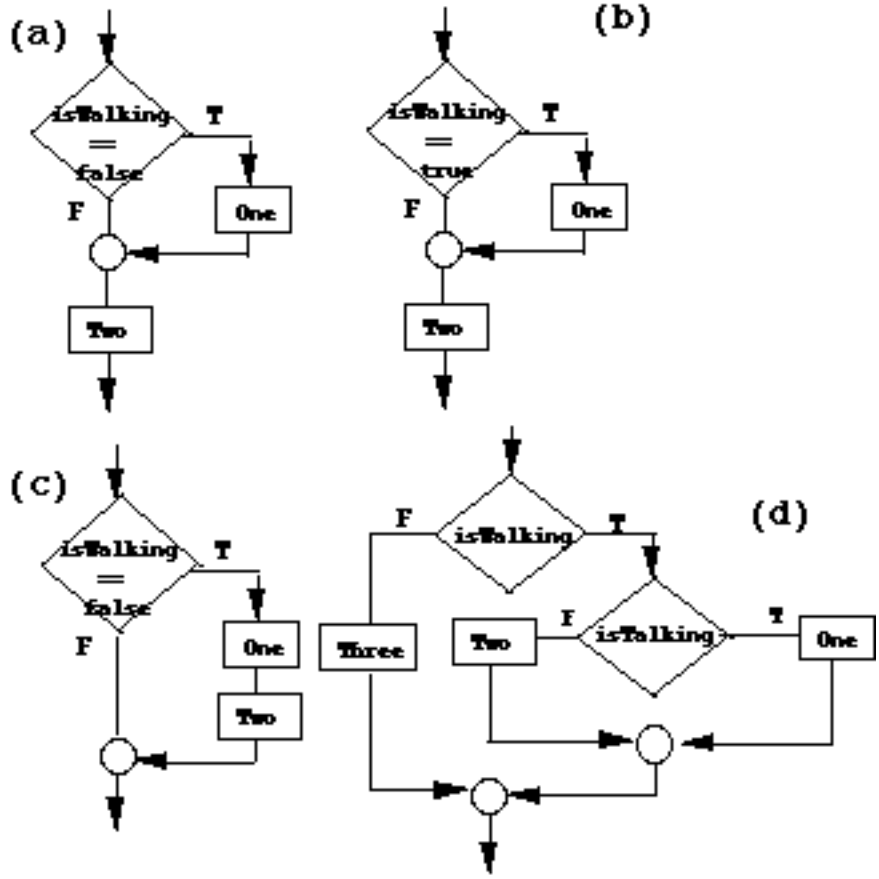


Figure 3.1: Flowcharts for Exercise 5.

The output for the statements is given below:

(a) Output: Two

(b) Output: One Two

(c) Output: No output

(d) Output: Three

6. Show what the output would be if the following version of `main()` were executed.

```
public static void main(String argv[])
{
    System.out.println("main() is starting");
    OneRowNim  game1;
    game1  = new OneRowNim(21);
    OneRowNim  game2;
    game2  = new OneRowNim(8);
    game1.takeSticks(3);
    game2.takeSticks(2);
    game1.takeSticks(1);
    game1.report();
    game2.report();
    System.out.println("main() is finished");
}
```

Answer:

```
Output:
        main() is starting
        Number of sticks left: 17
        Next turn by player: 1
        Number of sticks left: 6
        Next turn by player: 2
        main() is finished
```

7. Determine the output of the following program.

```
public class Mystery
{
    /**
     * myMethod() returns hello and the value of its parameter
     * @param s -- a String giving a person's name
     * @return a String saying ``Hello x'' where x is the value of s
     */
    public String myMethod(String s)
    {
        return("Hello" + s);
    }

    public static void main(String argv[])
    {
        Mystery mystery = new Mystery();
        System.out.println( mystery.myMethod(" dolly");
    }
}
```

Answer: Output: Hello dolly

8. Write a `boolean` method — a method that returns a `boolean` — that takes an `int` parameter and converts the integers 0 and 1 into `false` and `true`, respectively.

   Answer:

```
/**
 * myMethod() converts its int parameter to a boolean
 * @param n -- an int, usually either 0 or 1
 * @return -- a boolean giving true when n is 1 and false otherwise
 */
public boolean myMethod(int n)
{
    if (n == 1)
        return true;
    else
        return false;
}
```

9. Define an `int` method that takes a `boolean` parameter. If the parameter's value is `false`, the method should return 0; otherwise it should return 1.

   Answer:

```
/**
 * myMethod() converts its boolean parameter to an int
 * @param b -- a boolean
 * @return -- a int giving 0 if b is false and 1 otherwise
 */
public int myMethod(boolean b)
{
    if (b == false)
        return 0;
    else
        return 1;
}
```

10. Define a `void` method named `hello` that takes a single `boolean` parameter. The method should print "Hello" if its parameter is true; otherwise it should print "Goodbye."

    Answer:

```
/**
 * hello() prints hello when its parameter is true or else goodbye
 * @param b -- a boolean
 */
public void hello(boolean b)
{
    if (b == true)
```

```
            System.out.println("Hello");
        else
            System.out.println("Goodbye");
    }
```

11. Define a method named `hello` that takes a single `boolean` parameter. The method should return "Hello" if its parameter is true; otherwise it should return "Goodbye." Note the difference between this method and the one in the previous exercise. This one returns a `String`. That one was a `void` method.

    Answer:

```
/**
 * hello() returns hello when its parameter is true or else goodbye
 * @param b -- a boolean
 * @return -- a String giving either hello or goodbye
 */
public String hello(boolean b)
{
    if (b == true)
        return "Hello";
    else
        return "Goodbye";
}
```

12. Write a method named `hello` that takes a single `String` parameter. The method should return a `String` that consists of the word "Hello" concatenated with the value of its parameter. For example, if you call this method with the expression `hello(" dolly")`, it should return "hello dolly." If you call it with `hello(" young lovers wherever you are")`, it should return "hello young lovers wherever you are."

    Answer:

```
/**
 * hello() returns hello plus its parameter
 * @param s -- a String, usually giving a name
 * @return -- a String giving ``hello  x'' where x is the value of s
 */
public String hello(String s)
{
    return "Hello" + s;
}
```

13. Define a void method named `day1` that prints "a partridge in a pear tree."

    Answer:

```
/**
 * day1() prints a string
```

```
     */
    public void day1()
    {
        System.out.println(" a partridge in a pear tree");
    }
```

14. Write a Java application program called `TwelveDays` that prints the Christmas carol "Twelve Days of Christmas." For this version, write a void method name `intro()` that takes a single `String` parameter that gives the day of the verse and prints the intro to the song. For example, `intro("first")`, should print, "On the first day of Christmas my true love gave to me." Then write methods `day1()`, `day2()`, and so on, each of which prints its version of the verse. Then write a `main()` method that calls the other methods to print the whole song.

    Answer:

```
/*
 * File: TwelveDays.java
 * Author: Java, Java, Java
 * Description: This class sings the Twelve Days of
 *  Christmas carol. Its methods are named dayX(),
 *  where X is one of the 12 days. Each such method
 *  prints the verse for that day and then calls the
 *  previous day. So day2() will call day1().
 */

public class TwelveDays
{
    public void intro(String d)
    {
        System.out.println();
        System.out.println("On the " + d +
             " day of Christmas my true love gave to me,");
    }

    public void day1()
    {
        System.out.println("a partridge in a pear tree.");
    }

    public void day2()
    {
        System.out.println("two turtle doves, and ");
        day1();
    }

    public void day3()
    {
        System.out.println("three french hens,");
        day2();
```

```
}

public void day4()
{
    System.out.println("four calling birds,");
    day3();
}

public void day5()
{
    System.out.println("five gold rings,");
    day4();
}

public void day6()
{
    System.out.println("six geese a laying,");
    day5();
}

public void day7()
{
    System.out.println("seven swans a swimming,");
    day6();
}

public void day8()
{
    System.out.println("eight maids a milking,");
    day7();
}

public void day9()
{
    System.out.println("nine ladies dancing,");
    day8();
}

public void day10()
{
    System.out.println("ten pipers piping,");
    day9();
}

public void day11()
{
    System.out.println("eleven lords a leaping,");
    day10();
}
```

```
    public void day12()
    {
        System.out.println("twelve pizzas baking,");
        day11();
    }

    public static void main(String argv[])
    {
        TwelveDays song = new TwelveDays();
        song.intro("first");
        song.day1();
        song.intro("second");
        song.day2();
        song.intro("third");
        song.day3();
        song.intro("fourth");
        song.day4();
        song.intro("fifth");
        song.day5();
        song.intro("sixth");
        song.day6();
        song.intro("seventh");
        song.day7();
        song.intro("eighth");
        song.day8();
        song.intro("ninth");
        song.day9();
        song.intro("tenth");
        song.day10();
        song.intro("eleventh");
        song.day11();
        song.intro("twelfth");
        song.day12();
    } // main()
}// TwelveDays
```

15. Define a `void` method named `verse` that takes two `String` parameters and returns a verse of the Christmas carol, "Twelve Days of Christmas." For example, if you call this method with `verse("first", "a partridge in a pear tree")`, it should return, "On the first day of Christmas my true love gave to me, a partridge in a pear tree."

    Answer:

```
/*
 * File: TwelveDaysLine
 * Author: Java, Java, Java
 * Description: This class tests the verse() method.
 */
```

```
public class TwelveDaysLine
{
    /**
     * verse() returns a verse of the Twelve Days carol
     * @return -- a String giving the verse
     */
    public String verse(String day, String gift)
    {
        return "On the " + day +
          " day of christmas my true love gave to me " + gift;
    }

    /**
     * main() creates an instance and tests the verse
     */
    public static void main (String argv[])
    {
        TwelveDaysLine line1 = new TwelveDaysLine();
        System.out.println(line1.verse( "first",
                              "a partridge in a pear tree"));
    }
} // TwelveDaysLine
```

16. Define a `void` method named `permute`, which takes three `String` parameters and prints out all possible arrangements of the three strings. For example, if you called `permute("a","b","c")`, it would produce the following output: abc, acb, bac, bca, cab, cba, with each permutation on a separate line.

    Answer:

```
/*
 * File: Permute.java
 * Author: Java, Java, Java
 * Description: This class contains a permute() method
 *  which prints all possible permutations of its three
 *  parameters.
 */

public class Permute
{
    /**
     * permute() prints the permutations of a, b, and c,
     *  its three String parameters
     */
    public void permute(String a, String b, String c)
    {
        System.out.println(a + b + c);
        System.out.println(a + c + b);
        System.out.println(b + a + c);
        System.out.println(b + c + a);
```

```
        System.out.println(c + a + b);
        System.out.println(c + b + a);
    } //permute

    /**
     * main() creates an instance and tests the
     * permute() method
     */
    public static void main(String strv[])
    {
        Permute abc = new Permute();
        abc.permute("a","b","c");
    } // main()
} // Permute
```

17. Design a method that can produce limericks given a bunch of rhyming words. For example, if you call

```
limerick("Jones","stones","rained","pained","bones");
```

your method might print (something better than)

```
There once a person named Jones
Who had a great liking for stones,
But whenever it rained,
Jones' expression was pained,
Because stones weren't good for the bones.
```

Answer:

```
/*
 * File: Limerick.java
 * Author: Java, Java, Java
 * Description: This class "sings" a limerick. The
 *  limerick's content can be varied by providing
 *  different keywords to the limerick() method.
 */

public class Limerick
{
    /**
     * sing() prints the limerick, substituting its
     * parameters as the key rhyming words
     * @param name -- a String giving a person's name
     * @param noun -- a String giving an object that
     *   the person likes
     * @param verb -- a String giving a name for a
     *   weather condition
```

```
     * @param adjective -- a String giving a name for
     *   the person's mood
     * @param noun2 -- a String giving a name for the
     *   person's body part
     */
    public void sing(String name, String noun, String verb,
                           String adjective, String noun2)
    {
        System.out.println("There once was a person named " +
                                                    name + ",");
        System.out.println("Who had a great liking for " +
                                                    noun + ",");
        System.out.println("But whenever it " + verb + ",");
        System.out.println(name + "' expression was " +
                                                adjective + ",");
        System.out.println("Because " + noun +
                    " weren't good for the " + noun2 + ",");
    } // limerick()

    /**
     * main() creates a Limerick instance and tests it on a
     *  couple of examples.
     */
    public static void main(String argv[])
    {
        Limerick limerick = new Limerick();
        limerick.sing("Jones","stones","rained","pained","bones");
        limerick.sing("Green","trees","hailed","paled","knees");
    } // main()
} // Limerick
```

For each of the following exercises, write a complete Java application program.

18. Define a class named `Donor` that takes has two instance variables, the donor's name and rating, both `String`s. The name can be any string, but the rating should be one of the following values: "high," "medium," or "none." Write the following methods for this class: a constructor, `Donor(String,String)`, that allows you to set both the donor's and rating; and access methods to set and get both the name and rating of a donor.

    Answer:

```
/*
 * File: Donor.java
 * Author: Java, Java, Java
 * Description: This class represents a donor to an
 *  organization. It stores the donor's name and rating.
 *  The main() method tests the class's methods.
 */
```

```
public class Donor
{
    private String name = "no name";
    private String rating = "none";

    /**
     * Donor() constructor sets the object's name and rating
     * @param str -- a String giving the donor's name
     * @param str2 -- a String giving the donor's rating
     */
    public Donor(String str, String str2)
    {
        name = str;
        rating = str2;
    }

    /**
     * getName() returns the donor's name
     * @return a String giving the person's name
     */
    public String getName()
    {
        return name;
    }

    /**
     * getRating() returns the donor's rating
     * @return a String giving the person's rating
     */
    public String getRating()
    {
        if (rating.equals ("high"))
            return "high";
        if (rating.equals ("medium"))
            return "medium";
        else
            return "none";
    }

    /**
     * main() creates three Donor instances and tests this
     *  classes methods.
     */
    public static void main (String argv[])
    {
        Donor donor1 = new Donor("NewDonor", "high");
        System.out.println("Donor's name is " +
                                        donor1.getName());
        System.out.println(donor1.getName() +
                    "'s rating is " + donor1.getRating());
```

```
        Donor donor2 = new Donor("NewDonor2", "medium");
        System.out.println("Donor's name is " +
                                        donor2.getName());
        System.out.println(donor2.getName() +
                    "'s rating is " + donor2.getRating());

        Donor donor3 = new Donor("NewDonor3", "unknown");
        System.out.println("Donor's name is " +
                                       donor3.getName());
        System.out.println(donor3.getName() +
                   "'s rating is " + donor3.getRating());
    } // main()
} // Donor
```

19. **Challenge.** Define a `CopyMonitor` class that solves the following problem. A company needs a monitor program to keep track of when a particular copy machine needs service. The device has two important (boolean) variables: its toner level (too low or not) and whether it has printed more than 100,000 pages since its last servicing (it either has or has not). The servicing rule that the company uses is that service is needed when either 100,000 pages have been printed or the toner is too low. Your program should contain a method that reports either "service needed" or "service not needed" based on the machine's state. (Pretend that the machine has other methods that keep track of toner level and page count.)

    Answer

```
/*
 * File: CopyMoniter.java
 * Author: Java, Java, Java
 * Description: This class simulates a copy machine monitor.
 *  It has boolean variables to keep track of whether its
 *  toner is too low or whether it has exceeded its page
 *  limit. Its serviceNeeded() method reports whether or
 *  not service is needed.
 */

public class CopyMoniter
{
    private boolean tonerTooLow = false;
    private boolean printed100000Pages = false;

    /**
     * setVariables() sets the monitors boolean instance
     *   variables
     * @param tonerLow -- true if the toner is too low
     * @param pageCountHigh -- true if the page count is
     *   too high
     */
```

```
    public void setVariables(boolean tonerLow,
                                boolean pageCountHigh)
    {
        tonerTooLow = tonerLow;
        printed100000Pages = pageCountHigh;
    }

    /**
     * serviceNeeded() reports whether the machine requires
     * servicing based on a check of its toner and page count
     * @return a String reporting the service status
     */
    public String serviceNeeded()
    {
        if (tonerTooLow)
            return "Service Needed";
        if (printed100000Pages)
            return "Service Needed";
        else
            return "Service not needed";
    }

    public static void main(String argv[])
    {
        CopyMoniter cm = new CopyMoniter();
        cm.setVariables(true, true);
        System.out.println(cm.serviceNeeded());
        cm.setVariables(true, false);
        System.out.println(cm.serviceNeeded());
        cm.setVariables(false, true);
        System.out.println(cm.serviceNeeded());
        cm.setVariables(false, false);
        System.out.println(cm.serviceNeeded());
    } // main()
} // CopyMoniter
```

20. **Challenge.** Design and write an `OldMacdonald` class that sings several verses of "Old MacDonald Had a Farm." Use methods to generalize the verses. For example, write a method named `eieio()` to "sing" the "E I E I O" part of the verse. Write another method with the signature `hadAnX(String s)`, which sings the "had a duck" part of the verse, and a method `withA(String sound)` to sing the "with a quack quack here" part of the verse. Test your class by writing a `main()` method.

    Answer:

```
/*
 * File: OldMacdonald.java
 * Author: Java, Java, Java
 * Description: This version of OldMacdonald uses
```

```
 * parameters in the singVerse() method to generalize
 * the song. By substituting arguments for the parameters
 * when singVerse() is called, singVerse() can be used to
 * sing an unlimited number of different verses.
 */

public class OldMacdonald
{
    /**
     * singVerse() sings a verse of Old Macdonald
     * @param animal -- a String giving the name of an
     *   animal
     * @param sound -- a String giving a sound made by
     *   the animal
     */
    public void singVerse(String animal, String sound)
    {
        System.out.println("Old Macdonald had a farm");
        System.out.println("E I E I O");
        System.out.println("And on his farm he had a " +
                                                 animal);
        System.out.println("E I E I O");
        System.out.println("With a " + sound + " " +
                                      sound + " here," );
        System.out.println("And a " + sound + " " +
                                    sound +  " there," );
        System.out.println("Here a " + sound + ", there a " +
          sound + ", everywhere a " + sound + " " + sound);
        System.out.println("Old Macdonald had a farm");
        System.out.println("E I E I O");
    }

    /**
     * main() creates an instance of this class and calls
     * the singVerse() method with different arguments to
     * produce two verses of Old MacDonald
     */
    public static void main(String argv[])
    {
        OldMacdonald mac = new OldMacdonald();
        mac.singVerse("duck", "quack");
        mac.singVerse("cow", "moo");
    } // main()
} // OldMacdonald
```

Additional Exercises

21. Suppose you have an `Object A`, with public methods `a()`, `b()`, and private method `c()`. And suppose you have a subclass of `A` named `B` with methods

named `b()`, `c()` and `d()`. Draw a UML diagram showing the relationship between these two classes. Explain the inheritance relationships between them and identify those methods which would be considered polymorphic.

Answer: The method `b()` is polymorphic. It will have different behavior for an object of class A and an object of class B. Class B inherits methods `a()` and `b()` because they are public. The private method `A.c()` is not inherited by B. A UML diagram for classes A and B appears in Figure-3.2.

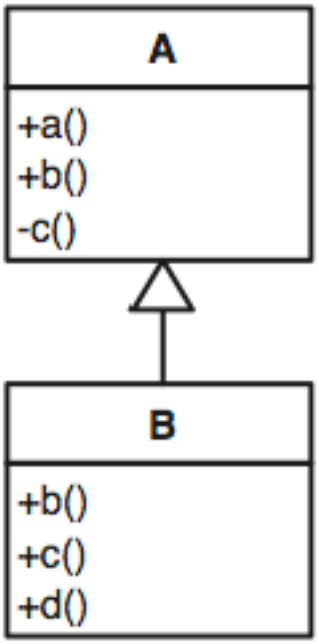


Figure 3.2: UML diagram for the classes A and B.

22. Consider the definition of the class `C`. Define a subclass of `C` named `B` that overrides method `m1()` so that it returns the difference between `m` and `n` instead of their sum.

```
public class C {
    private int m;
    private int n;
    public C(int mIn, int nIn) {
        m = mIn;
        n = nIn;
    }
    public int m1() {
        return m+n;
```

```
    }
}
```

Answer:

```
public class B extends C {
    public C(int mIn, int nIn) {
        m = mIn;
        n = nIn;
    }
    public int m1() {
        return m-n;
    }
}
```

Chapter 4

Input/Output: Designing the User Interface

1. Fill in the blanks in each of the following sentences:

   (a) An _____ is a Java program that can be embedded in a Web page. **Answer: applet**

   (b) A method that lacks a body is an _____ method. **abstract**

   (c) An _____ is like a class except that it contains only instance methods, no instance variables. **Answer: interface**

   (d) In a Java class definition a class can _____ a class and _____ an interface. **Answer: extend, implement**

   (e) Classes and methods not defined in a program must be _____ from the Java class library. **Answer: inherited**

   (f) A subclass of a class inherits the class's _____ instance variables and instance methods. **Answer: public and protected**

   (g) An object can refer to itself by using the _____ keyword. **Answer: this**

   (h) The `JButton`, `JTextField` and `JComponent` classes are defined in the _____ package. **Answer: javax.swing**

   (i) Java applets utilize a form of control known as _____ programming. **Answer: event driven**

   (j) When the user clicks on an applet's `JButton`, an _____ will automatically be generated. **Answer: ActionEvent**

   (k) Two kinds of objects that generate `ActionEvents` _____ and _____. **Answer: JButton, JTextField**

   (l) `JButtons`, `JTextFields`, and `JLabels` are all subclasses of _____. **Answer: JComponent**

   (m) The `JApplet` class is a subclass of _____. **Answer: Applet**

(n) If an applet intends to handle `ActionEvent`s, it must implement the _____ interface. **Answer: ActionListener**

(o) When an applet is started, its _____ method is called automatically. **Answer: init()**

2. Explain the difference between the following pairs of concepts:

(a) *Class* and *interface*.

Answer: An *interface* contains only instance methods and constants. A *class* also contains instance variables.

(b) *Extending a class* and *instantiating an object*.

Answer: In *extending a class* you are defining a new class. When *instantiating an object* you are creating a new object.

(c) *Defining a method* and *implementing a method*.

Answer: This is subtle. *Defining a method* means writing the method's header. *Implementing a method* actually involves writing the code that makes up the method's body. The `actionPerformed()` method is defined in the `ActionListener` interface and implemented in classes that implement that interface.

(d) A `protected` method and a `public` method.

Answer: A `protected` method is inherited by subclasses but cannot be accessed by other objects. A `public` method is inherited by subclasses and can be accessed by other objects.

(e) A `protected` method and a `private` method.

Answer: A `protected` method is inherited by subclasses but a `private` method is not inherited.

(f) An `ActionEvent` and an `ActionListener` method.

Answer: An `ActionEvent` is an object that represents the event that occurs when a user clicks on a button . An `ActionListener` is an interface that defines the (`actionPerformed()`), which handles `ActionEvent`s.

3. Draw a hierarchy chart to represent the following situation. There are lots of languages in the world. English, French, Chinese and Korean are examples of natural languages. Java, C, and C++ are examples of formal languages. French and Italian are considered Romance languages, while Greek and Latin are considered classical languages.

Answer: A hierarchy chart for the languages is displayed in Figure 4.1.

4. Arrange the awt and swing subclasses of the `Component` class used in this chapter into their proper hierarchy, using the `Object` class as the root of the hierarchy.

Answer: A hierarchy chart for part of the Java Swing hierarchy is displayed in Figure 4.2

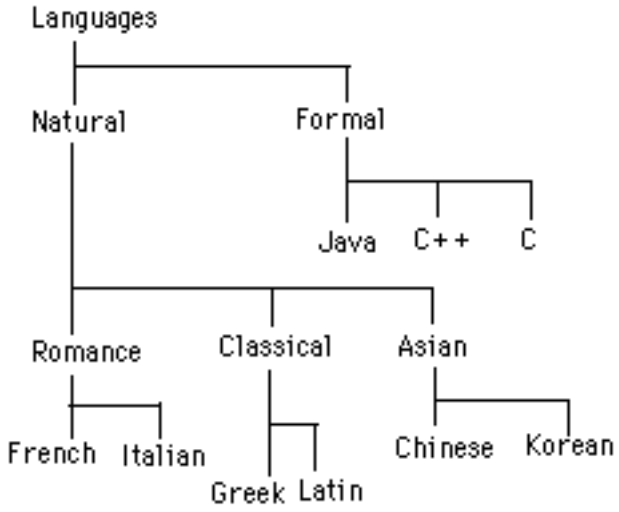


Figure 4.1: The language hierarchy described in Exercise 4.3.

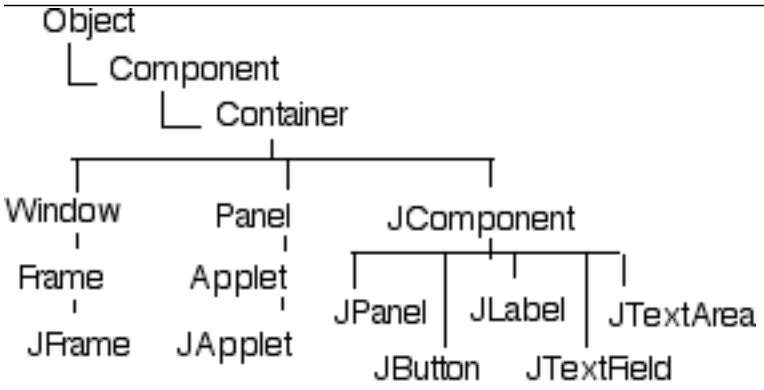


Figure 4.2: Part of the Java Swing hierarchy described in Exercise 4.4.

5. Look up the documentation for the `JButton` class on Sun's Web site:

   ```
   http://java.sun.com/j2se/1.5.0/docs/api
   ```

   List the signatures of all its constructors.

   Answer:

   ```
   JButton( )
   JButton(Action a)
   JButton(Icon icon)
   JButton(String text)
   JButton(String text, Icon icon)
   ```

6. Suppose we want to set the text in our applet's `JTextField`. What method should we use and where is this method defined? (*Hint*: Look up the documentation for `JTextField`. If no appropriate method is defined there, see if it is inherited from a superclass.)

   Answer: The `setText()` method should be used. It is defined in the `JTextComponent` class.

7. Does an `JApplet` have an `init()` method? Explain.

   Answer: Yes. The `init()` method is inherited from the `Applet` class where it is defined as a stub method which can be overridden.

8. Does a `JApplet` have an `add()` method? Explain.

   Answer: Yes. An `Applet` is a `Container`, so it inherits the `add()` method from the `Container` class. However the method cannot be used to add components to a `JApplet` object. Components must be added using the `add()` method of the content pane of the `JApplet` object.

9. Does a `JButton` have an `init()` method? Explain.

   Answer: Yes, a `JButton` has an `init()` method which it inherits from the `AbstractButton` class.

10. Does a `JButton` have an `add()` method? Explain.

    Answer: Yes. The a `JButton` has an `add()` method which it inherits from the `Container` class.

11. Suppose you type the URL for a "Hello World" applet into your browser. Describe what happens — that is, describe the processing that takes place in order for the applet to display "Hello World" in your browser.

    Answer: First the browser creates an instance of the applet and begins its execution by calling the `init()` method, which prints "Hello World." The `stop()` method is then invoked to stop the applet.

12. Suppose you have an applet containing a `JButton` named `button`. Describe what happens, in terms of Java's event handling model, when the user clicks on the button.

    Answer: First an `ActionEvent` object is created for the event. It will record the event that took place and the important details about that event. This event is then passed to the applet's `ActionPerformed()` method, if the applet is designated as the `ActionListener` for that particular type of event. The `actionPerformed()` method then performs the action that the button was meant to perform.

13. Java's `Object` class contains a public method, `toString()`, which returns a string that represents this object. Because every class is a subclass of `Object`, the `toString()` method can be used by any object. Show how you would invoke this method for a `JButton` object named `button`.

    Answer:

    ```
    button.toString()
    ```

14. The following applet contains a semantic error in its `init()` method. The error will cause the `actionPerformed()` method never to display "Clicked" even though the user clicks on the button in the applet. Why? (*Hint*: Think scope!)

    ```
    public class SomeApplet extends JApplet
    {
                              // Declare instance variables
        private JButton button;

        public void init()
        {                     // Instantiate the instance variable
            JButton button = new JButton("Click me");
            add(button);
            button.addActionListener(this);
        } // init()

        public void actionPerformed(ActionEvent e)
        {
            if (e.target == button)
               System.out.println("Clicked");
        } // actionPerformed()
    } // SomeApplet
    ```

    Answer: The `JButton` declared in the `init()` and assigned an `ActionListener` there, has *local scope*. So it cannot be referred to in the `actionPerformed()` method. The statement

    ```
    JButton button = new JButton("Click me");
    ```

should be replaced with the statement

```
button = new JButton("Click me");
```

There is a second error related to using the `add()` method of the `JApplet` class. This method will not add a component like `JButton` to the applet. One must use the `add()` method of the `JApplet`'s content pane with a `BorderLayout`. Thus the statement

```
add(button);
```

should be replaced with the statement

```
getContentPane().add("Center", button);
```

15. What would be output by the following applet(with typos corrected)?

```
public class SomeApplet extends JApplet
{
                              // Declare instance variables
    private JButton button;
    private JTextField field;

    public void init()
    {
                              // Instantiate instance variables
        button = new JButton("Click me");
        getContentPane().add("North", button);  // Typo fixed
        field = new JTextField("Field me");
        getContentPane().add("South", field);   // Typo fixed
        System.out.println(field.getText() + button.getText());
    } // init()
} // SomeApplet
```

Answer: A line of output saying "Field meClick me" would also be sent to `System.out`.

16. Write a simple Java applet that has a `JButton`, a `JTextField`, and a `JLabel` and then uses the `toString()` method to display each object's string representation.

Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `ToStringPanel` and `ToStringFrame` classes create a GUI application solution to the exercise. The `ToStringPanel` and `ToStringApplet` classes with the `index.html` HTML code create an applet solution to the exercise. The strings returned by the `toString` methods are very long lists of the objects' instance variables and their values.

```
/*
 * File: ToStringPanel.java
 * Author: Java Java Java  - Part of a solution to exercise 4.16.
 * Description: This class defines a GUI in a JPanel which contains
 * a JButton, JLabel, and JTextField. The output of the toString()
 * method for each of these three objects are displayed in a
 * JTextArea and also written to System.out.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ToStringPanel extends JPanel implements ActionListener
{   private JTextArea display;
    private JTextField aField;
    private JButton aButton;
    private JLabel aLabel;

    public ToStringPanel()
    {
        buildGUI();
    } // ToStringPanel()

    private void buildGUI()
    {   display = new JTextArea(10,30);
        aField = new JTextField(10);
        aButton = new JButton("Click me");
        aButton.addActionListener(this);
        aLabel = new JLabel("Press button to see output");
        add(aField);
        add(aLabel);
        add(aButton);
        add(display);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {   if (e.getSource() == aButton)
        {
            display.append(aButton.toString() + "\n");
            display.append(aLabel.toString() + "\n");
            display.append(aField.toString() + "\n");
            System.out.println(aButton.toString());
            System.out.println(aLabel.toString());
            System.out.println(aField.toString());
        } // if
    } // actionPeformed()
} // ToStringPanel class

/*
 * File: ToStringFrame.java
```

```
 * Author: Java Java Java - Part of the solution of Exercise 4.16.
 * Description: This program creates a ToStringPanel and
 *  adds it to the Frame's content pane and sets its size.
 */
import javax.swing.*;

public class ToStringFrame extends JFrame
{
    public ToStringFrame()
    {
        getContentPane().add(new ToStringPanel());
    } // ToStringFrame() constructor

    public static void main(String args[]){
        ToStringFrame gframe = new ToStringFrame();
        gframe.setSize(600,400);
        gframe.setVisible(true);
    } // main()

} // ToStringFrame class

/*
 * File: ToStringApplet.java
 * Author: Java Java Java - Part of solution to Exercise 4.16.
 * Description: This applet creates a ToStringPanel and
 *  adds it to the applet's content pane.
 */
import javax.swing.*;

public class ToStringApplet extends JApplet
{   public void init()
    {    getContentPane().add(new ToStringPanel());
    }
} // ToStringApplet class

<!-- ***********************************************
file: index.html - Part of the solution for Exercise 4.16
************************************************
-->

<html>
<head><title>ToStringApplet</title></head>
<body>
<hr>
<applet code=ToStringApplet.class width=600 height=400>
</applet>
<hr>
<a href="ToStringApplet.java">ToStringApplet.java source code</a>
</body>
</html>
```

17. The `JButton` class inherits a `setText(String)` from its `AbstractButton` superclass. Using that method, design and implement a GUI that has a single button labeled initially "The Doctor is out". Each time the button is clicked, it should toggle its label to "The Doctor is in" and vice versa.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `DoctorPanel` and `DoctorFrame` classes create a GUI application solution to the exercise. The `DoctorPanel` and `DoctorApplet` classes with the `index.html` HTML code create an applet solution to the exercise. The algorithm for exchanging the `JButton` labels is only one of several algorithms that accomplish this task.

```
/*
 * File: DoctorPanel.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 4.17.
 * Description: This class defines a GUI in a JPanel
 * which contains a JButton with initial label "The
 * Doctor is in".  Pressing the button toggles the
 * label between "The Doctor is out" and "The Doctor
 * is out".
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DoctorPanel extends JPanel
                         implements ActionListener
{
    private JButton docButton;
    String presentText = "The Doctor is out";
    String nextText = "The Doctor is in";
    String tempText; // To use to exchange labels

    public DoctorPanel()
    {
        buildGUI();
    } // DoctorPanel()

    private void buildGUI()
    {   docButton = new JButton("The Doctor is out");
        docButton.addActionListener(this);
        add(docButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {   if (e.getSource() == docButton)
        {
            tempText = presentText;
            presentText = nextText;
```

```
            nextText = tempText;
            docButton.setText(presentText);
        } // if
    } // actionPeformed()
} // DoctorPanel class

/*
 * File: DoctorFrame.java
 * Author: Java Java Java - Part of the solution of
 * Exercise 4.17
 * Description: This program creates a DoctorPanel
 * and adds it to the Frame's content pane and sets
 * its size.
 */
import javax.swing.*;

public class DoctorFrame extends JFrame
{
    public DoctorFrame()
    {
        getContentPane().add(new DoctorPanel());
    } // DoctorFrame() constructor

    public static void main(String args[]){
        DoctorFrame dframe = new DoctorFrame();
        dframe.setSize(600,400);
        dframe.setVisible(true);
    } // main()

} // DoctorFrame class

/*
 * File: DoctorApplet.java
 * Author: Java Java Java - Part of solution to
 * Exercise 4.17.
 * Description: This applet creates a DoctorPanel
 * and adds it to the applet's content pane.
 */
import javax.swing.*;

public class DoctorApplet extends JApplet
{   public void init()
    {    getContentPane().add(new DoctorPanel());
    }
} // DoctorApplet class

<!-- ***********************************************
file: index.html - Part of the solution for
Exercise 4.17
************************************************
```

```
-->

<html>
<head><title>DoctorApplet</title></head>
<body>
<hr>
<applet code=DoctorApplet.class width=600 height=400>
</applet>
<hr>
<a href="DoctorApplet.java">
                    DoctorApplet.java source code</a>
</body>
</html>
```

18. Design and implement a GUI that contains two `JButton`s, initially labeled "Me first!" and "Me next!" Each time the user clicks either button, the labels on both buttons are exchanged. (*Hint*: You don't need an if-else statement for this problem.)

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `MeFirstPanel` and `MeFirstFrame` classes create a GUI application solution to the exercise. The `MeFirstPanel` and `MeFirstApplet` classes with the `index.html` HTML code create an applet solution to the exercise. The algorithm for exchanging the `JButton` labels is only one of several algorithms that accomplish this task.

```
/*
 * File: MeFirstPanel.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 4.18.
 * Description: This class defines a GUI in a JPanel
 * which contains two JButton with initial labels "Me first!"
 * and "Me next!". Pressing either button causes the labels
 * to be exchanged.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MeFirstPanel extends JPanel
                                  implements ActionListener
{
    private JButton aButton;
    private JButton bButton;
    String aText = "Me first!";
    String bText = "Me next!";
    String tempText; // To use to exchange labels

    public MeFirstPanel()
```

```
    {
        buildGUI();
    } // MeFirstPanel()

    private void buildGUI()
    {
        aButton = new JButton(aText);
        aButton.addActionListener(this);
        bButton = new JButton(bText);
        bButton.addActionListener(this);
        add(aButton);
        add(bButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
            tempText = aText;  // Exchange the strings
            aText = bText;
            bText = tempText;
            aButton.setText(aText); // Set button labels
            bButton.setText(bText);
    } // actionPeformed()
} // MeFirstPanel class

/*
 * File: MeFirstFrame.java
 * Author: Java Java Java - Part of the solution of
 * Exercise 4.18
 * Description: This program creates a MeFirstPanel
 * and adds it to the Frame's content pane and sets
 * its size.
 */
import javax.swing.*;

public class MeFirstFrame extends JFrame
{
    public MeFirstFrame()
    {
        getContentPane().add(new MeFirstPanel());
    } // MeFirstFrame() constructor

    public static void main(String args[]){
        MeFirstFrame aframe = new MeFirstFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()
} // MeFirstFrame class

/*
 * File: MeFirstApplet.java
```

```
 * Author: Java Java Java - Part of solution to
 * Exercise 4.18.
 * Description: This applet creates a MeFirstPanel
 * and adds it to the applet's content pane.
 */
import javax.swing.*;

public class MeFirstApplet extends JApplet
{   public void init()
    {    getContentPane().add(new MeFirstPanel());
    }
} // MeFirstApplet class

<!-- ***********************************************
file: index.html - Part of the solution for
Exercise 4.18
************************************************
-->

<html>
<head><title>MeFirstApplet</title></head>
<body>
<hr>
<applet code=MeFirstApplet.class width=600 height=400>
</applet>
<hr>
<a href="MeFirstApplet.java">
                    MeFirstApplet.java source code</a>
</body>
</html>
```

19. Modify the GUI in the previous exercise so that it contains three `JButton`s, initially labeled "First," "Second," and "Third." Each time the user clicks on one of the buttons, the labels on the buttons are rotated. Second should get First's label, Third should get Second's, and First should get Third's label.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `ThreeButtonsPanel` and `ThreeButtonsFrame` classes create a GUI application solution to the exercise. The `ThreeButtonsPanel` and `ThreeButtonsApplet` classes with the `index.html` HTML code create an applet solution to the exercise. The algorithm for rotating the `JButton` labels is only one of several algorithms that accomplish this task.

```
/*
 * File: ThreeButtonsPanel.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 4.19.
 * Description: This class defines a GUI in a JPanel
 * which contains three JButtons with initial labels
```

```
 * "First", "Second" and "Third". Pressing any button
 * causes the labels to be rotated.  The label on the
 * first button goes to the second, and so on.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ThreeButtonsPanel extends JPanel
                            implements ActionListener
{
    private JButton aButton;
    private JButton bButton;
    private JButton cButton;
    String aText = "First";
    String bText = "Second";
    String cText = "Third";
    String tempText; // To use to rotate labels

    public ThreeButtonsPanel()
    {
        buildGUI();
    } // ThreeButtonsPanel()

    private void buildGUI()
    {
        aButton = new JButton(aText);
        aButton.addActionListener(this);
        bButton = new JButton(bText);
        bButton.addActionListener(this);
        cButton = new JButton(cText);
        cButton.addActionListener(this);
        add(aButton);
        add(bButton);
        add(cButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
            tempText = bText; // Rotate the strings
            bText = aText;
            aText = cText;
            cText = tempText;
              // Set button labels
            aButton.setText(aText);
            bButton.setText(bText);
            cButton.setText(cText);
    } // actionPeformed()
} // ThreeButtonsPanel class
```

```
/*
 * File: ThreeButtonsFrame.java
 * Author: Java Java Java - Part of the solution of
 * Exercise 4.19
 * Description: This program creates a ThreeButtonsPanel
 * and adds it to the Frame's content pane and sets
 * its size.
 */
import javax.swing.*;

public class ThreeButtonsFrame extends JFrame
{
    public ThreeButtonsFrame()
    {
        getContentPane().add(new ThreeButtonsPanel());
    } // ThreeButtonsFrame() constructor

    public static void main(String args[]){
        ThreeButtonsFrame aframe =
                              new ThreeButtonsFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()

} // ThreeButtonsFrame class

/*
 * File: ThreeButtonsApplet.java
 * Author: Java Java Java - Part of solution to
 * Exercise 4.19.
 * Description: This applet creates a ThreeButtonsPanel
 *  and adds it to the applet's content pane.
 */
import javax.swing.*;

public class ThreeButtonsApplet extends JApplet
{   public void init()
    {    getContentPane().add(new ThreeButtonsPanel());
    }
} // ThreeButtonsApplet class

<!-- ***********************************************
file: index.html - Part of the solution for
Exercise 4.19
***********************************************
-->

<html>
<head><title>ThreeButtonsApplet</title></head>
<body>
```

```
<hr>
<applet code=ThreeButtonsApplet.class width=600 height=400>
</applet>
<hr>
<a href="ThreeButtonsApplet.java">
                    ThreeButtonsApplet.java source code</a>
</body>
</html>
```

20. Design and implement a GUI contains a `JTextField` and two `JButton`s, initially labeled "Left" and "Right." Each time the user clicks a button, display its label in the `JTextField`. A `JButton`'s label can be gotten with the `getText()` method.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `LeftRightPanel` and `LeftRightFrame` classes create a GUI application solution to the exercise. The `LeftRightPanel` and `LeftRightApplet` classes with the `index.html` HTML code create an applet solution to the exercise.

```
/*
 * File: LeftRightPanel.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 4.20.
 * Description: This class defines a GUI in a JPanel
 * which contains two JButtons with initial labels
 * "Left"and  "Right" and a JTextField. Pressing a
 * button causes the label of that button to be
 * printed into the textfield.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class LeftRightPanel extends JPanel
                          implements ActionListener
{
    private JButton leftButton;
    private JButton rightButton;
    private JTextField outField;

    public LeftRightPanel()
    {
        buildGUI();
    } // LeftRightPanel()

    private void buildGUI()
    {
        leftButton = new JButton("Left");
```

```
        leftButton.addActionListener(this);
        rightButton = new JButton("Right");
        rightButton.addActionListener(this);
        outField = new JTextField(10);
        add(leftButton);
        add(outField);
        add(rightButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == leftButton)
            outField.setText(leftButton.getText());
        else
            outField.setText(rightButton.getText());
    } // actionPeformed()
} // LeftRightPanel class

/*
 * File: LeftRightFrame.java
 * Author: Java Java Java - Part of the solution of
 * Exercise 4.20
 * Description: This program creates a LeftRightPanel
 * and adds it to the Frame's content pane and sets
 * its size.
 */
import javax.swing.*;

public class LeftRightFrame extends JFrame
{
    public LeftRightFrame()
    {
        getContentPane().add(new LeftRightPanel());
    } // LeftRightFrame() constructor

    public static void main(String args[]){
        LeftRightFrame aframe = new LeftRightFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()

} // LeftRightFrame class

/*
 * File: LeftRightApplet.java
 * Author: Java Java Java - Part of solution to
 * Exercise 4.20.
 * Description: This applet creates a LeftRightPanel
 * and adds it to the applet's content pane.
 */
```

```
import javax.swing.*;

public class LeftRightApplet extends JApplet
{   public void init()
    {    getContentPane().add(new LeftRightPanel());
    }
} // LeftRightApplet class

<!-- ***********************************************
file: index.html - Part of the solution for
Exercise 4.20
***********************************************
-->

<html>
<head><title>LeftRightApplet</title></head>
<body>
<hr>
<applet code=LeftRightApplet.class width=600 height=400>
</applet>
<hr>
<a href="LeftRightApplet.java">
                     LeftRightApplet.java source code</a>
<a href="LeftRightPanel.java">
                      LeftRightPanel.java source code</a>
</body>
</html>
```

21. You can change the size of a `JFrame` by using the `setSize(int h, int v)` method, where *h* and *v* give the horizontal and vertical dimensions of the applet's window in pixels. Write a GUI application that contains two `JButton`s, labeled "Big" and "Small." Whenever the user clicks on Small, set the applets dimensions to 200 x 100, and whenever the user clicks on Big, set the dimensions to 300 x 200.

    Answer: The code below for `BigSmallFrame` is modeled after the `GreeterGUI` JFrame class in the chapter.

```
/*
 * File: BigSmallFrame.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 4.21.
 * Description: This class defines a GUI in a JFrame
 * which contains two JButtons with initial labels "Big"
 * and  "Small". Pressing the Big button sizes the
 * {\tt JFrame} to 300X200. Pressing the Small button
 * sizes the {\tt JFrame} to 200X100.
 */
import javax.swing.*;
import java.awt.*;
```

```
import java.awt.event.*;

public class BigSmallFrame extends JFrame
                              implements ActionListener
{
    private JButton bigButton;
    private JButton smallButton;
    private Container cPane;

    public BigSmallFrame(String title)
    {
        buildGUI();
        setTitle(title);
        setSize(300,200);
        pack();
        setVisible(true);
    } // BigSmallFrame()

    private void buildGUI()
    {
        cPane = getContentPane();
        cPane.setLayout(new BorderLayout());
        bigButton = new JButton("Big");
        bigButton.addActionListener(this);
        smallButton = new JButton("Small");
        smallButton.addActionListener(this);
        cPane.add("West", bigButton);
        cPane.add("East", smallButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == bigButton)
           setSize(300,200);
        else
           setSize(200,100);
    } // actionPeformed()

    public static void main(String args[]){
        BigSmallFrame theFrame =
                       new BigSmallFrame("BigSmallGUI");
        theFrame.setSize(300,200);
    } // main()
} // BigSmallFrame class
```

22. Rewrite your solution to the previous exercise so that it uses a single button, whose label is toggled appropriately each time it is clicked. Obviously, when the `JButton` is labeled "Big," clicking it should give the applet its big dimensions.

    Answer: The code below for `BigSmallFrame2` is modeled after the `GreeterGUI`

JFrame class in the chapter.

```
/*
 * File: BigSmallFrame2.java
 * Author: Java Java Java  - A solution to exercise 4.22.
 * Description: This class defines a GUI in a JFrame
 * which contains a JButton with initial label "Big".
 * Pressing the button sizes the {\tt JFrame} to 300X200
 * and changes the button label to "Small". Pressing the
 * button when it is labeled "Small"  sizes the {\tt JFrame}
 * to 200X100 and changes the button label to "Big".
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class BigSmallFrame2 extends JFrame
                                  implements ActionListener
{
    private JButton aButton;
    private Container cPane;
    private String presentLabel = "Big";
    private String nextLabel = "Small";
    private String tempString; // For exchanging labels.

    public BigSmallFrame2(String title)
    {
        buildGUI();
        setTitle(title);
        pack();
        setVisible(true);
    } // BigSmallFrame2()

    private void buildGUI()
    {
        cPane = getContentPane();
        cPane.setLayout(new BorderLayout());
        aButton = new JButton("Big");
        aButton.addActionListener(this);
        cPane.add("North", aButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
        if (presentLabel == "Big")
           setSize(300,200);
        else
           setSize(200,100);
        aButton.setText(nextLabel);
        tempString = nextLabel;
```

```
        nextLabel = presentLabel;
        presentLabel = tempString;
    } // actionPeformed()

    public static void main(String args[]){
        BigSmallFrame2 theFrame =
                  new BigSmallFrame2("BigSmallGUI2");
        theFrame.setSize(200,100);
    } // main()
} // BigSmallFrame2 class
```

23. **Challenge:** Design and write a Java GUI that allows the user to change the applet's background color to one of three choices, indicated by buttons. Like all other Java `Components`, `JFrames` have an associated background color, which can be set by the following commands:

```
setBackground( Color.red );
setBackground( Color.yellow );
```

The `setBackground()` method is defined in the `Component` class, and 13 primary colors — `black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow` — are defined in the `java.awt.Color` class.

Answer: The code below for `ColorFrame` is modeled after the `GreeterGUI` JFrame class in the chapter.

```
/*
 * File: ColorFrame.java
 * Author: Java Java Java  - A solution to exercise 4.23.
 * Description: This class defines a GUI in a JFrame
 * which contains 3 JButtons with labels "Blue", "Yellow",
 * and "Red". Pressing a changes the background of the
 * {\tt JFrame} to the indicated color
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ColorFrame extends JFrame
                                implements ActionListener
{
    private JButton blueButton;
    private JButton yellowButton;
    private JButton redButton;
    private Container cPane;

    public ColorFrame(String title)
    {
```

```
        buildGUI();
        setTitle(title);
        pack();
        setVisible(true);
    } // ColorFrame()

    private void buildGUI()
    {
        cPane = getContentPane();
        cPane.setLayout(new BorderLayout());
        blueButton = new JButton("Blue");
        blueButton.addActionListener(this);
        cPane.add("West", blueButton);
        yellowButton = new JButton("Yellow");
        yellowButton.addActionListener(this);
        cPane.add("North", yellowButton);
        redButton = new JButton("Red");
        redButton.addActionListener(this);
        cPane.add("East", redButton);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == blueButton)
           setBackground(Color.blue);
        else if (e.getSource() == yellowButton)
           setBackground(Color.yellow);
        else
           setBackground(Color.red);
    } // actionPeformed()

    public static void main(String args[]){
        ColorFrame theFrame =
                           new ColorFrame("Color GUI");
        theFrame.setSize(400,200);
    } // main()
} // ColorFrame class
```

Additional Exercises

24. Given the classes with the following headers

```
public class Animal ...
public class DomesticAnimal extends Animal ...
public class FarmAnimal extends DomesticAnimal...
public class HousePet extends DomesticAnimal...
public class Cow extends FarmAnimal ...
public class Goat extends FarmAnimal ...
```

```
public class DairyCow extends Cow ...
```

draw a UML class diagram representing the hierarchy created by these declarations.

Answer: The UML diagram is shown in Figure 4.3.

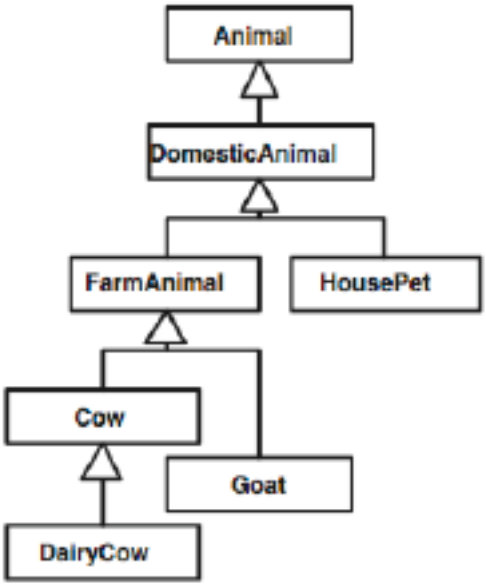


Figure 4.3: The animal hierarchy described in Exercise 4.24.

25. Given the preceding hierarchy of classes, which of the following are legal assignment statements?

```
DairyCow dc = new FarmAnimal();
FarmAnimal fa = new Goat();
Cow c1 = new DomesticAnimal();
Cow c2 = new DairyCow();
DomesticAnimal dom = new HousePet();
```

Answer:

```
DairyCow dc = new FarmAnimal();       // Error
FarmAnimal fa = new Goat();           // Ok
Cow c1 = new DomesticAnimal();        // Error
Cow c2 = new DairyCow();              // Ok
DomesticAnimal dom = new HousePet();  // Ok
```

Chapter 5

Java Data and Operators

1. Explain the difference between the following pairs of terms.

   (a) *Representation* and *action*.

   Answer: *Representation* is finding a way to look at a problem. *Action* is the process of taking well defined steps to solve a problem

   (b) *Binary operator* and *unary operation*.

   Answer: A *binary* operator takes two operands whereas a *unary* operator takes only one operand. For example, the NOT operator (!o1) is a unary operator and takes only one operand but the AND operator (o1 && 02) takes two operands.

   (c) *Class constant* and *class variable*.

   Answer: A *class variable* is a variable that is declared static and is associated with the class itself, not with its instances. A class variable that is also declared final is a *class constant* and cannot be changed by the program.

   (d) *Helper method* and *class method*.

   Answer: A *helper method* is a method that is declared private and therefore used only within the class itself. A *class method* is static method that is associated with the class itself, not with its instances.

   (e) *Operator overloading* and *method overloading*.

   Answer: *Operator overloading* is using the same operator for more then one type-dependent operation, such using as the + operator for integer and real addition. *Method overloading* is using a method with the same name but different signatures to perform different tasks.

   (f) *Method call* and *method composition*.

   Answer: *Method call* is when an object calls a method to perform a particular task. *Method composition* is the nesting of two or more method calls, so that the result of one method call is passed to a second method.

   (g) *Type conversion* and *type promotion*.

Answer: *Type conversion* is changing a datum from one type to another. *Type promotion* is when the compiler automatically converts a variable from one type to another to prevent a potential loss of information.

2. For each of the following data types, list how many bits are used in its representation and how many values can be represented:

```
(a)  int     (b) char     (c) byte      (d) long     (e) double
```

Answer: (a) 32, 2^{32} (b) 16, 2^{16} (c) 8, 2^8 (d) 64, 2^{64} (e) 64, 2^{64}

3. Fill in the blank.

   (a) Methods and variables that are associated with a class rather than with its instances must be declared _____. **Answer: static**

   (b) When an expression involves values of two different types, one value must be _____ before the expression can be evaluated. **Answer: promoted**

   (c) Constants should be declared _____. **Answer: final**

   (d) Variables that take `true` and `false` as their possible values are known as _____. **Answer: boolean**

4. Arrange the following data types into a *promotion* hierarchy: `double`, `float`, `int`, `short`, `long`.

   Answer:

```
short  -->  int --> long -->  float -->  double
```

5. Assuming that `o1` is true, `o2` is false, and `o3` is false, evaluate each of the following expressions:

```
(a) o1 || o2 && o3        (b) o1 ^ o2            (c) !o1 && !o2
```

Answer: (a) true (b) true (c) false

6. Arrange the following operators in precedence order:

```
+ - () * / % < ==
```

Answer:

```
HIGHEST ()
        *, /, %
        +, -
        <
LOWEST  ==
```

7. Arrange the following operators into a precedence hierarchy:

```
  *,++, %, ==

{\bf Answer:}
  HIGHEST ++
          *, %
  LOWEST  ==
```

8. Parenthesize and evaluate each of the following expressions. If an expression is invalid, mark it as such.

```
(a) 11 / 3 % 2 == 1          (b) 11 / 2 % 2 > 0        (c) 15 % 3 >= 21 % 3
(d) 12.0 / 4.0 >= 12 / 3     (e) 15 / 3 == true
```

Answer:

```
(a) ((11 / 3) % 2) == 1      ==> true
(b) ((11 / 2) % 2) > 0       ==> true
(c) (15 % 3) >= (21 % 3)     ==> true
(d) (12.0 / 4.0) >= (12 / 3) ==> false
(e) (15 / 3) == true         ==> invalid
```

9. What value would `m` have after each of the following statements is executed? Assume that `m, k, j` are reinitialized before each statement.

```
   int m = 5, k = 0, j = 1;
(a) m = ++k + j;          (b) m += ++k * j;          (c) m %= ++k + ++j;
(d) m = m - k - j;        (e) m = ++m;
```

Answer: (a) 2 (b) 6 (c) 2 (d) 4 (e) 6

10. What value would `b` have after each of the following statements is executed? Assume that `m, k, j` are reinitialized before each statement. It may help to parenthesize the right-hand side of the statements before evaluating them.

```
   boolean b;
   int m = 5, k = 0, j = 1;
(a) b = m > k + j;               (b) b = m * m != m * j;
(c) b = m <= 5 && m % 2 == 1;    (d) b = m < k  || k < j;
(e) b = --m == 2 * ++j;
```

Answer: (a) true (b) true (c) true (d) true (e) true

11. For each of the following expressions, if it is valid, determine the value of the variable on the left-hand side. If not, change it to a valid expression.

```
    char c = 'a' ;
    int  m = 95;
(a) c = c + 5;          (b) c = 'A' + 'B';  (c) m = c + 5;
(d) c = (char) m + 1;   (e) m = 'a' - 32;

{\bf Answer:}
(a) c = (char)(c + 5)   (b) c = (char)('A' + 'B') (c) m == 102
(d) c = (char)(m + 1)   (e) m = 65
```

12. Translate each of the following expressions into Java.

```
(a) Area equals pi times the radius squared.
(b) Area is assigned pi times the radius squared.
(c) Volume is assigned pi times radius cubed divide by h.
(d) If m and n are equal then m is incremented by one,
    otherwise n is.
(e) If m is greater than n times 5 then square m and double
    n otherwise square n and double m.
```

Answer:

```
(a) A == 3.14 * pow(r,2)
(b) A = 3.14 * pow(r,2)
(c) V = ( 3.14 * pow(r,3) ) / h
(d) if (m == n)
        ++m;
    else
        ++n;
(e) if (m > (n * 5)) {
        m = pow(m,2);
        n *= 2;
    } else {
        n = pow(n,2);
        m *= 2;
    }
```

13. What would be output by the following code segment?

```
    int m = 0, n = 0, j = 0, k = 0;
    m = 2 * n++;
    System.out.println("m= " + m + " n= " + n);
    j += ( --k * 2 );
    System.out.println("j= " + j + " k= " + k);
```

Answer:

```
    Output:
    m= 0 n= 1
    j= -2 k= -1
```

Each of the following problems that follows asks you to write a method. Test the method as you develop it in a stepwise fashion. Here's a simple application program that you can use for this purpose.

```
public class MethodTester {
     public static int square(int n) {
          return n * n;
     }

     public static void main(String args[]) {
          System.out.println("5 squared = " + square(5));
     }
}
```

Replace the `square()` method with your method. Note that you must declare your method `static` if you want to call it directly from `main()` as we do here.

14. Write a method to calculate the sales tax for a sale item. The method should take two `double` parameters, one for the sales price and the other for the tax rate. It should return a `double`. For example, `calcTax(20.0, 0.05)` should return 1.0.

    Answer:

```
public static double calcTax(double price, double rate) {
    return price * rate;
}
```

15. **Challenge:** Suppose you're writing a program that tells what day of the week someone's birthday falls on this year. Write a method that takes an `int` parameter, representing what day of the year it is, and returns a `String` like "Monday." For example, for 2004, a leap year, the first day of the year was on Thursday. The thirty-second day of the year (February 1, 2004) was a Sunday, so `getDayOfWeek(1)` should return "Thursday" and `getDayOfWeek(32)` should return "Sunday." (*Hint*: If you divide the day of the year by 7, the remainder will always be a number between 0 and 6, which can be made to correspond to days of the week.)

    Answer:

```
public static String getDayOfWeek(int dayOfYear) {
    int num = dayOfYear % 7;
    if (num == 4)
        return "Sunday";
    else if (num == 5)
        return "Monday";
    else if (num == 6)
        return "Tuesday";
    else if (num == 0)
```

```
        return "Wednesday";
    else if (num == 1)
        return "Thursday";
    else if (num == 2)
        return "Friday";
    else if (num == 3)
        return "Saturday";
    else
        return "Error";
} // getDayOfWeek()
```

16. **Challenge:** As part of the same program you'll want a method that takes the month and the day as parameters, and returns what day of the year it is. For example, `getDay(1,1)` should return 1; `getDay(2,1)` should return 32; and `getDay(12,31)` should return 366. (*Hint*: If the month is 3, and the day is 5, you have to add the number of days in January plus the number of days in February to 5 to get the result: 31 + 29 + 5 = 65.)

    Answer:

```
public static int getDay(int month, int day) {
    if (month == 1)
        return day;
    else if (month == 2)
        return 31 + day;
    else if (month == 3) // In 2004 February has 29 days
        return 31 + 29 + day;
    else if (month == 4)
        return 31 + 29 + 31 + day;
    else if (month == 5)
        return 31 + 29 + 31 + 30 + day;
    else if (month == 6)
        return 31 + 29 + 31 + 30 + 31 + day;
    else if (month == 7)
        return 31 + 29 + 31 + 30 + 31 + 30 + day;
    else if (month == 8)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + day;
    else if (month == 9)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                                    day;
    else if (month == 10)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                                 30 + day;
    else if (month == 11)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                              30 + 31 + day;
    else if (month == 12)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                           30 + 31 + 30 + day;
    else
```

```
        return 0;
} // getDay()
```

17. Write a Java method that converts a `char` to lowercase. For example, `toLowerCase('A')` should return 'a'. Make sure you guard against method calls like `toLowerCase('a')`.

    Answer:

```
public static char toLowerCase(char ch) {
    if (ch >= 'A' && ch <= 'Z')
        return (char)(ch + 32);
    return ch;
}
```

18. **Challenge:** Write a Java method that shifts a `char` by *n* places in the alphabet, wrapping around to the start of the alphabet, if necessary. For example, `shift('a',2)` should return 'c'; `shift('y',2)` should return 'a'. This method can be used to create a Caesar cipher, in which every letter in a message is shifted by `n` places — hfu ju? (Refer to the Chapter 1 exercises for a refresher on the Caesar cipher.)

    Answer

```
public static char shift(char ch, int num) {
    if (ch >= 'a' && ch <= 'z')
    {
        if (((char)(ch + num) >= 'a') &&
                            ((char)(ch + num) <= 'z'))
            return (char)(ch + num);
        else if ((char)(ch + num) >= 'z')
            return (char)((ch + num) - 26);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
        if (((char)(ch + num) >= 'A') &&
                            ((char)(ch + num) <= 'Z'))
            return (char)(ch + num);
        else if ((char)(ch + num) >= 'Z')
            return (char)((ch + num) - 26);
    }
    return ch;
} // shift()
```

19. Write a method that converts its `boolean` parameter to a `String`. For example, `boolToString( true )` should return "true."

    Answer

```
public static String boolToString(boolean bool) {
    if (bool)
        return "true";
```

```
        else
            return "false";
    }
```

20. Write a Java application that first prompts the user for three numbers that represent the sides of a rectangular cube, and then computes and outputs the volume and the surface area of the cube.

    Answer:

```
/*
 * File: Cube.java
 * Author: Java, Java, Java - A solution to exercise 5.20
 * Description: This class represents a geometric cube
 * in terms of the lengths of its 3 sides. It contains
 * methods to calculate the cubes volume and surface area.
 * The main() method creates an instance of the class and
 * lets the user input the lengths of the sides and prints
 * out the volume and surface area of the cube.
 */

import java.io.*;

public class Cube
{
    private int side1, side2, side3; // This cube's sides

    /**
     * Cube() constructor creates an instance given the
     * lengths of its 3 sides.
     * @param s1, s2 and s3 are ints representing the
     *   lengths of the sides
     */
    public Cube(int s1, int s2, int s3)
    {
        side1 = s1;
        side2 = s2;
        side3 = s3;
    }

    /**
     * calcVolume() returns the volume of this cube
     * using the formula V = s^3
     */
    public int calcVolume() {
        return side1 * side2 * side3;
    }

    /**
     * calcSurfaceArea() returns the cube's surface area
```

```
     * using the formula A = 2ab + 2bc + 2ac, where a, b,
     * c are the sides.
     */
    public int calcSurfaceArea() {
        return 2 * (side1 * side2) + 2 * (side2 * side3) +
                                      2 * (side1 * side3);
    }

    /**
     *  main() creates an instance of this class using
     *  values input by the user. It then tests the
     *  methods.
     */
    public static void main(String args[])
                                    throws IOException{
        KeyboardReader kb = new KeyboardReader();
        System.out.println("Enter Length of First Side:");
        int side1 = kb.getKeyboardInteger();

        System.out.println("Enter Length of Second Side:");
        int side2 = kb.getKeyboardInteger();

        System.out.println("Enter Length of Third Side:");
        int side3 = kb.getKeyboardInteger();

        Cube cube = new Cube(side1, side2, side3);
        System.out.println("The Volume is " +
                                      cube.calcVolume());
        System.out.println("The Surface Area is " +
                                  cube.calcSurfaceArea());
    } // main()
}
```

21. Write a Java application that prompts the user for three numbers, and then outputs the three numbers in increasing order.

    Answer:

```
/*
 * File: IncreasingOrder.java
 * Author: Java, Java, Java
 * Description: This application prompts the user
 * for three integers and the outputs the integers
 * in increasing order. Its main() method creates
 * an instance of the class in order to test that
 * its methods work correctly.
 *
 * A java.io.BufferedReader object is used to
 * input the data.
 */
```

```
import java.io.*;

public class IncreasingOrder
{
    /**
     * smallest() returns the smallest of its three
     * parameters. Boolean operators are used to
     * perform the comparisons.
     * @param num1, num2, num3 are three integers
     * @return a int representing the smallest value
     *   is returned
     */
    public int smallest(int num1, int num2, int num3) {
        if (num1 <= num2 && num1 <= num3)
            return num1;
        else if (num2 <= num1 && num2 <= num3)
            return num2;
        else if (num3 <= num1 && num3 <= num2)
            return num3;
        return 0;
    } // smallest()

    /**
     * middle() returns the middle value of its three
     * parameters. Boolean operators are used to
     *   perform the comparisons.
     * @param num1, num2, num3 are three integers
     * @return a int representing the middle value
     *   is returned
     */
    public int middle(int num1, int num2, int num3) {
        if (num1 <= num3 && num1 >= num2)
            return num1;
        else if (num1 >= num3 && num1 <= num2)
            return num1;
        else if (num2 <= num1 && num2 >= num3)
            return num2;
        else if (num2 >= num1 && num2 <= num3)
            return num2;
        else if (num3 <= num1 && num3 >= num2)
            return num3;
        else if (num3 >= num1 && num3 <= num2)
            return num3;
        return 0;
    } // middle()

    /**
     * largest() returns the largest of its three
     * parameters. Boolean operators are used to
```

```
     * perform the comparisons.
     * @param num1, num2, num3 are three integers
     * @return a int representing the largest value
     *   is returned
     */
    public int largest(int num1, int num2, int num3) {
        if (num1 >= num2 && num1 >= num3)
            return num1;
        else if (num2 >= num1 && num2 >= num3)
            return num2;
        else if (num3 >= num1 && num3 >= num2)
            return num3;
        return 0;
    } // largest()

    /**
     * main() creates an instance of this class and
     * uses it to test the ordering methods. Note
     * the use of a KeyboardReader to perform keyboard
     * input.
     */
    public static void main(String args[])
                                throws IOException{
        IncreasingOrder orderer = new IncreasingOrder();

        KeyboardReader input = new KeyboardReader();
             // Prompt the user
        System.out.println("Enter First Number:");
             // Read the input
        int num1 = input.getKeyboardInteger();

        System.out.println("Enter Second Number:");
        int num2 = input.getKeyboardInteger();

        System.out.println("Enter Third Number:");
        int num3 = input.getKeyboardInteger();

              // Print results
        System.out.println("Numbers in increasing order are: ");
        System.out.println( orderer.smallest(num1,num2,num3) + ","
                            + orderer.middle(num1,num2,num3) + ","
                            + orderer.largest(num1,num2,num3));
    } // main()
} // IncreasingOrder
```

22. Write a Java application that inputs two integers and then determines whether the first is divisible by the second. (*Hint*: Use the modulus operator.)

    Answer:

```
/*
```

```
 * File: Divisible.java
 * Author: Java, Java, Java - Solution to Exercise 5.22
 * Description: This program inputs two integers from the
 *  user and determines whether the first is divisible by
 *  the second.
 */

import java.io.*;

public class Divisible
{

    /**
     * divisible() returns true iff its first parameter
     *  is divisible by its second
     * @param num1 - integer representing the numerator
     * @param num2 - integer representing the denominator
     */
    public boolean divisible(int num1, int num2) {
        if (num1 % num2 == 0)
            return true;
        else
            return false;
    } // divisible()


    /**
     * main() inputs two integers from the user. It then
     *  creates an instance of Divisible and tests whether
     *  the first integer is divisible by the second.
     */
    public static void main(String args[])
                                      throws IOException{
        KeyboardReader input = new KeyboardReader();
        System.out.print("Enter First Number:");
        int num1 = input.getKeyboardInteger();

        System.out.print("Enter Second Number:");
        int num2 = input.getKeyboardInteger();
            // Create an instance
        Divisible div = new Divisible();

        if (div.divisible(num1,num2))
            System.out.println(num1 +
                       " is evenly divisible by " + num2);
        else
            System.out.println(num1 +
                    " is NOT evenly divisible by " + num2);
    } // main()
} // Divisible
```

23. Write a Java application that prints the following table:

```
N   SQUARE   CUBE
1   1        1
2   4        8
3   9        27
4   16       64
5   25       125
```

Answer:

```
/*
 * File: Table.java
 * Author: Java, Java, Java
 * Description: This application prints a table of the
 *  squares and cubes of the numbers from 1 to 5.
 */

import java.lang.*;

public class Table
{
    /**
     * square() returns the square of its parameter
     * @param num -- an integer to be squared
     * @return the integer square
     */
    public int square(int num) {
        return (int)Math.pow(num,2);
    }

    /**
     * cube() returns the cube of its parameter
     * @param num -- an integer to be cubed
     * @return the integer cube
     */
    public int cube(int num) {
        return (int)Math.pow(num,3);
    }

    /**
     * main() creates an instance of this class and
     * invokes its square() and cube() methods to
     * print a table of squares and cubes.
     */
    public static void main(String args[])  {
        Table table = new Table();
        System.out.println("N  SQUARE  CUBE");
        System.out.println(1 + "    " +
                 table.square(1) + " " + table.cube(1));
```

```
        System.out.println(2 + "    " +
           table.square(2) + "     " + table.cube(2));
        System.out.println(3 + "    " +
           table.square(3) + "     " + table.cube(3));
        System.out.println(4 + "    " +
            table.square(4) + "    " + table.cube(4));
        System.out.println(5 + "    " +
            table.square(5) + "    " + table.cube(5));
    } // main()
} // Table
```

24. Design and write a Java GUI that converts kilometers to miles and vice versa. Use a `JTextField` for I/O and `JButton`s for the various conversion actions.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `ConvertDistance`, `ConvertDistancePanel` and `ConvertDistanceFrame` classes create a GUI application solution to the exercise. The `ConvertDistance`, `ConvertDistancePanel`, and `ConvertDistanceApplet` classes with the `index.html` HTML code create an applet solution to the exercise.

```
/*
 * File: ConvertDistance.java
 * Author: Java, Java, Java
 * Description: This class contains methods for
 * converting miles to kilometers and vice versa.
 * It is modeled after that java.lang.Math class in
 * that its conversion methods are declared static.
 * This means that they are associated with the class
 * itself, rather than with its instances. So to use
 * one of the methods, you can just refer to it as
 * ConvertDistance.convertToMi().
 */

public class ConvertDistance
{
    /**
     * convertToMi() converts its parameters to
     * distance in miles
     * @param km -- a double giving the distance
     *   in kilometers
     * @return -- a double giving the distance
     *   in miles
     */
    public static double convertToMi(double km)
    {
        return km * (6.2 / 10.0);
    } // convertToMi()

    /**
     * convertToKm() converts its parameters to
```

```
     * distance in kilometers
     * @param km -- a double giving the distance
     *   in miles
     * @return -- a double giving the distance
     *   in kilometers
     */
    public static double convertToKm(double mi)
    {
        return mi * (10.0 / 6.2);
    } // convertToKm()
} // ConvertDistance

/*
 * File: ConvertDistancePanel.java
 * Author: Java Java Java
 * Description: This class defines a GUI in a
 * JPanel which containstwo JButtons with initial
 * labels "Left"and  "Right" and a JTextField.
 * Pressing a button causes the label of that
 * button to be printed into the textfield.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ConvertDistancePanel extends JPanel
                        implements ActionListener
{
    private JLabel prompt;     // GUI components
    private JTextField inputField;
    private JTextField outputField;
    private JButton button1;
    private JButton button2;
       // Stores distance in miles
    private double distanceInMiles;
       // Stores distance in km
    private double distanceInKm;
       // The user's input
    private String inputString;
       // Keeps track of the units (km or miles)
    private boolean distEnteredInKm;

    public ConvertDistancePanel()
    {
        buildGUI();
    } // ConvertDistancePanel()

    private void buildGUI()
    {
        String str =
```

```
      "Input a floating point distance (mi. or km.):";
    prompt = new JLabel(str);
    inputField = new JTextField(10);
    inputField.setEditable(true);
    outputField = new JTextField(30);
    outputField.setEditable(false);
    button1 = new JButton("Convert To Miles");
    button1.addActionListener(this);
    button2 = new JButton("Convert To Kilometers");
    button2.addActionListener(this);

    add( prompt);
    add(inputField);
    add(button1);
    add(button2);
   add(outputField);
} //buildGUI()

public void actionPerformed(ActionEvent e)
{
            inputString = inputField.getText();
    double distance = Double.parseDouble(inputString);

    if (e.getSource() == button1)
    {
        distanceInMiles =
               ConvertDistance.convertToMi(distance);
        distanceInKm =
             Double.parseDouble(inputField.getText());
        distEnteredInKm = true;
    }
    else if (e.getSource() == button2)
    {
        distanceInKm =
               ConvertDistance.convertToKm(distance);
        distanceInMiles =
            Double.parseDouble(inputField.getText());
        distEnteredInKm = false;
    }

    String outStr;
    if (distEnteredInKm)
      outStr = distanceInKm + "Km = " +
                             distanceInMiles + "Mi";
    else
      outStr = distanceInMiles + "Mi = " +
                                distanceInKm + "Km";

    outputField.setText(outStr);
} // actionPerformed()
```

```
} // ConvertDistancePanel class

/*
 * File: ConvertDistanceFrame.java
 * Author: Java Java Java
 * Description: This program creates a ConvertDistancePanel,
 * adds it to the Frame's content pane, and sets its size.
 */
import javax.swing.*;

public class ConvertDistanceFrame extends JFrame
{
    public ConvertDistanceFrame()
    {
        getContentPane().add(new ConvertDistancePanel());
    } // ConvertDistanceFrame() constructor

    public static void main(String args[]){
        ConvertDistanceFrame aframe =
                            new ConvertDistanceFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()

} // ConvertDistanceFrame class

/*
 * File: ConvertDistanceApplet.java
 * Author: Java Java Java
 * Description: This applet creates a ConvertDistancePanel
 * and adds it to the applet's content pane.
 */
import javax.swing.*;

public class ConvertDistanceApplet extends JApplet
{   public void init()
    {    getContentPane().add(new ConvertDistancePanel());
    }
} // ConvertDistanceApplet class

<!-- ***********************************************
file: index.html - Part of the solution for Exercise 5.24
************************************************
-->

<html>
<head><title>ConvertDistanceApplet</title></head>
<body>
<hr>
```

```
<applet code=ConvertDistanceApplet.class width=600 height=400>
</applet>
<hr>
<a href="ConvertDistanceApplet.java">
                    ConvertDistanceApplet.java source code</a>
<a href="ConvertDistancePanel.java">
                     ConvertDistancePanel.java source code</a>
</body>
</html>
```

25. Design and write a GUI that allows a user to calculate the maturity value of a CD. The user should enter the principal, interest rate, and years, and the GUI should then display the maturity value. Make use of the `BankCD` class covered in this chapter. Use separate `JTextField`s for the user's inputs and a separate `JTextField` for the result.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `BankCD`, `CDInterestPanel` and `CDInterestFrame` classes create a GUI application solution to the exercise. The `BankCD`, `CDInterestPanel`, and `CDInterestApplet` classes with the `index.html` HTML code create an applet solution to the exercise.

```
/*
 * File: CDInterestPanel.java
 * Author: Java Java Java
 * Description: This class defines a GUI in a JPanel
 * which contains JTextFields for entering principal,
 * interest rate, and years and a JButton which when
 * clicked will compute the maturity value of a CD
 * with those values and will display this value in
 * a JTextField.  It uses the BankCD class.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.text.NumberFormat;

public class CDInterestPanel extends JPanel
                             implements ActionListener
{
    private JLabel prompt1; // Prompt for the principal.
    private JLabel prompt2; // Prompt for the interest rate.
    private JLabel prompt3; // Prompt for the number of years.
    private JTextField inputField1; // For principal.
    private JTextField inputField2; // For interest rate.
    private JTextField inputField3; // For number of years
    private JLabel resultLabel;  // Labels the output
    private JTextField resultField; // The output
    private JButton button; //Click to compute maturity value.
```

```
public CDInterestPanel()
{
    buildGUI();
} // CDInterestPanel()

private void buildGUI()
{
    prompt1 =
      new JLabel("Enter the CD's initial principal:");
    prompt2 =
      new JLabel("Enter the CD's interest rate:");
    prompt3 =
      new JLabel("Enter the number of years to maturity:");
    resultLabel = new JLabel("The Maturity Value is:");
    inputField1 = new JTextField(10);
    inputField1.setEditable(true);
    inputField2 = new JTextField(10);
    inputField2.setEditable(true);
    inputField3 = new JTextField(10);
    inputField3.setEditable(true);
    resultField = new JTextField(10);
    button = new JButton("Calculate the Maturity Value");
    button.addActionListener(this);

    add(prompt1);
    add(inputField1);
    add(prompt2);
    add(inputField2);
    add(prompt3);
    add(inputField3);
    add(button);
    add(resultLabel);
    add(resultField);
} //buildGUI()


/**
 * actionPerformed() handles clicks on the calculate
 * button. It inputs the data from the JTextFields,
 * converts them to numeric values and calls on the
 * BankCD object to perform the calculation. It
 * translates the result into a currency format and
 * displays it in a JTextField.
 */
public void actionPerformed(ActionEvent e)
{
    String inputString = inputField1.getText();
    double principal = Double.parseDouble(inputString);
    inputString = inputField2.getText();
    double rate = Double.parseDouble(inputString);
```

```
        inputString = inputField3.getText();
        double years = Double.parseDouble(inputString);
        BankCD cd = new BankCD(principal, rate, years);
        double maturityValue = cd.calcYearly();

        NumberFormat dollars =
                    NumberFormat.getCurrencyInstance();
        String resultStr = dollars.format(maturityValue);
        resultField.setText(resultStr);
    } // actionPeformed()
} // CDInterestPanel class

/*
 * File: CDInterestFrame.java
 * Author: Java Java Java
 * Description: This program creates a CDInterestPanel
 * and adds it to the Frame's content pane and sets its
 * size.
 */
import javax.swing.*;

public class CDInterestFrame extends JFrame
{
    public CDInterestFrame()
    {
        getContentPane().add(new CDInterestPanel());
    } // CDInterestFrame() constructor

    public static void main(String args[]){
        CDInterestFrame aframe = new CDInterestFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()
} // CDInterestFrame class

/*
 * File: CDInterestApplet.java
 * Author: Java Java Java
 * Description: This applet creates a CDInterestPanel
 * and adds it to the applet's content pane.
 */
import javax.swing.*;

public class CDInterestApplet extends JApplet
{   public void init()
    {    getContentPane().add(new CDInterestPanel());
    }
} // CDInterestApplet class

<!-- ********************************************
```

```
file: index.html - Part of a solution for Exercise 5.25
************************************************
-->

<html>
<head><title>CD Interest Applet</title></head>
<body>
<hr>
<applet code=CDInterestApplet.class width=400 height=400>
</applet>
<hr>
<a href="CDInterstApplet.java">CDInterestApplet.java</a>
<br>
<a href="CDInterstPanel.java">CDInterestPanel.java</a>
<br>
<a href="CDInterest.java">CDInterest.java</a>
</body>
</html>

/*
 * File: CDInterest.java
 * Author: Java, Java, Java
 * Description: This class calculates the maturity
 * value of a CD given its principal, interest rate
 * and time period in years. It contains a single
 * method to perform this task.
 */

  public class BankCD
  {     // Purchase price of the CD.
    private double principal;
        // Yearly interest rate.
    private double rate;
        // The period until maturity of CD.
    private double years;

    /**
     * BankCD constructor assigns values to instance
     * variables.
     * @param p - a double representing initial principal
     * @param r - a double representing interest rate
     * @param y -- a double representing number of years.
     */
    public BankCD(double p, double r, double y)
    {
        principal = p;
        rate = r;
        years = y;
    }
```

```
    /**
     * calcYearly() calculates the maturity value
     * of a CD given its principal, yearly interest
     * rate, and maturity period. It uses the formula
     * a = p * (1 + r)^y
     * @param p - a double representing initial principal
     * @param r - a double representing interest rate
     * @param y - a double representing number of years
     * @return - a double in currency format
     */
    public double calcYearly()
    {
        return (principal * Math.pow(1 + rate, years));
    } // calcYearly()

    /**
     * calcDaily() calculates the maturity value of a
     * CD given its principal, yearly interest rate, and
     * maturity period assuming daily compound interest.
     * It uses the formula a = p * (1 + r/365)^(365*y)
     * @param p - a double representing initial principal
     * @param r -- a double representing interest rate
     * @param y -- a double representing number of years
     * @return -- a doublein currency format is returned
     */
    public double calcDailyly()
    {
     return (principal * Math.pow(1 + rate/365, years*365));
    } // calcDaily()

} // BankCD class
```

26. Design and write a GUI that lets the user input a birth date (month and day), and reports what day of the week it falls on. Use the `getDayOfWeek()` and `getDay()` methods that you developed in previous exercises.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `DayOfWeek`, `DayOfWeekPanel` and `DayOfWeekFrame` classes create a GUI application solution to the exercise. The `DayOfWeek`, `DayOfWeekPanel`, and `DayOfWeekApplet` classes with the `index.html` HTML code create an applet solution to the exercise.

```
/*
 * File: DayOfWeekPanel.java
 * Author: Java Java Java
 * Description: This class defines a GUI in a JPanel
 * which calculates what day of the week a given date
 * (mon/day/2004) falls on. The interface contains of
 * three TextFields -- two for inputting the month and
 * day, and one for outputting the result -- and a button,
```

```
 * which directs the program to perform its calculations.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class DayOfWeekPanel extends JPanel
                            implements ActionListener
{
    private JLabel prompt1;     // Prompt for month.
    private JLabel prompt2;     // Prompt for day.
    private JTextField inputField1;  // For month
    private JTextField inputField2;  // For day.
    private JLabel resultLabel;
    private JTextField resultField; // Displays result.
    private JButton button;  // Press to compute.

    public DayOfWeekPanel()
    {
        buildGUI();
    } // DayOfWeekPanel()

    private void buildGUI()
    {
        prompt1 =
          new JLabel("Enter the 2004 Month (1 - 12):");
        prompt2 = new JLabel("Enter the 2004 Day:");
        resultLabel =
          new JLabel("In 2004, Date Falls on a:");
        inputField1 = new JTextField(10);
        inputField2 = new JTextField(10);
        resultField = new JTextField(10);
          // Suppress input in the result field
        resultField.setEditable(false);
        button =
          new JButton("Calculate the Day of The Week");
        button.addActionListener(this);

        add(prompt1);
        add(inputField1);
        add(prompt2);
        add(inputField2);
        add(button);
        add(resultLabel);
        add(resultField);
    } //buildGUI()

    /**
     * actionPerformed() handles clicks on the calculate
     * button. It inputs the data from the JTextFields,
```

```
     * converts them to numeric values and calls on
     * the DayOfWeek object to perform the calculation
     * and displays it in a JTextField.
     */
    public void actionPerformed(ActionEvent e)
    {
        int month =
               Integer.parseInt(inputField1.getText());
        int day =
               Integer.parseInt(inputField2.getText());
        int dayOfYear = DayOfWeek.getDay(month, day);
        String dayStr =
                    DayOfWeek.getDayOfWeek(dayOfYear);
        resultField.setText(dayStr);
    } // actionPeformed()

} // DayOfWeekPanel class

/*
 * File: DayOfWeekFrame.java
 * Author: Java Java Java
 * Description: This program creates a DayOfWeekPanel and
 *  adds it to the Frame's content pane and sets its size.
 */
import javax.swing.*;

public class DayOfWeekFrame extends JFrame
{
    public DayOfWeekFrame()
    {
        getContentPane().add(new DayOfWeekPanel());
    } // DayOfWeekFrame() constructor

    public static void main(String args[]){
        DayOfWeekFrame aframe = new DayOfWeekFrame();
        aframe.setSize(600,400);
        aframe.setVisible(true);
    } // main()
} // DayOfWeekFrame class

/*
 * File: DayOfWeekApplet.java
 * Author: Java Java Java
 * Description: This applet creates a DayOfWeekPanel and
 *  adds it to the applet's content pane.
 */
import javax.swing.*;

public class DayOfWeekApplet extends JApplet
{   public void init()
```

```
    {    getContentPane().add(new DayOfWeekPanel());
    }
} // DayOfWeekApplet class

<!-- ***********************************************
file: index.html
***********************************************
-->
<html>
<head><title>Day of Week Applet</title></head>
<body>
<hr>
<applet code=DayOfWeekApplet.class width=400 height=400>
</applet>
<hr>
<a href="DayOfWeek.java">DayOfWeek.java source code</a>
<br>
<a href="DayOfWeekApplet.java">
                  DayOfWeekApplet.java source code</a>
<br>
<a href="DayOfWeekPanel.java">
                  DayOfWeekApplet.java source code</a>
</body>
</html>

/*
 * File: DayOfWeek.java
 * Author: Java, Java, Java
 * Description: This class calculates the day of the week
 *  (Monday, Tuesday, etc.) given the month and day for
 *  2002. It is modeled after the java.lang.Math class in
 *  that its conversion methods are declared static. Thus
 *  to convert a
 */

public class DayOfWeek
{
    /**
     * getDay() converts the month and day for the
     * year 2004 into the ordinal day of the year, where
     * Jan. 1 is day 1 and Dec. 31 is day 366.
     * @param month -- an int representing the month, 1..12
     * @param day -- an int representing the day, 1..31
     * @return -- an int giving the day of the year 2004
     * Algorithm: Treat each of the 12 months as a separate
     * case, adding the appropriate number of days through
     * the last day of the previous month. Then add in the day.
     */
        public static int getDay(int month, int day) {
        if (month == 1)
```

```
        return day;
    else if (month == 2)
        return 31 + day;
    else if (month == 3) // In 2004 February has 29 days
        return 31 + 29 + day;
    else if (month == 4)
        return 31 + 29 + 31 + day;
    else if (month == 5)
        return 31 + 29 + 31 + 30 + day;
    else if (month == 6)
        return 31 + 29 + 31 + 30 + 31 + day;
    else if (month == 7)
        return 31 + 29 + 31 + 30 + 31 + 30 + day;
    else if (month == 8)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + day;
    else if (month == 9)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                                  day;
    else if (month == 10)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                               30 + day;
    else if (month == 11)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                          30 + 31 + day;
    else if (month == 12)
        return 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 +
                                     30 + 31 + 30 + day;
    else
        return 0;
} // getDay()

/**
 * getDayOfWeek() converts the month and day for the
 *  year 2004 into a string giving the day of the week
 * @param month -- an int representing the month, 1..12
 * @param day -- an int representing the day, 1..31
 * @return -- an String giving the day of the week
 *   (e.g., "Sunday")
 * Algorithm: For 2004, 1/1/2004 was a Thursday. So
 * compute the day of the year. Then divide the day of
 * the year by 7, take the remainder, and determine the
 * day of the week by looking at each of the 7 possible
 * cases.
 */
public static String getDayOfWeek(int dayOfYear) {
    int num = dayOfYear % 7;
    if (num == 4)
        return "Sunday";
    else if (num == 5)
        return "Monday";
```

```
        else if (num == 6)
            return "Tuesday";
        else if (num == 0)
            return "Wednesday";
        else if (num == 1)
            return "Thursday";
        else if (num == 2)
            return "Friday";
        else if (num == 3)
            return "Saturday";
        else
            return "Error";
    } // getDayOfWeek()
} // DayOfWeek
```

27. Design and write a GUI that allows users access input their exam grades for a course and computes their average and probable letter grade. The GUI should contain a single `JTextField` for inputting a grade and a single `JTextField` for displaying the average and letter grade. The program should keep track internally of how many grades the student has entered. Each time a new grade is entered, it should display the current average and probable letter grade.

    Answer: The code below is modeled after the `GreeterGUI` classes in the chapter. The `GradeCalculator`, `GradeCalcPanel` and `GradeCalcFrame` classes create a GUI application solution to the exercise. The `GradeCalculator`, `GradeCalcPanel`, and `GradeCalcApplet` classes with the `index.html` HTML code create an applet solution to the exercise.

```
/*
 * File: GradeCalcPanel.java
 * Author: Java Java Java
 * This class provides a user interface to
 * the GradeCalc class, which calculates a student's average
 * and letter grade for grades input into a JTextField.
 * The interface consists of input and output JTextFields and
 * and button to calculate the course average and letter grade.
 * For each grade entered, an updated average and letter grade
 * is calculated.
 *   Note that for this problem its important that an instance
 * of the GradeCalculator class is used. The instance stores the
 * running total over time as part of its internal state. This
 * couldn't be done if static methods were used. So this is a
 * case where we do NOT want to model the calculator class after
 * the java.lang.Math class.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

```
public class GradeCalcPanel extends JPanel
                                    implements ActionListener
{
    private JLabel prompt;         // GUI components
    private JTextField inputField;
    private JLabel resultLabel;
    private JTextField resultField;
    private JButton button;
    private GradeCalculator calculator; // The Calculator object

    public GradeCalcPanel()
    {
        buildGUI();
    } // GradeCalcPanel()

    private void buildGUI()
    {      // Create a calculator instance
        calculator = new GradeCalculator();

        prompt = new JLabel("Enter a grade:");
        resultLabel = new JLabel("Your average is:");
        inputField = new JTextField(10);
        resultField = new JTextField(20);
        resultField.setEditable(false);
        button =
          new JButton("Calculate Your Average and Letter Grade");
        button.addActionListener(this);

        add(prompt);
        add(inputField);
        add(button);
        add(resultLabel);
        add(resultField);
    } //buildGUI()

    /**
     * actionPerformed() handles clicks on the button.
     * It takes the data from the input JTextFields, and
     * sends them to the GradeCalculater class to
     * calculate a running average and  computes the
     * letter grade, which are displayed in TextFields.
     * @param e -- the ActionEvent the generated this
     *   system call
     */
    public void actionPerformed(ActionEvent e)
    {
        String inputString = inputField.getText();
        double grade = Double.parseDouble(inputString);
        String average = "" + calculator.calcAvg(grade);
        String letterGrade = calculator.calcLetterGrade();
```

```
            resultField.setText(average + " " + letterGrade);
        } // actionPeformed()

    } // DayOfWeekPanel class

    /*
     * File: GradeCalcFrame.java
     * Author: Java Java Java
     * Description: This program creates a GradeCalcPanel and
     *  adds it to the Frame's content pane and sets its size.
     */
    import javax.swing.*;

    public class GradeCalcFrame extends JFrame
    {
        public GradeCalcFrame()
        {
            getContentPane().add(new GradeCalcPanel());
        } // GradeCalcFrame() constructor

        public static void main(String args[]){
            GradeCalcFrame aframe = new GradeCalcFrame();
            aframe.setSize(600,400);
            aframe.setVisible(true);
        } // main()

    } // GradeCalcFrame class

    /*
     * File: GradeCalcApplet.java
     * Author: Java Java Java
     * Description: This applet creates a GradeCalcPanel and
     *  adds it to the applet's content pane.
     */
    import javax.swing.*;

    public class GradeCalcApplet extends JApplet
    {   public void init()
        {    getContentPane().add(new GradeCalcPanel());
        }
    } // GradeCalcApplet class

    <!-- ***********************************************
    file: index.html
    ************************************************
    -->
    <html>
    <head><title>Day of Week Applet</title></head>
    <body>
    <hr>
```

```
<applet code=DayOfWeekApplet.class width=400 height=400>
</applet>
<hr>
<a href="DayOfWeek.java">DayOfWeek.java source code</a>
<br>
<a href="DayOfWeekApplet.java">
                  DayOfWeekApplet.java source code</a>
<br>
<a href="DayOfWeekPanel.java">
                  DayOfWeekApplet.java source code</a>
</body>
</html>

/*
 * File: GradeCalculator.java
 * Author: Java, Java, Java
 * Description: Instances of this class are used to
 * calculate a course average and a letter grade. In
 * order to calculate the average and the letter grade,
 * a GradeCalculator must store two essential pieces of
 * data: the number of grades and the sum of the grades.
 * Therefore these are declared as instance variable.
 * Each time calcAverage(grade) is called, a new grade
 * is added to the running total, and the number
 * of grades is incremented.
 */

public class GradeCalculator
{      // GradeCalculator's internal state
    private int gradeCount = 0;
    private double gradeTotal = 0.0;

    /**
     * calcAverage() is given a grade, which is added
     * to the running total. It then increments the
     * grade count and returns the running average.
     */
    public double calcAvg(double grade)
    {
        gradeTotal += grade;
        ++gradeCount;
        return gradeTotal/gradeCount;
    } // calcAvg

    /**
     * calcLetterGrade() returns the letter grade for
     * this object.
     * Algorithm: The course average is first computed
     * from the stored  gradeTotal and gradeCount
     * and then converted into a letter grade.
```

```
     * @return a String representing "A" through "F"
     */
    public String calcLetterGrade ()
    {     // Get the average
        double avg = gradeTotal / gradeCount;
        if (avg >= 90.0)
            return "A";
        else if (avg >= 80.0)
            return "B";
        else if (avg >= 70.0)
            return "C";
        else if (avg >= 60.0)
            return "D";
        else if (avg >= 0)
            return "F";
        return "Error";
    } //calcLetterGrade()
} // GradeCalculator
```

Additional Exercises

28. One of the reviewers of this text has suggested an alternative design for the `Temperature` class (Text Figure 5.5). According to this design, the class would contain an instance variable, say `temperature`, and access methods that operate on it. The access methods would be: `setFahrenheit(double)`, `getFahrenheit():double`, `setCelsius(double)`, and `getCelsius():double`. One way to implement this design is to store the temperature in the Kelvin scale and then convert from and to Kelvin in the access methods. The formula for converting Kelvin to Celsius is:

    ```
    K = C + 273.15
    ```

    Draw a UML class diagram representing this design of the `Temperature` class. Which design is more object oriented, this one or the one used in Text Figure 5-5?

    Answer: The UML diagram is shown in Figure 5.1.

29. Write an implementation of the `Temperature` class using the design described in the previous exercise.

    Answer:

    ```
    /*
     * File: Temperature.java
     * Author: Java, Java, Java
     * Description: This version of the Temperature class stores
     *   the temperature in Kelvin. It has methods to set and get
     *   the temperature in either Fahrenheit or celsius.
    ```

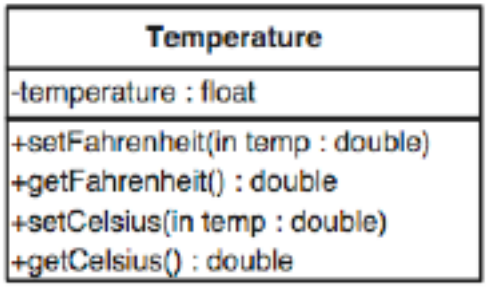


Figure 5.1: The `Temperature` class described in Exercise 5.28.

```
 *
 * One way in which this version is more object-oriented than
 *  the version given is that it has a state, thus more closely
 *  resembling what we commonly think of as an object.
 */

public class Temperature
{
    private double kelvin = 0;  // Temperature in Kelvin

    public Temperature() {}

    /**
     * setCelsius() sets the temperature given a F value
     * @param cels -- gives the temperature in Celsius
     */
    public void setCelsius(double cels)
    {
        kelvin = cels + 273.15;
    }

    /**
     * getCelsius() returns the temperature in Celsius
     * @return -- a double giving the temperature in Celsius
     */
    public double getCelsius()
    {
        return kelvin - 273.15;
    }

    /**
     * setFahrenheit() sets the temperature given a F value
     * @param fahr -- gives the temperature in Fahrenheit
     */
```

```
    public void setFahrenheit(double fahr)
    {
        double celsius = 5.0 * (fahr - 32.0) / 9.0;
        kelvin = celsius + 273.15;
    }

    /**
     * getFahrenheit() returns the temperature in Fahrenheit
     * @return -- a double giving the temperature in Fahrenheit
     */
    public double getFahrenheit()
    {
        double celsius = kelvin -273.5;
        return 9.0 * celsius / 5.0 + 32.0;
    }

} // Temperature
```

Chapter 6

Control Structures

1. Explain the difference between the following pairs of terms.

   (a) *Counting loop* and *conditional loop*.

   Answer: A *counting loop* is used to perform repetitive tasks when the number of iterations required is known beforehand. A *conditional loop* is used to perform repetitive tasks when the number of iterations depends on a non-counting bound.

   (b) *For statement* and *while statement*.

   Answer: A *for statement* is a counting loop that is usually used when the number of iterations is known beforehand. A *while statement* is a conditional loop that is used when the number of iterations depends on a non-counting bound.

   (c) *While statement* and *do-while statement*. **Answer:** A *do-while statement* checks the loop bound after the loop is executed whereas the *while statement* checks it before. This makes it useful to use the while statement for loops that may not be executed at all and to use do-while statements for loops to be executed at least one time.

   (d) *Zero indexing* and *unit indexing*.

   Answer: In *zero indexing* a counter starts at zero. In *unit indexing* a counter starts at one.

   (e) *Sentinel bound* and *limit bound*.

   Answer: A *limit bound* terminates a loop when a certain limit is reached. A *sentinel bound* terminates a loop when a certain special value (the sentinel) is reached.

   (f) *Counting bound* and *flag bound*.

   Answer: A *counting bound* terminates a loop when a certain count is reached. A *flag bound* terminates a loop when a certain boolean variable is set to true.

(g) Loop *initializer* and *updater*.

Answer: A loop *initializer* is a statement that is executed before the first iteration. A loop *updater* is a statement that is executed during each iteration.

(h) *Named constant* and *literal*.

Answer: A *named constant* is a final variable whose value remains the same throughout the program. A *literal* is an actual value or quoted string. Using named constants instead of literals makes the program self documenting and makes it easier to read and revise.

(i) *Compound statement* and *null statement*.

Answer: A *compound statement* is a sequence of simple statements contained within a set of curly brackets. A *null statement* is the absence of a statement – for example, as the body of a loop.

2. Fill in the blank:

(a) The process of reading a data item before entering a loop is known as a _____. **Answer: priming road**

(b) A loop that does nothing except iterate is an example of _____ . **Answer: busy waiting**

(c) A loop that contains no body is an example of a _____ statement. **Answer: null**

(d) A loop whose entry condition is stated as $(k < 100 \mid\mid k >= 0)$ would be an example of an _____ loop. **Answer: compound condition**

(e) A loop that should iterate until the user types in a special value should use a _____ bound. **Answer: sentinel**

(f) A loop that should iterate until its variable goes from 5 to 100 should use a _____ bound. **Answer: counting**

(g) A loop that should iterate until the difference between two values is less than 0.005 is an example of a _____ bound. **Answer: limit**

3. Identify the syntax errors in each of the following:

```
(a) for (int k = 0; k < 100; k++) System.out.println(k)
(b) for (int k = 0; k < 100; k++); System.out.println(k);
(c) int k = 0 while k < 100 {System.out.println(k); k++;}
(d) int k = 0; do {System.out.println(k); k++;} while k < 100;
```

Answer:

```
(a) Error: Missing a semicolon at the end.
(b) Error: Should be no semicolon before System.out.println(k);
      This ends the loop so the last k causes a scope error.
(c) Error: Missing a semicolon after int k = 0 and also it is
      missing parentheses around k < 100.
(d) Error: Missing parentheses around k < 100.
```

4. Determine the output and/or identify the error in each of the following code segments:

```
(a) for (int k = 1; k == 100; k += 2) System.out.println(k);
(b) int k = 0;   while ( k < 100 ) System.out.println(k); k++;
(c) for (int k = 0; k < 100; k++); System.out.println(k);
```

Answer:

```
(a) Error: The exit condition will never be met.
(b) Error: There are braces missing around the loop body, so
      k will never be updated and an infinite loop will result.
(c) Output: The output will be all integers from 0 to 99.
```

5. Write pseudocode algorithms for the following activities, paying particular attention to the *initializer*, *updater*, and *boundary condition* in each case.

```
(a) a softball game
(b) a five-question quiz
(c) looking up a name in the phone book
```

Answer:

```
(a) a softball game:
  for (inning = 1, inning <= 9, inning ++){
      let the home team bat and the away team field.
      let the home team field and the away team bat.
 }
(b) a five-question quiz:
  for (question = 1, question <= 5, question ++){
      read the question,
      answer the question,
   }
(c) looking up a name in the phone book:
  start at first name in the phone book
  while (haven't found name yet)
      if the current name is NOT the right one
          go onto the next name
```

6. Identify the pre- and post-conditions for each of the following statements. Assume that all variables are `int` and have been properly declared.

```
(a)    int result = x / y;
(b)    int result = x % y;
(c) int x = 95;  do     x /= 2;  while (x >= 0);
```

Answer:

```
(a) // Pre: result == 0
    int result = x / y;
    // Post: result == x/y
(b) // Pre: result == 0
    int result = x % y;
    // Post: result == x%y
(c) int x = 95;
    // Pre: x == 95
    do  x /= 2;   while (x >= 0);
    // Post: x < 0
```

7. Write three different loops — a for loop, a while loop, and a do-while loop — to print all the multiples of 10, including 0, up to and including 1,000.

   Answer:

```
for (int j = 0; j <= 1000; j++)     // For loop
    if (j % 10 == 0)  System.out.println(j);

int j = 0;                          // While loop
while (j <= 1000)
{
    if (j % 10 == 0)  System.out.println(j);
    j++;
}

int j = 0;                         // do-while loop
do
{
    if (j % 10 == 0)  System.out.println(j);
    j++;
 } while (j <= 1000);
```

8. Write three different loops — a for loop, a while loop, and a do-while loop — to print the following sequence of numbers: 45, 36, 27, 18, 9, 0, -9, -18, -27, -36, -45.

   Answer:

```
for (int j = 5; j >= -5; j--)  // for loop
    System.out.println(9*j);

int j = 5;                 // while loop
while (j >= -5)
{
    System.out.println(9*j);
    j--;
}
```

```
int j = 5;                  // do-while loop
do
{
    System.out.println(9*j);
    j--;
} while (j >= -5);
```

9. Write three different loops — a for loop, a while loop, and a do-while loop — to print the following ski-jump design:

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

Answer:

```
for (int row = 1; row <= 7; row++){   // for loop
    for (int col = 1; col <= row; col++)
        System.out.print('#');
     System.out.println();
}

int row = 1;                          // while loop
while (row <= 7) {
   int col = 1;
   while (col <= row) {
       System.out.print('#');
       col++;
   }
   System.out.println();
   row++;
}

int row = 1;                      // do-while loop
do {
    int col = 1;
    do {
        System.out.print('#');
        col++;
    } while (col <= row);
    System.out.println();
    row++;
} while (row <= 7);
```

10. The Straight Downhill Ski Lodge in Gravel Crest, Vermont, gets lots of college students on breaks. The lodge likes to keep track of repeat visitors. Straight Downhill's database includes an integer variable, *visit*, which gives the number of times a guest has stayed at the lodge (one or more). Write the pseudocode to catch those visitors who have stayed at the lodge at least twice and to send them a special promotional package (pseudocode = send promo). (*Note:* The largest number of stays recorded is eight. The number nine is used as an end-of-data flag.)

    Answer:

```
Read a visitor record
while (visit != 9)
    if (visit >= 2)
        Send visitor a pamphlet
    Read the next visitor record
```

11. Modify your pseudocode in the previous exercise. In addition to every guest who has stayed at least twice at the lodge receiving a promotional package, any guest with three or more stays should also get a $40 coupon good for lodging, lifts, or food.

    Answer:

```
Read a visitor record
while (visit != 9)
    if (visit >= 2)
        Send visitor a pamphlet
    if (visit >= 3)
        Send visitor a coupon for lodging, lift, or food
    Read the next visitor record
```

12. Write a method that is passed a single parameter, *N*, and displays all the even numbers between 1 and *N*.

    Answer:

```
/**
 * evens() displays all the even numbers between 1 and its
 * parameter
 * @param n -- the upper bound for the number sequence
 */
private void evens (int n) {
    for (int j = 1; j <= n; j++)
        if (j % 2 == 0)
            System.out.println(j);
} // evens()
```

13. Write a method that is passed a single parameter, *N*, that prints all the odd numbers between 1 and *N*.

    Answer:

```
/**
 * odds() displays all the odd numbers between 1 and its
 * parameter
 * @param n -- the upper bound for the number sequence
 */
private void odds (int n) {
    for (int j = 1; j <= n; j++)
        if (j % 2 != 0)
            System.out.println(j);
} // odds()
```

14. Write a method that is passed a single parameter, *N*, that prints all the numbers divisible by 10 from *N* down to 1.

    Answer:

```
/**
 * divby10() displays all numbers divisible by 10 between 1
 * and its parameter
 * @param n -- the upper bound for the number sequence
 */
private void divby10 (int n) {
    for (int j = 1; j <= n; j++)
        if (j % 10 == 0)
            System.out.println(j);
} // divby10()
```

15. Write a method that is passed two parameters — a `char Ch` and an `int N`, and prints a string of *N* Chs.

    Answer:

```
/**
 * chars() prints a sequence of n characters
 * @param n -- the number characters in the sequence
 * @param ch -- the characters that are printed
 */
private void chars (char ch, int n) {
    for (int j = 1; j <= n; j++)
        System.out.print(ch);
} // chars()
```

16. Write a method that uses a nested for loop to print the following multiplication table:

```
   1  2  3  4  5  6  7  8  9
1  1
2  2  4
3  3  6  9
4  4  8 12 16
```

```
5  5 10 15 20 25
6  6 12 18 24 30 36
7  7 14 21 28 35 42 49
8  8 16 24 32 40 48 56 64
9  9 18 27 36 45 54 63 72 81
```

Answer:

```
/**
 * multTable() prints a triangular multiplication table
 *  with row and column headings
 */
private static void multTable () {
    for (int col = 1; col <= 9; col++) // Column headings
        System.out.print(col + " ");
    System.out.println();
    for (int row = 1; row <= 9; row++) {
        System.out.print(row + " ");   // Row heading
        for (int col = 1; col <= row; col++)
            System.out.print(col * row + " ");
        System.out.println();
    }
} // multTable()
```

17. Write methods that use nested for loops to print the following patterns. Your method should use the following statement to print the patterns: `System.out.print('#')`.

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

Answer:

```
/*
 * File: Patterns.java
 * Author: Java, Java, Java
 * Description: This program uses nested loops to
 *  draw four two-dimensional patterns.
 */

public class Patterns {

    private static void pattern1 () {
```

```
    for (int row = 1; row <= 8; row++) {
        for (int k = 2; k <= row; k++) // Skip some blanks
            System.out.print("  ");
        for (int n = 9 - row; n >= 1; n--)
            System.out.print("# ");
        System.out.println();
    }
} // pattern1()

private static void pattern2 () {
    for (int row = 1; row <= 8; row++) {
        for (int n = 9 - row; n >= 1; n--)
            System.out.print("# ");
        System.out.println();
    }
} // pattern2()

private static void pattern3 () {
    for (int row = 1; row <= 8; row++) {
        for (int col = 1; col <= 8; col++)
            if (row == 1 || row == 8)
                System.out.print("# ");
            else if (row == col)
                System.out.print("# ");
            else if (row == 9 - col)
                System.out.print("# ");
            else
                System.out.print("  ");
            System.out.println();
    }
} // pattern3()

private static void pattern4 () {
    for (int row = 1; row <= 8; row++) {
        for (int col = 1; col <= 8; col++)
            if (row == 1 || row == 8)
                System.out.print("# ");
            else if (row == 9 - col)
                System.out.print("# ");
            else
                System.out.print("  ");
            System.out.println();
    }
} // pattern4()

public static void main(String args[]) {
    pattern1();
    pattern2();
    pattern3();
    pattern4();
```

```
    } // main
} // Patterns
```

18. Write a program which asks the user for the number of rows and the number of columns in a box of asterisks. Then use nested loops to generate the box.

    Answer:

```
/*
 * File: Box.java
 * Author: Java, Java, Java
 * Description: This program draws a rectangular box of
 *  asterisks given the number of rows and columns from
 *  the user.
 */

import java.io.*;

public class Box {

    /**
     * drawBox() draws a row x col box of asterisks
     * @param row -- an int giving the number of rows
     * @param col -- an int giving the number of columns
     */
    public static void drawBox (int row, int col) {
        for (int k = row; k > 0; k--) {
            for (int j = col;j > 0; j--)
                System.out.print('*');
            System.out.println();
        }
    } // drawBox()

    /**
     * main() inputs the number of rows and columns from
     * the user and invokes the Box.drawBox() method to
     * draw a box.
     */
    public static void main(String args[])
                                        throws IOException{
        int nRows = 0;
        int nColumns = 0;

        KeyboardReader input = new KeyboardReader();

        System.out.print("Enter the number of rows you " +
                                       "want in your box: ");
        nRows = input.getKeyboardInteger();
        System.out.print("Enter the number of colums you " +
                                       "want in your box: ");
```

```
        nColumns =  input.getKeyboardInteger();
        System.out.println("This is what your box looks like\n");
        Box.drawBox(nRows, nColumns);
    } // main()
} // Box
```

19. Write a Java application that lets the user input a sequence of consecutive numbers. In other words, the program should let the user keep entering numbers as long as the current number is one greater than the previous number.

    Answer:

```
/*
 * File: ConsecutiveSequence.java
 * Author: Java, Java, Java
 * Description: This program lets the user input a
 *  sequence of consecutive integers. The sequence
 *  ends when an input value is "out of sequence."
 */

import java.io.*;

public class ConsecutiveSequence {

    /**
     * main() does all the work in this program. It
     * inputs a sequence of consecutive integers
     * terminated by any out-of-sequence value.
     *
     * The loop uses a sentinel bound. A do-while
     * structure is used  because there has to be at
     * least one iteration.
     */
    public static void main(String args[])
                                throws IOException{

        KeyboardReader input = new KeyboardReader();

        System.out.println("Input consecutive " +
                           "integers, one at a time.");
        System.out.println("Any out-of-sequence " +
                           "value exits the program.");
        System.out.print("Enter the first value: ");
           // Read first value.
        int newNum = input.getKeyboardInteger();
        int prevNum = 0; // Initialize previous value
        do {
            prevNum = newNum; // Remember prev value
               //  Input the next value
            System.out.print("Enter the next value: ");
```

```
            newNum = input.getKeyboardInteger();
        } while (prevNum + 1 == newNum);
          // Until: an out-of-sequence value
        System.out.println("Thank you.");
    } // main()
} // ConsecutiveSequence
```

20. Write a Java application that lets the user input a sequence of integers terminated by any negative value. The program should then report the largest and smallest values that were entered.

    Answer:

```
/*
 * File: SequenceTester.java
 * Author: Java, Java, Java
 * Description: This program lets the user input any
 * sequence of positive integers. It terminates as
 * soon as the user inputs a negative value. It then
 * reports the largest and smallest values that were
 * entered.
 */

import java.io.*;

public class SequenceTester {

    /**
     * main() does all the work in this program. It
     * inputs a sequence of integers terminated by a
     * negative value.For each integer it compares it
     * to the largest and smallest so far, updating
     * these variables if necessary. When the loop
     * terminates, it reports the largest and smallest
     * values.
     *
     * The loop uses a sentinel bound. A while structure
     * is used because it's quite possible that the first
     * value entered will be the sentinel.
     */
    public static void main(String args[])
                                      throws IOException{

        KeyboardReader input = new KeyboardReader();

        System.out.println("This finds the largest and " +
                           "smallest values in a sequence.");
        System.out.print("Enter a positive integer (or " +
                                    "negative to quit): ");
```

```
        int newNum = input.getKeyboardInteger();
          // Initialize the largest and smallest
        int largestNum = newNum;
        int smallestNum = newNum;

        while (newNum >= 0) {       // Sentinel bound
            if (newNum < smallestNum)
                smallestNum = newNum;
            else if (newNum > largestNum)
                largestNum = newNum;
                      // Update step: Input the next value
            System.out.print("Enter a positive integer " +
                             "(or negative to quit): ");
            newNum = input.getKeyboardInteger();
        } // while
        System.out.println("The smallest number entered was "
                                          + smallestNum);
        System.out.println("The largest number entered was "
                                           + largestNum);
    } // main()
} // SequenceTester
```

21. How many guesses does it take to guess a secret number between 1 and *N*? For example, I'm thinking of a number between 1 and 100. I'll tell you whether your guess is too high or too low. Obviously, an intelligent first guess would be 50. If that's too low, an intelligent second guess would be 75. And so on. If we continue to divide the range in half, we'll eventually get down to one number. Because you can divide 100 seven times (50,25,12,6,3,1,0), it will take at most seven guesses to guess a number between 1 and 100. Write a Java applet that lets the user input a positive integer, *N*, and then reports how many guesses it would take to guess a number between 1 and *N*.

    Answer:

```
/*
 * File: GuessingGame.java
 * Author: Java, Java, Java - Part of a solution to Exercise 6.21.
 * Description: This program calculates the maximum number
 *  of guesses required to guess a secret number between
 *  1 and N.
 */

public class GuessingGame {

    /**
     * countGuesses() counts the number of guesses needed to
     *  guess a secret number between 1 and its parameter
     * @param num -- an int giving the bound on the secret
     *   number range
     * @return -- an int giving the number of guesses required
```

```
     * Algorithm: Guessing a secret number in the sequence 1..N
     * requires that you repeatedly divide the sequence in half
     * until there's only one number left. This requires a
     * non-counting loop and uses a limit bound, with the limit
     * being > 0.
     */
    public static int countGuesses(int num) {
        int numOfGuesses = 0;
        int k = num;     // Initially the sequence is 1 to k
        while (k > 0) {  // As long as k is nonzero
            k /= 2;        //  Divide the sequence in half
            numOfGuesses++;//  And count the guess
        } // while
        return numOfGuesses;
    } // countGuesses()

} // GuessingGame

/*
 * File: GuessingApplet.java
 * Author: Java Java Java - Part of a solution to
 * Exercise 6.21.
 * Description: This applet has a prompt JLabel, an input
 * JTextField and an output JTextField and adds them to
 * the applet's content pane. The GuessingGame class is
 * used to determine the number of guesses required to guess
 * a number and this number is reported in the output
 * JTextField. A JButton is used to execute the calculation.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GuessingApplet extends JApplet
                               implements ActionListener{

    private JLabel prompt;           // GUI components
    private JTextField inputField;
    private JTextField resultField;
    private JButton button;

   public void init()
    {
        Container pane = getContentPane();
        prompt = new JLabel("Enter max integer for guess:");
        inputField = new JTextField(10);
        resultField = new JTextField(20);
        resultField.setEditable(false);
        button =
          new JButton("Calculate number of guesses needed");
```

```
        button.addActionListener(this);

        pane.add("West", prompt);
        pane.add("Center", inputField);
        pane.add("North", button);
        pane.add("South", resultField);
    }

    /**
     * actionPerformed() handles clicks on the button.
     * It takes the data from the input JTextField, and
     * sends it to the Guessing class to calculate a maximum
     * number of guesses needed to guess the number.  It is
     * displayed in a JTextField.
     * @param e - the ActionEvent that generated this call
     */
    public void actionPerformed(ActionEvent e)
    {
        int max = Integer.parseInt(inputField.getText());
        int num = GuessingGame.countGuesses(max);
        resultField.setText(num + " guesses are needed.");
    } // actionPeformed()

} // GuessingApplet class

<!-- ***********************************************
file: index.html   - Part of solution to Exercise 6.21
************************************************
-->
<html>
<head><title>Guessing Game Applet</title></head>
<body>
<hr>
<applet code="GuessingApplet.class" width=400 height=400>
</applet>
<hr>
<a href="GuessingGame.java">
                        GuessingGame.java source code</a>
<br>
<a href="GuessingApplet.java">
                      GuessingApplet.java source code</a>
</body>
</html>
```

22. Suppose you determine that the fire extinguisher in your kitchen loses X percent of its foam every day. How long before it drops below a certain threshold (Y percent), at which point it is no longer serviceable? Write a Java applet that lets the user input the values X and Y, and then reports how many weeks the fire extinguisher will last?

Answer:

```
/*
 * File: FireExtinguisher.java
 * Author: Java, Java, Java
 * Description:  This program tests the life of a
 *  fire extinguisher using data input by the user.
 *  Given two numbers, X and Y, which
 *  represent the percentage of foam lost each day (X)
 *  and the threshold (Y) beyond which the extinguisher
 *  is no longer useful. A method calculates how many days
 *  the extinguisher will last.
 */

import java.io.*;

public class FireExtinguisher {

    /**
     * calculateLifeTime() counts the number of weeks a
     * extinguisher should last given its daily loss rate
     * and its usefulness threshold
     * @param lossRate -- a double giving the percentage
     *   loss each day
     * @param threshold -- a double giving the percentage
     *   of foam needed
     * @return an double giving the extinguisher's expected
     *   life in weeks
     * Algorithm: This loop uses a limit bound which requires
     * a non-counting loop. The extinguisher starts at 100%
     * full and loses lossRate percent  per day. Note that
     * lossRate must be divided by 100.
     */
    public static double calculateLifeTime(double lossRate,
                                           double threshold) {
        double amountLeft = 100; //Initially, 100 percent full
        double numDays = 0;
        lossRate /= 100.0;       // Turn lossRate to a decimal
        while (amountLeft > threshold) {
                  // Subtract amount lost
            amountLeft -= amountLeft * lossRate;
            numDays += 1.0;                  // Count the days
        }
        return numDays / 7.0;
    } // calculateLifeTime()

} // FireExtinguisher

/*
 * File: ExtinguisherApplet.java
```

```
 * Author: Java Java Java - Part of a solution to
 * Exercise 6.22.
 * Description: This JApplet has a prompt JLabel, two
 * input JTextFields and an output JTextField and adds
 * them to the applet's content pane. The FireExtinguisher
 * class is used to determine the number of weeks a fire
 * extinguisher remains useable given the percent daily loss
 * (input X) and the useable threshold percent (input Y). The
 * result is output a JTextField when a JButton is clicked.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class ExtinguisherApplet extends JApplet
                                  implements ActionListener{

    private JLabel prompt1;   // Prompt for percent daily loss
    private JTextField lossField;  // Percent daily loss
    private JLabel prompt2;   // Prompt for threshold percent.
    private JTextField thresholdField;  // Threshold percent.
    private JTextField resultField;
    private JButton button;

   public void init()
    {
        Container pane = getContentPane();
        JPanel inputPanel = new JPanel();
        prompt1 = new JLabel("Enter daily percent loss:");
        lossField = new JTextField(10);
        prompt2 = new JLabel("Enter threshold percent:");
        thresholdField = new JTextField(10);
        inputPanel.add(prompt1);
        inputPanel.add(lossField);
        inputPanel.add(prompt2);
        inputPanel.add(thresholdField);
        resultField = new JTextField(20);
        resultField.setEditable(false);
        button = new JButton("Calculate number of weeks");
        button.addActionListener(this);

        pane.add("Center", inputPanel);
        pane.add("North", button);
        pane.add("South", resultField);
    }

    /**
     * actionPerformed() handles clicks on the button.
     * It takes the data from the input JTextFields, and
     * sends it to the FireExtinguisher class to calculate
```

```
     * the number of weeks  the fire extinguisher remains
     * useable.  The result is displayed in a JTextField.
     * @param e - the ActionEvent that generated this call
     */
    public void actionPerformed(ActionEvent e)
    {
        double loss =
                  Double.parseDouble(lossField.getText());
        double threshold =
              Double.parseDouble(thresholdField.getText());
        double weeks =
          FireExtinguisher.calculateLifeTime(loss,threshold);
        resultField.setText("Extinguisher useable for " +
                                          weeks + " weeks");
    } // actionPeformed()

} // ExtinguisherApplet class

<!-- ***********************************************
file: index.html   - Part of solution to Exercise 6.22
************************************************
-->
<html>
<head><title>Fire Extiguisher Applet</title></head>
<body>
<hr>
<applet code="ExtinguisherApplet.class" width=400 height=400>
</applet>
<hr>
<a href="FireExtinguisher.java">
                      FireExtinguisher.java source code</a>
<br>
<a href="ExtinguisherApplet.java">
                    ExtinguisherApplet.java source code</a>
</body>
</html>
```

23. Leibnitz's method for computing π is based on the following convergent series: $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots$ How many iterations does it take to compute π to a value between 3.141 and 3.142 using this series? Write a Java program to find out.

Answer:

```
/*
 * File: PiTester.java
 * Author: Java, Java, Java - A solution to Exercise 6.23
 * Description: This program tests Leibniz's method for
 * computing the value of PI. Leibniz's method is based
 * on the following infinite series:
```

```
 *   PI/4 = 1 - 1/3 + 1/5 - 1/7 + ...
 *
 * The program counts how many iterations are needed to
 * calculate PI to between 3.141 and 3.142.  Note that
 * the countIterations() method is declared static. So it
 * is not necessary to create an instance of this class
 * to use that method.
 */

public class PiTester {

    /**
     * countIterations() counts the number of iterations
     * required to compute PI using Leibniz's method, which
     * is based on the formula PI/4 = 1 - 1/3 + 1/5 - 1/7 + ...
     * @return -- a long integer  giving the number of iterations
     * Algorithm: This loop uses a limit bound. It terminates
     * when Leibniz's value is between 3.141 and 3.142.
     */
    public static long countIterations() {
        double n = 1.0;
        double valueOfSeries = 0;
        long counter = 0;
        while ((4*valueOfSeries <= 3.141) ||
                              (4*valueOfSeries >= 3.142)) {
            valueOfSeries += 1.0 / n;
            n = -n;
            if (n > 0)
                n += 2.0;
            if (n < 0)
                n -= 2;
            counter++;
        }
        System.out.println("Leibniz's PI = "+valueOfSeries * 4);
        return counter;
    } // countIterations()

    /**
     * main() invokes PiTester.countIterations() to compute
     *  the value of PI and count the iterations. It then
     *  prints the results.
     */
    public static void main(String args[]) {
        System.out.println("Math.PI = " + Math.PI);
        long count = PiTester.countIterations();
        System.out.println("It takes " + count
                           + " iterations to calculate PI"
                           + " using Leibniz's method.");
    } // main()
} // PiTester
```

24. Newton's method for calculating the square root of *N* starts by making a (non zero) guess at the square root. It then uses the original guess to calculate a new guess, according to the following formula:

```
guess = (( N / guess) + guess) / 2;
```

No matter how wild the original guess is, if we repeat this calculation, the algorithm will eventually find the square root. Write a square root method based on this algorithm. Then write a program to determine how many guesses are required to find the square roots of different numbers. Use `Math.sqrt()` to determine when to terminate the guessing.

Answer:

```
/*
 * File: NewtonTester.java
 * Author: Java, Java, Java A solution to Exercise 6.24
 * Description: The program uses Newton's method to calculate
 *  square roots. Newton's method is based on the formula:
 *         guess = (( N / guess) + guess) / 2;
 *  To compute a square root, start with any nonzero guess,
 *  then use the formula to update the guess until it converges
 *  on the square root. The program tests how many guesses are
 *  needed by comparing Newton's value with Math.sqrt().
 */


public class NewtonTester {

    /**
     * countGuesses() counts the number of guesses needed to
     *  compute the square root of num using Newton's method
     * @param num -- the number whose square root is computed
     * @param guess -- the initial guess (should be nonzero)
     * @return an int giving the number of guesses required
     * Algorithm: The loop in this method uses a limit bound.
     *  It repeatedly computes Newton's value and compares it
     *  to Math.sqrt() until their difference is less than 0.001.
     */
    public static int countGuesses(double num, double guess) {
        int counter = 0;
        double sqrtN = Math.sqrt(num);
        while (Math.abs(sqrtN - guess) > 0.001 ) {
            guess = ((num / guess) + guess) / 2;
            counter++;
            System.out.println("New guess = " + guess);
        } // while
        return counter;
    } // countGuesses()
```

```
    /**
     * main() inputs a number from the user and then computes
     * its square root using Newton's method. It reports how
     * many iterations Newton's method takes to converge on
     * the root.
     */
    public static void main(String args[]){
        double number; // The number to find square root of
        double firstGuess;  // User's guess

        KeyboardReader input = new KeyboardReader();
        NewtonTester tester = new NewtonTester();

        System.out.println("Newton's method calculates " +
                                        "a square root.");
        System.out.print("Enter a number to take the " +
                                       "square root of: ");
        number = input.getKeyboardDouble();

        System.out.print("Enter a positive guess for square ");
        System.out.print("root of " + number + ": ");
        firstGuess = input.getKeyboardDouble();
        System.out.println("The exact square root is " +
                                         Math.sqrt(number));
        int numOfGuesses = tester.countGuesses(number,
                                               firstGuess);
        System.out.print("Newton takes " + numOfGuesses +
                                               " guesses to");
        System.out.println(" calculate the square root " +
                                               "within .001");
    } // main()
} // NewtonTester
```

25. Your employer is developing encryption software and wants you to develop a Java applet that will display all of the primes less than *N*, where *N* is a number to be entered by the user. In addition to displaying the primes themselves, provide a count of how many there are.

    Answer:

```
/*
 * File: PrimesTester.java
 * Author: Java, Java, Java
 * Description: This program lets the user input an integer,
 *  and then displays all the prime numbers between 1 and that
 *  integer. It also provides a count of the number of such
 *  primes.  The main() method provides a command line user
 *  interface.  The countPrimesLessThanN() method can also
 *  be used in a GUI or applet user interface.
 */
```

```
import java.io.*;

public class PrimesTester {

    /**
     * countPrimesLessThanN() prints the primes between 1 and
     * n and returns their count.
     * @param n -- the upper bound on the sequence of primes
     * @return -- an int giving the number of primes between
     *   1 and n.
     * Algorithm: To test every value between 1 and n we can
     * use a counting loop. A more efficient algorithm would
     * print 1, 2, 3 and then check every other value between
     * 3 and n.
     */
    public String countPrimesLessThanN(int n) {
        int counter = 0;
        String primeList = "";
        for (int num = 1; num < n; num++) {
            if (isPrime(num)) {
                primeList = primeList + num + " ";
                counter++;
                if (counter % 10 == 0)
                    primeList = primeList + "\n";
            } // if
        } // for
        primeList = primeList + "\nThere are " + counter;
        primeList = primeList + " primes less than " + n;
        return primeList;
    } // countPrimesLessThanN()

    /**
     * isPrime() returns true iff its parameter is a prime
     * @param n -- the value to be tested for primality
     * @return -- a boolean set to true iff n is prime
     * Algorithm: The primality test attempts to divide n by
     *  every value k between 2 and the square root of n
     *  or what is equivalent every k with k * k <= n.
     *  The loop exits when n is
     *  divisible by k or if it passes each test.
     */
    public boolean isPrime(int n) {
        boolean notDivisibleYet = true;
        int k = 2;
        while (k * k < n + 1) {
            if (n % k == 0) {
                return false;
            }
            k++;
```

```
        } // while
        return true;
    } // isPrime()

    /**
     * main() prompts the user for a number and creates
     *  and instance of the PrimesTester class to print
     *  all the primes between 1 and the user's bound.
     */
    public static void main(String args[])
                                    throws IOException {
        String inputString;
        int bound;        // The user's bound
             // The tester object
        PrimesTester tester = new PrimesTester();

        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));

        System.out.print("Enter a number: ");
        inputString = input.readLine();
        bound = Integer.parseInt(inputString);
        System.out.println("The primes less than " +
                                        bound + " are:");
        System.out.println(tester.countPrimesLessThanN(bound));
    }// main()
} // PrimesTester

/*
 * File: PrimesApplet.java
 * Author: Java Java Java - Part of a solution to
 * Exercise 6.25.
 * Description: This JApplet has a prompt JLabel, an
 * input JTextField and an output JTextArea and adds
 * them to the applet's content pane. The PrimesTester
 * class is used to determine the number of primes less
 * than an input number when a JButton is clicked.  The
 * list of primes and a count of them are output to the
 * JTextArea.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class PrimesApplet extends JApplet
                           implements ActionListener{

    private JLabel prompt; // Prompt for a large integer.
    private JTextField numField;  // For the integer.
    private JTextArea outArea;
```

```
    private JButton button;

   public void init()
    {
        Container pane = getContentPane();
        JPanel inputPanel = new JPanel();
        prompt =
          new JLabel("Enter an upper bound for a set " +
                                          "of primes:");
        numField = new JTextField(10);
        button = new JButton("Find Primes");
        button.addActionListener(this);
        inputPanel.add(prompt);
        inputPanel.add(numField);
        inputPanel.add(button);
        outArea = new JTextArea(30,20);

        pane.add("North", inputPanel);
        pane.add("South", outArea);
    }

    /**
     * actionPerformed() handles clicks on the button.
     * It takes the integer from the input JTextField, and
     * sends it to the PrimesTester class to calculate the
     * number of primes less than the input number. The
     * result is displayed in the JTextArea.
     * @param e -- the ActionEvent the generated this call
     */
    public void actionPerformed(ActionEvent e)
    {
        int n  = Integer.parseInt(numField.getText());
        PrimesTester tester = new PrimesTester();
        outArea.setText(tester.countPrimesLessThanN(n));
    } // actionPeformed()

} // PrimesApplet class

<!-- ***********************************************
file: index.html   - Part of solution to Exercise 6.25
************************************************
-->
<html>
<head><title>PrimesApplet</title></head>
<body>
<hr>
<applet code="PrimesApplet.class" width=800 height=600>
</applet>
<hr>
<a href="PrimesTester.java">
```

```
                        PrimesTester.java source code</a>
<br>
<a href="PrimesApplet.java">
                        PrimesApplet.java source code</a>
</body>
</html>
```

26. Your little sister asks you to help her with her multiplication and you decide to write a Java application that tests her skills. The program will let her input a starting number, such as 5. It will generate multiplication problems ranging from from 5×1 to 5×12. For each problem she will be prompted to enter the correct answer. The program should check her answer and should not let her advance to the next question until the correct answer is given to the current question.

    Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: MultTester.java
 * Author: Java, Java, Java
 * Description: This program conducts a drill and
 *  practice tutorial for a student learning the
 *  multiplication tables. The user inputs a starting
 *  value, say 5. Then the program presents multiplication
 *  problems starting at 5 x 1, up through 5 x 12. The
 *  user must provide the correct answer before being
 *  able to advance.
 */

import java.io.*;

public class MultTester {

    /**
     * multDrill() presents a multiplication drill starting at
     *  n x 1 and going up through n x 12. The user must
     *  provide the correct answer before being able to advance.
     * @param n -- the starting value for the drill
     * Algorithm: We can't use a counting loop here because
     *  must force the user to repeat a drill until correct.
     *  So the loop variable, k, is incremented only when the
     *  correct answer is given.
     */
    public static void multDrill (int n) throws IOException{
        String inputString;
        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));
```

```
        int k = 1;
        while (k <= 12) {
            System.out.print(n + " x " + k + " is equal to: ");
            inputString = input.readLine();
            int answer = Integer.parseInt(inputString);
            if (answer != k * n)
              System.out.println("That is incorrect, try again");
            else {
              System.out.println("Very good, that is the " +
                                                 "correct answer");
              k++;
            }
        } // while
    } // multDrill()

    /**
     * main() creates an instance of MultTester and uses it to
     *  perform the multiplication drill. The user is prompted
     *  for a starting value. To exit the program, the user
     *  should enter 0.
     * Algorithm: This loop employs a sentinel bound (0) and
     *  it has to be a noncounting loop, because you don't know
     *  in advance how many drills the users wants to do.
     */
    public static void main(String args[]) throws IOException {
        String inputString;
        int startNumber = 0;
            // Create an instance
        MultTester tester = new MultTester();

        BufferedReader input =
          new BufferedReader(new InputStreamReader(System.in));

        do {
          System.out.println("To practice your multiplication,\n"
            + " enter a positive number.  Or enter 0 to quit.");
          System.out.print("Enter a number: ");
          inputString = input.readLine();
          startNumber = Integer.parseInt(inputString);
          if (startNumber > 0)
              multDrill(startNumber);
        } while (startNumber != 0);
    } // main()
} // MultTester
```

27. Write an application that prompts the user for four values and draws corresponding bar graphs using an ASCII character. For example, if the user entered 15, 12, 9, and 4, the program would draw

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

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

Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: Grapher.java
 * Author: Java, Java, Java
 * Description: This program draws a horizontal bar
 *  graph from input received from the user.
 */

import java.io.*;

public class Grapher {

    /**
     * drawBar() draws a horizontal bar of length n
     * @param n -- the number of asterisks in the bar
     * Algorithm: Because we know how many asterisks to
     *  to draw, we use a counting (for) loop
     */
    public void drawBar(int n) {
        for (int k = 0; k < n; k++) {
            System.out.print('*');
        }
        System.out.println();
    } // drawBar()

    /**
     * main() gets the input from the user and uses an
     *  instance of this class to draw the bar graph.
     */
    public static void main(String args[]) throws IOException {

        int n1, n2, n3, n4;  // Store input data for the bar graph
        String inputString;

                      // Get the user's input
        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));
        System.out.print("Enter a value for the bar graph: ");
        inputString = input.readLine();
        n1 = Integer.parseInt(inputString);
        System.out.print("Enter a value for the bar graph: ");
        inputString = input.readLine();
```

```
        n2 = Integer.parseInt(inputString);
        System.out.print("Enter a value for the bar graph: ");
        inputString = input.readLine();
        n3 = Integer.parseInt(inputString);
        System.out.print("Enter a value for the bar graph: ");
        inputString = input.readLine();
        n4 = Integer.parseInt(inputString);

             // Create a Grapher instance and
            //  draw the graph
        Grapher grapher = new Grapher();
        System.out.println("Your graph looks like:\n");
        grapher.drawBar(n1);
        grapher.drawBar(n2);
        grapher.drawBar(n3);
        grapher.drawBar(n4);
    } // main()
} // Grapher
```

28. Revise the application in the previous problem so that the bar charts are displayed vertically. For example, if the user inputs 5, 2, 3, and 4, the program should display

```
 **
 **        **
 **     ** **
 ** ** ** **
 ** ** ** **
-------------
```

Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: BarGraph.java
 * Author: Java, Java, Java
 * Description: This program prints a vertical bar graph
 *  with the four values input by the user. The trick to
 *  solving this problem is knowing the size of the largest
 *  bar in the graph. The getLargest() method determines
 *  that value.
 */

import java.io.*;

public class BarGraph {
```

```
/**
 * getLargest() returns the value of the largest of its
 *  four int parameters.
 * @param -- four integer values
 * @return -- the value of the largest parameter
 * Algorithm: For each of the 4 parameters, assume that
 *  it is the largest and compare it with all the others.
 *  If it is greater than or equal to the others, set
 *  the boolean flag (foundLargest) to true. This will
 *  cause the loop to exit. Note that we have used a
 *  for loop with a compound entry condition. The can
 *  terminate because all four cases have been tried or
 *  because the first value (a) is the largest.
 */
public static int getLargest(int a, int b, int c, int d) {
    int largest = 0;
    boolean foundLargest = false;
       // For each of the 4 params
    for (int k = 0; k < 4 && !foundLargest; k++) {
        if (k == 0)          // Assume it is the largest
            largest = a;
else if (k == 1)
            largest = b;
        else if (k == 2)
            largest = c;
        else
            largest = d;
        if (largest >= a && largest >= b &&
                            largest >= c && largest >= d)
            foundLargest = true;
    } // for
    return largest;
} // getLargest()

/**
 * printGraph() prints a vertical bar graph of its four
 * parameter values. To determine the height of the graph
 * it first begins by finding the largest of the four
 * parameters. That determines the bound on the outer for
 * loop, which determines the height of the graph. The inner
 * for loop prints each row. So it goes through each of the
 * four variables and either prints asterisks or spaces for
 * that variable, depending on whether that variable is
 * greater than the current height or not.
 */
public static void printGraph(int a, int b, int c, int d) {
    int number = 0;
       // For each row
    for (int hgt = getLargest(a,b,c,d); hgt > 0; hgt--) {
          // For each variable
```

```
            for (int var = 0; var < 4; var++) {
                if (var == 0)
                    number = a; // Assign its value to number
                else if (var == 1)
                    number = b;
                else if (var == 2)
                    number = c;
                else
                    number = d;
                      // If the var's value is high enough
                if (number >= hgt)
                    System.out.print("**  "); //print asterisks
                else                // Otherwise print blanks
                    System.out.print("    ");
            } // inner for
            System.out.println();  // Start a new line
        } // outer for
    } // printGraph()

    /**
     * main() inputs four integers and prints a horizontal
     *  bar graph showing their relative values.
     */
    public static void main(String args[])
                                        throws IOException{
        String inputString;
        int A, B, C, D;     // The four input values

        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));

        System.out.println("Enter the first value:");
        inputString = input.readLine();
        A = Integer.parseInt(inputString);
        System.out.println("Enter the second value:");
        inputString = input.readLine();
        B = Integer.parseInt(inputString);
        System.out.println("Enter the third value:");
        inputString = input.readLine();
        C = Integer.parseInt(inputString);
        System.out.println("Enter the fourth value:");
        inputString = input.readLine();
        D = Integer.parseInt(inputString);
        System.out.println("This is what your graph looks like");
        printGraph(A,B,C,D);
    } // main()
} // BarGraph
```

29. The Fibonacci sequence (named after the Italian mathematician Leonardo of Pisa, ca.1200) consists of the numbers $0, 1, 1, 2, 3, 5, 8, 13, \ldots$ in which each

number (except for the first two) is the sum of the two preceding numbers. Write a method `fibonacci(N)` that prints the first *N* Fibonacci numbers.

Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: Fibonacci.java
 * Author: Java, Java, Java
 * Description:  This program prints the first N values
 *  of the Fibonacci sequence. The sequence starts with
 *  0, 1, 1, 2, 3, 5, 8, 13, ... Each term in the sequence
 *  (except the first two) is the sum of the two previous
 *  terms.
 */

import java.io.*;

public class Fibonacci {

    /**
     * doFibonacci() prints the first n values of the
     *  fibonacci sequence.
     * @param n -- an int giving the number of values to print
     * Algorithm: The bound in this case is a counting bound
     *  and therefore we can use a counting (for) loop.
     */
    public static void doFibonacci(int n) {
        int a = 0;
        int b = 1;
        int c = 0;
        for (int counter = 1; counter <= n; counter++) {
            if (counter == 1)
                System.out.print(a + ", ");
            if (counter == 2)
                System.out.print(b + ", ");
            if (counter >= 3) {
                c = a + b;
                a = b;
                b = c;
                System.out.print(c + ", ");
            }
        } // for
        System.out.println();
    } // doFibonacci

    /**
     * main() inputs a number, N,  from the user and then
     * prints the first N values of the fibonacci sequence.
```

```
     */
    public static void main(String args[])
                                        throws IOException{
        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));

        System.out.println("To print the first N " +
                                      "fibonacci numbers.");
        System.out.print("Input a value for N: ");
        String inputString = input.readLine();
        int number = Integer.parseInt(inputString);
        System.out.println("The first " + number +
                               " fibonacci numbers are: ");
        Fibonacci.doFibonacci(number);
    } // main()
} // Fibonacci
```

30. The Nuclear Regulatory Agency wants you to write a program that will help them determine how long certain radioactive substances will take to decay. The program should let the user input two values: a string giving the substance's name, and its half-life in years. (A substances half-life is the number of years required for the disintegration of half of its atoms.) The program should report how many years it will take before there is less than 2 percent of the original number of atoms remaining.

    Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: HalfLife.java
 * Author: Java, Java, Java
 * Description: This program calculates how many years
 * will transpire before a substance X with a half life
 * of Y will have less than 2 percent of its radioactivity
 * left.
 */

import java.io.*;

public class HalfLife {

    /**
     * calcYears() computes the number of years a substance
     * with a half life of n years will last before 2 percent
     * of its radioactive atoms are left
     * @param n -- an int giving the stuff's half life
     * @return -- an int giving the number of years of life
     * Algorithm:  The loop in this case uses a limit bound.
```

```
     *  It will iterate as long as the amount is greater than
     *  2 percent of the original amount.
     */
    public static int calcYears(int n) {
        double amt = 100; //Start with 100 percent of the stuff
        int years = 0;
        while (amt > 2) {      // While more than 2 percent
            amt -= amt * 0.5;  // Divide the substance in half
            years += n;        // Count the years
        }
        return years;
    } // calcYears()

    /**
     * main() inputs the name and half life of a radioactive
     * substance and then calls a method to calculate the
     * number of years it will last.
     */
    public static void main(String args[]) throws IOException{
        BufferedReader input =
          new BufferedReader(new InputStreamReader(System.in));

        System.out.println("The lifetime of a radioactive " +
                                    "substance is computed. ");
        System.out.print("Enter the name of the substance: ");
        String name = input.readLine();
        System.out.print("Enter the half life of the " +
                                     "substance (in years): ");
        String inputString = input.readLine();
        int halfLife = Integer.parseInt(inputString);
        System.out.println("The " + name + " will last for " +
         calcYears(halfLife)+" years before less then 2% is left");
    } // main()
} // HalfLife
```

31. Modify the `CarLoan` program so that it calculates a user's car payments for loans of different interest rates and different loan periods. Let the user input the amount of the loan. Have the program output a table of monthly payment schedules.

    Answer: The following class definition uses the `BufferedReader` class from the java.io.* package for keyboard input. Obviously an alternate way to get keyboard input would be to use the `KeyboardReader` class and `UserInterface` interface from Chapter 4.

```
/*
 * File: CarLoan.java
 * Author: Java, Java, Java
 * Description:  This program prints a table showing the
 *  total cash outlay for a car purchase. The table gives
```

```
 *  a breakdown for different interest rates and for 2
 *  through 8 years.
 */

import java.io.*;
import java.text.NumberFormat;

public class CarLoan {
    /**
     * convertStringToDouble() converts its string parameter
     * to a double
     * @param s -- a String to be converted
     * @return a double representing the number stored in s
     */
    private static double convertToDouble(String s) {
        Double doubleObject = Double.valueOf(s);
        return doubleObject.doubleValue();
    } // convertToDouble()

    /**
     * printTable() prints a two-dimensional table showing a
     * car payment schedule for different loans for a given
     * priced car
     * @price -- a double giving the car's price
     */
    private static void printTable(double price) {
        double loanPrice;

        NumberFormat dollars =
                        NumberFormat.getCurrencyInstance();
        NumberFormat percent =
                         NumberFormat.getPercentInstance();
        percent.setMaximumFractionDigits(2);

        for (int rate = 8; rate <= 11; rate++)
            System.out.print("\t" + percent.format(rate/100.0));
        System.out.println("");

        for (int years = 2; years <= 8; years++) {
          System.out.print("Year " + years + "\t");
          for (int rate = 8; rate <= 11; rate++) {
            loanPrice = price *
              Math.pow(1 + rate / 100.0 / 365.0, years * 365.0);
            System.out.print(dollars.format(loanPrice /
                                          (years*12))+"\t");
          }
          System.out.println("");
        }
    } // printTable()
```

```
    public static void main(String args[]) throws IOException{
        BufferedReader input = new BufferedReader
            (new InputStreamReader(System.in));
        String inputString;
        double price;

        System.out.print("Enter the price of your car: ");
        inputString = input.readLine();
        price = CarLoan.convertToDouble(inputString);
        System.out.println("This is what your payments " +
                                        "would look like\n");
        CarLoan.printTable(price);
    } // main()
} // CarLoan
```

The next chapter also contains a number of loop exercises.

Chapter 7

Strings and String Processing

1. Explain the difference between the following pairs of terms.

   (a) *Unit indexing* and *zero indexing*. **Answer:** Zero indexing is when the index of the first character in the string is zero. If it is equal to one it is called unit indexing.

   (b) *Data structure* and *data type*. **Answer:** Data structure is a collection of data that is organized in some way. Data type is just the type of data that is present such as integer, character, boolean, double or long.

   (c) `StringBuffer` and `String`. **Answer:** A String is an object that cannot be changed at all or added to. A StringBuffer is a class that allows you to add characters to it and at the end output a String.

   (d) `String` and `StringTokenizer`. **Answer:** A String is an object that cannot be changed or broken up. A StringTokenizer is a class that allows you to break up a String into individual parts.

   (e) *Declaring a variable* and *instantiating a* `String`. **Answer:** When instantiating a String, a new object is not created unless the String() method is used.

   (f) A `Font` and a `FontMetrics` object. **Answer:** A `Font` object stores the name, size, and style of a font that can be associated with a window or button or other component. A `FontMetrics` object stores a reference to a `Font` object and provides methods for accessing properties of that font like the width in pixels of a particular letter.

2. Fill in the blanks.

   (a) When the first character in a string has index 0, this is known as _____. **Answer: zero indexing**

   (b) A sequence of characters enclosed within quotes is known as a _____. **Answer:String literal**

3. Given the `String str` with the value "to be or not to be that is the question," write Java expressions to extract each of the substrings shown below. For each substring, provide two sets of answers. One that uses the actual index numbers of the substrings — for example, the first "to" goes from 0 to 2 — and the second, more general solution that will retrieve the same substring from the following string "it is easy to become what you want to become." (*Hint*: In the second case, use `length()` and `indexOf()` along with `substring()` in your expressions. If necessary, you may use local variables to store intermediate results. The answer to (a) is provided as an example.)

   (a) the first "to" in the string

```
str.substring(0,2);    // Answer 1
str.substring(str.indexOf("to"), str.indexOf("to") + 2);  // Answer 2
```

   (b) the last "to" in the string **Answers:**

```
str.substring(14,16);
str.substring(str.lastIndexOf("to"), str.lastIndexOf("to") + 2);
```

   (c) the first "be" in the string **Answers:**

```
str.substring(3,5);
str.substring(str.indexOf("be"), str.indexOf("be") + 2);
```

   (d) the last "be" in the string **Answers:**

```
str.substring(17,19);
str.substring(str.lastIndexOf("be"), str.lastIndexOf("be") + 2);
```

   (e) the first four characters in the string **Answers:**

```
str.substring(0,4);
str.substring(0,4);
```

   (f) the last four characters in the string **Answers:**

```
str.substring(35,39);
str.substring(str.length() - 4, str.length());
```

4. Identify the syntax errors in each of the following assuming that the `String s` equals "exercise."

```
(a) s.charAt("hello")
(b) s.indexOf(10)
(c) s.substring("er")
(d) s.lastIndexOf(er)
(e) s.length
```

Answer:

```
(a) s.charAt("hello")   // "hello" is not an integer
(b) s.indexOf(10)       // 10 is not a char or string
(c) s.substring("er")   // "er" is not an integer
(d) s.lastIndexOf(er)   // er is not a String
(e) s.length            // There are no parentheses after length
```

5. Evaluate each of the following expressions assuming that the `String s` equals "exercise."

```
(a) s.charAt(5)
(b) s.indexOf("er")
(c) s.substring(5)
(d) s.lastIndexOf('e')
(e) s.length()
```

/bf Answers: (a) 'i', (b) 2, (c) "ise", (d) 7, (e) 8

6. Write your own `equalsIgnoreCase()` method using only other `String` methods. **Answer:**

```
/**
 * equalsIgnoreCase() returns true iff String str1 equals String
 *  str2 regardless of the case of their respective letters.
 *  So "hello" will equal "HELLO" in this method.
 * @param str1, str2 -- two strings to be compared
 * @return a boolean set to true iff the strings are equal
 */
public boolean equalsIgnoreCase(String str1, String str2) {
    String s1 = str1.toLowerCase();
    String s2 = str2.toLowerCase();
    return s1.equals(s2);
} // equalsIgnoreCase()
```

7. Write your own `String` equality method without using `String.equals()`. (*Hint*: Modify the `precedes()` method.) **Answer:**

```
/**
 * equals() returns true iff String str1 and String str2
 *  have exactly the same letters.
 * @param str1, str2 -- two strings to be compared
 * @return a boolean set to true iff the strings are equal
 * Algorithm: First the lengths of the strings are compared
 *  then a letter-by-letter comparison is made. If all of the
 *  these comparisons succeed, return true.
 */
public boolean equals(String str1, String str2) {
```

```
    if (str1.length() != str2.length())
        return false;
    else for (int k = 0; k < str1.length(); k++)
        if (str1.charAt(k) != str2.charAt(k))
            return false;
    return true;
} // equals()
```

8. Even though Java's `String` class has a built-in `toLowerCase()` method, write your own implementation of this method. It should take a `String` parameter and returns a `String` with all its letters written in lowercase. **Answer:**

```
/**
 * lowercase() converts its String parameter to lowercase
 * @param str -- a String to be converted
 * @return a String containing only lowercase letters
 * Algorithm: Go through each letter in the string and if
 *  a letter is an uppercase, convert it to lowercase.
 */
public String lowercase(String str){
    StringBuffer result = new StringBuffer();
    for (int k = 0; k < str.length(); k++) {
        if ((str.charAt(k) > 'A') && (str.charAt(k) < 'Z'))
            result.append((char)(str.charAt(k) + 32));
        else
            result.append((char)str.charAt(k));
    }
    return result.toString();
} // lowercase()
```

9. Write a method that converts its `String` parameter so that letters are written in blocks five letters long. For example, consider the following two versions of the same sentence:

```
Plain:    This is how we would ordinarily write a sentence.
Blocked : Thisi showw ewoul dordi naril ywrit easen tence
```

Answer:

```
/**
 * convertToFive() removes the natural word boundaries in s
 *  and creates a new string with blocks of 5 characters
 * @param s -- a String to be processed
 * @return a String with 5-character blocks
 */
public static String convertToFive(String s){
    StringBuffer result = new StringBuffer();
    int nc = 0; //To store the number of non-space chars
    for (int k = 0; k < s.length(); k++) {
```

```
            char ch = s.charAt(k);
            if (((ch >= 'A')&&(ch <= 'Z')) || ((ch >= 'a') &&
                                              (ch <= 'z'))){
                 result.append(s.charAt(k));
                 nc++;
                 if (nc % 5 == 0)  result.append(' ');
               } // if ch is a letter
          } // for
          return result.toString();
      } // convertToFive()
```

10. Design and implement an applet that lets the user type a document into a `JTextArea`, and then provides the following analysis of the document: the number of words in the document, the number of characters in the document, and the percentage of words that have more than six letters.

    Answer:

```
/*
 * File: DocumentAnalyzer.java
 * Author: Java, Java, Java -- Part of solution to
 * Exercise 7.10.
 * Description: This class provides methods for
 * analyzing text documents. It counts words,
 * characters and long words. It stores the
 * document both as a String and as a StringTokenizer,
 * which greatly simplifies the analysis.
 */

import java.util.*;

public class DocumentAnalyzer {
       // The document being analyzed
    private String document;
       // The tokens in the document
    private StringTokenizer words;

    /**
     * DocumentAnalyzer() creates an instance given a
     *  string of text
     * @param s -- a document to be analyzed
     */
    public DocumentAnalyzer(String s) {
        document = new String(s);
        words = new StringTokenizer(s);
    }

    /**
     * countWords() counts the words in the document
     * It assumes that words are delimited by whitespace.
```

```
     * It simply creates a string tokenizer of the document.
     * @return an int giving the number of words
     */
    public int countWords() {
        return words.countTokens();
    } // numberOfWords()

    /**
     * countChars() counts the characters, including
     * whitespace, in the document
     * @return an int giving the number of characters
     */
    public int countChars() {
        return document.length();
    }

    /**
     * countLongWords() counts the number of words having
     * more than n characters. Note that it is necessary
     * to use a a local StringTokenizer here because each
     * time nextToken() is called it removes a token from
     * the StringTokenizer.
     * @param n -- the length of a long word
     * @return an int giving the number of long words
     */
    public int countLongWords(int n) {
        StringTokenizer tokenizer =
                          new StringTokenizer(document);
        int counter = 0;
        while (tokenizer.hasMoreTokens()) {
            String s = tokenizer.nextToken();
            if (s.length() > n)
                counter++;
        }
        return counter;
    } // countLongWords()
} // DocumentAnalyzer

/*
 * File: AnalyzeDocApplet.java
 * Author: Java, Java, Java - Part of a solution to
 * Exercise 7.10
 * Description: This applet provides a user interface
 * for the DocumentAnalyzer object. The user can enter
 * DocumentAnalyzer text into a JTextArea and click a
 * button. The will then report a count of the characters
 * and words, and the percentage of words with more than
 * 6 characters,
 */
```

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


public class AnalyzeDocApplet extends JApplet
                            implements ActionListener {
       // Length of a long word
    private static final int LONG = 6;

    private JLabel prompt;             // GUI components
    private JTextArea inputArea;
    private JButton button;
    private JTextArea outputArea;

    /**
     * init() sets up the GUI for the applet.
     */
    public void init() {
        Container pane = getContentPane();
        prompt = new JLabel("Enter a document to analyze:");
        inputArea = new JTextArea(5,50);
        inputArea.setEditable(true);
        outputArea = new JTextArea(5,50);
        button = new JButton("Analyze");
        button.addActionListener(this);
        pane.add("North", prompt);
        pane.add("Center", inputArea);
        pane.add("West", button);
        pane.add("South", outputArea);

        setSize(600,400);
    } // init()

    /**
     * actionPerformed() creates a DocumentAnalyzer and
     * gets it to perform the analysis of the text in the
     * TextArea.
     */
    public void actionPerformed(ActionEvent e) {
        DocumentAnalyzer analyzer =
            new DocumentAnalyzer(inputArea.getText());
        outputArea.setText("");
        int nWords = analyzer.countWords();
        int nChars = analyzer.countChars();
        int nLongWords = analyzer.countLongWords(LONG);
        outputArea.append("There are " + nWords +
                          " words in your document. \n");
        outputArea.append("There are " + nChars +
                     " characters in your document. \n");
```

```
        outputArea.append((1.0 * nLongWords/nWords * 100) +
                                " per cent of the words");
        outputArea.append(" have more than " +  LONG +
                                      " characters. \n");
    } // actionPerformed()
} // AnaylzeDocApplet

<!-- ***********************************************
file: index.html
************************************************
-->
<html>
<head><title>Analyze Doc Applet</title></head>
<body>
<hr>
<applet code=AnalyzeDocApplet.class width=600 height=400>
</applet>
<hr>
<a href="AnalyzeDocApplet.java">AnalyzeDocApplet.java</a>
<br>
<a href="DocumentAnalyzer.java">DocumentAnalyzer.java</a>
</body>
</html>
```

11. Design and write an applet that searches for single-digit numbers in a text and changes them to their corresponding words. For example, the string "4 score and 7 years ago" would be converted into "four score and seven years ago."

    Answer:

```
/*
 * File: DigitConverter.java
 * Author: Java, Java, Java
 * Description: This class contains methods that
 *  convert the digits in a text file to their
 *  corresponding words -- e.g., '7' to "seven".
 */

import java.util.*;

public class DigitConverter {

    /**
     * isDigit() returns true iff its parameter is a digit
     * @param ch -- a char
     * @return a boolean true iff ch is a digit
     */
    public boolean isDigit (char ch) {
        if (ch >= '0' && ch <= '9')
            return true;
```

```
    else
        return false;
} // isDigit()

/**
 * convert() converts is digit parameter to a word
 *  for example '9' to "nine"
 * @digit a char giving a '0' to '9'
 * @return a String representing the digit
 */
public String convert (char ch) {
    if (ch == '0')
        return "zero";
    else if (ch == '1')
        return "one";
    else if (ch == '2')
        return "two";
    else if (ch == '3')
        return "three";
    else if (ch == '4')
        return "four";
    else if (ch == '5')
        return "five";
    else if (ch == '6')
        return "six";
    else if (ch == '7')
        return "seven";
    else if (ch == '8')
        return "eight";
    else if (ch == '9')
        return "nine";
    else
        return "error";
} // convert()

/**
 * findAndChange() changes each digit in s to
 *  its corresponding number name: '7' to "seven"
 * @param s -- a String of text to be converted
 * @return a String giving the processed text
 */
public String findAndChange(String s) {
    StringBuffer result = new StringBuffer();
    char ch1, ch2, ch3;

    ch2 = s.charAt(0); //Check the first char
    ch3 = s.charAt(1);
    if (isDigit(ch2)&&!isDigit(ch3))
        result.append(convert(ch2));
    else
```

```
            result.append(ch2);

        // Check all but the first and last chars.
        for (int k = 1; k < s.length() - 1; k++) {
            ch1 = s.charAt(k - 1);
            ch2 = s.charAt(k);
            ch3 = s.charAt(k + 1);
            if (!isDigit(ch1)&&isDigit(ch2)&&!isDigit(ch3))
                result.append(convert(ch2));
            else
                result.append(ch2);
        } // for
          //Check the last char
        ch1 = s.charAt(s.length() - 2);
        ch2 = s.charAt(s.length() - 1);
        if (!isDigit(ch1)&&isDigit(ch2))
            result.append(convert(ch2));
        else
            result.append(ch2);

        return result.toString();
    } // findAndChange()
} // DigitConverter

/*
 * File: DigitConverter.java
 * Author: Java, Java, Java
 * Description: This class contains methods that
 *  convert the digits in a text file to their
 *  corresponding words -- e.g., '7' to "seven".
 */

import java.util.*;

public class DigitConverter {

    /**
     * isDigit() returns true iff its parameter is a digit
     * @param ch -- a char
     * @return a boolean true iff ch is a digit
     */
    public boolean isDigit (char ch) {
        if (ch >= '0' && ch <= '9')
            return true;
        else
            return false;
    } // isDigit()

    /**
     * convert() converts is digit parameter to a word
```

```
     *  for example '9' to "nine"
     * @digit a char giving a '0' to '9'
     * @return a String representing the digit
     */
    public String convert (char ch) {
        if (ch == '0')
            return "zero";
        else if (ch == '1')
            return "one";
        else if (ch == '2')
            return "two";
        else if (ch == '3')
            return "three";
        else if (ch == '4')
            return "four";
        else if (ch == '5')
            return "five";
        else if (ch == '6')
            return "six";
        else if (ch == '7')
            return "seven";
        else if (ch == '8')
            return "eight";
        else if (ch == '9')
            return "nine";
        else
            return "error";
    } // convert()

    /**
     * findAndChange() changes each digit in s to
     *  its corresponding number name: '7' to "seven"
     * @param s -- a String of text to be converted
     * @return a String giving the processed text
     */
    public String findAndChange(String s) {
        StringBuffer result = new StringBuffer();
        char ch1, ch2, ch3;

        ch2 = s.charAt(0); //Check the first char
        ch3 = s.charAt(1);
        if (isDigit(ch2)&&!isDigit(ch3))
            result.append(convert(ch2));
        else
            result.append(ch2);

        // Check all but the first and last chars.
        for (int k = 1; k < s.length() - 1; k++) {
            ch1 = s.charAt(k - 1);
            ch2 = s.charAt(k);
```

```
            ch3 = s.charAt(k + 1);
            if (!isDigit(ch1)&&isDigit(ch2)&&!isDigit(ch3))
                result.append(convert(ch2));
            else
                result.append(ch2);
        } // for
           //Check the last char
        ch1 = s.charAt(s.length() - 2);
        ch2 = s.charAt(s.length() - 1);
        if (!isDigit(ch1)&&isDigit(ch2))
            result.append(convert(ch2));
        else
            result.append(ch2);

        return result.toString();
    } // findAndChange()
} // DigitConverter

<!-- ***********************************************
file: index.html
************************************************
-->

<html>
<head><title>Digit Converter Applet</title></head>
<body>
<hr>
<applet code=DigitConverterApplet.class
                               width=600 height=400>
</applet>
<hr>
<a href="DigitConverterApplet.java">
                      DigitConverterApplet.java</a>
<br>
<a href="DigitConverter.java">DigitConverter.java</a>
</body>
</html>
```

12. A palindrome is a string that is spelled the same way backward and forward. For example, *mom, dad, radar, 727* and *able was i ere i saw elba* are all examples of palindromes. Write a Java applet that lets the user type in a word or phrase, and then determines whether the string is a palindrome.

    Answer:

```
/*
 * File: Palindrome.java
 * Author: Java, Java, Java
 * Description: This class contains methods to determine
 *  whether a string is a palindrome or not. A palindrome
```

```
 *  is a string that is equal to its reverse.
 */

import java.util.*;

public class Palindrome  {

    /**
     * reverse() reverses the characters in its parameter
     * @param s -- a String to be reversed
     * @return a String giving the reverse of s
     */
    private String reverse(String s) {
        StringBuffer result = new StringBuffer();
        for (int k = s.length() - 1; k >= 0; k--)
            result.append(s.charAt(k));
        return result.toString();
    }

    /**
     * isPalindrome() tests whether its string is a palindrome
     * @param s -- a String to be tested
     * @return a boolean set to true iff s is a palindrome
     */
    public boolean isPalindrome(String s) {
        return s.equals(reverse(s));
    } // isPalindrome()
} // Palindrome

/*
 * File: PalindromeApplet.java
 * Author: Java, Java, Java - Part of a solution to
 * Exercise 7.12
 * Description: This applet provides a user interface
 * for the Palindrome object, which determines whether
 * strings input into a JTextArea are palindromes.
 * The interface contains two JTextAreas, one for input
 * and one for output, and a button to start the test.
 */

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

public class PalindromeApplet extends JApplet
                           implements ActionListener {

    private JLabel prompt;          // The GUI elements
    private JTextArea inputArea;
    private JButton button;
```

```
    private JTextArea outputArea;

    /**
     * init() sets up the user interface
     */
    public void init(){
        Container pane = getContentPane();
        prompt = new JLabel("Enter a sentence to test:");
        inputArea = new JTextArea(5,40);
        inputArea.setEditable(true);
        outputArea = new JTextArea(5,40);
        button = new JButton("Test Palindrome");
        button.addActionListener(this);
        pane.add("North", prompt);
        pane.add("Center", inputArea);
        pane.add("East", button);
        pane.add("South", outputArea);
        setSize(600,400);
    } // init()

    /**
     * actionPerformed() tests the string in the input
     * TextArea to see if it is a palindrome and reports
     * the result.
     */
    public void actionPerformed(ActionEvent e){
        Palindrome pal = new Palindrome();
        String document = new String(inputArea.getText());
        if (pal.isPalindrome(document))
            outputArea.setText(document +
                               "  is a palindrome");
        else
            outputArea.setText(document +
                              "  is NOT a palindrome");
    } // actionPerformed()
} // PalindromeApplet.java

<!-- ***********************************************
file: index.html
************************************************
-->

<html>
<head><title>Palindrom Applet</title></head>
<body>
<hr>
<applet code=PalindromeApplet.class
                                  width=600 height=400>
</applet>
<hr>
```

```
<a href="PalindromeApplet.java">PalindromeApplet.java</a>
<br>
<a href="Palindrome.java">Palindrome.java</a>
</body>
</html>
```

13. Write a maze program that uses a string to store a representation of the maze. Write a method that accepts a `String` and prints a two-dimensional representation of a maze. For example, the maze shown here, where O marks the entrance and exit,can be generated from the following string:

```
String: XX_XXXXXXXX__XXX_XXXX_XX____XXX_XX_XX_XXX___X____XXXXXXXXXX
Maze:
         O
        XX XXXXXXX
        X  XXX XXX
        X XX    XX
        X XX XX XX
        X    X    O
        XXXXXXXXXX
```

Write a method that accepts such a string as a parameter and prints a two-dimensional representation of the maze.

Answer:

```
/*
 * File: Maze.java
 * Author: Java, Java, Java
 * Description: This class contains a method to
 *  convert a String such into a two-dimensional maze
 *  of asterisks.
 */

public class Maze {

    /**
     * convertToMaze() prints specially encoded string
     *  into a two-dimensional maze.
     * @param s -- a specially encoded string, such as
     *    XX_XXXXXXXX__XXX_XXXX_XX____
     */
    public void convertToMaze(String s){
        StringBuffer result = new StringBuffer("");
        for (int k = 0; k < s.length(); k++) {
           if (s.charAt(k) == 'X')
               result.append('X');
           else
               result.append(" ");
```

```
            if ( (k+1) % 10 == 0)
                result.append("\n");   // Start new line
        } // for
        System.out.println(result.toString());
    } // convertToMaze()

    public static void main(String argv[]) {
        Maze maze = new Maze();
        String s = "XX_XXXXXXXX__XXX_XXXX_XX____XXX_XX_XX_XXX____X____XXXXXXXXXX";
        System.out.println("String " + s);
        maze.convertToMaze(s);
    } // main()
} // Maze
```

14. Write a method that takes a delimited string, which stores a name and address, and prints a mailing label. For example, if the string contains "Sam Penn:14 Bridge St.:Hoboken:NJ:01881", the method should print

```
Sam Penn
14 Bridge St.
Hoboken, NJ 01881
```

Answer:

```
/*
 * File: Label.java
 * Author: Java, Java, Java
 * Description: This class contains a method to print a
 *  label from a String with delimiter ":".
 */
import java.util.StringTokenizer;

public class Label {
    /**
     * convertToAddress() converts a delimited string to
     *  a mailing label
     * @param s -- a string of the form
     *                          name:street:city,state zip
     */
    public static void convertToAddress(String s){
        StringTokenizer words = new StringTokenizer(s,":");
        while (words.hasMoreTokens()) {
            System.out.println(words.nextToken());
        } // while
    } // convertToAddress()

    /**
     * main() tests the convertToAddress() method. It
     * creates two delimited strings with addresses and
```

```
     * prints two labels
     * @param argv[] --not used in this program.
     */
    public static void main(String argv[]) {
        String str1 =
                "Sam Penn:14 Bridge St.:Hoboken,NJ 01881";
        String str2 =
                   "Ann Finn:93 Center St.:Avon,CT 06001";
        Label.convertToAddress(str1);
        System.out.println();
        Label.convertToAddress(str2);
    } // main()

} //Label class
```

15. Design and implement an applet that plays Time Bomb with the user. Here's how the game works. The computer picks a secret word and then prints one asterisk for each letter in the word: * * * * *. The user guesses at the letters in the word. For every correct guess, an asterisk is replaced by a letter: * e * * *. For every incorrect guess, the time bomb's fuse grows shorter. When the fuse disappears, after, say, six incorrect guesses, the bomb explodes. Store the secret words in a delimited string and invent your own representation for the time bomb.

    Answer:

```
/*
 * File: SecretWord.java
 * Author: Java, Java, Java
 * Description: This class manages secret words for
 *  the time bomb game. A collection of words is stored
 *  in a StringTokenizer object. Each time a new game is
 *  played the next word in the list is selected. As
 *  correct guesses are made, the guessedWord stores
 *  the current status of the guessed word. The word
 *  is guessed when the guessedWord equals the secretWord.
 */

import java.util.*;

public class SecretWord {
    private static final String WORDS =
                              "hello:java:program:applet";
    StringTokenizer wordList =
                          new StringTokenizer(WORDS,":");

    private String secretWord;   // The secret word
                // Word where correct guesses are shown
    private String guessedWord;
```

```
/**
 *  reset() selects a new secretWord from wordList
 */
public void reset() {
    if (wordList.hasMoreTokens())
        secretWord = wordList.nextToken();
    else
        secretWord = "last";
    StringBuffer result = new StringBuffer("");
    for (int k = 0; k < secretWord.length(); k++)
        result.append('*');
    guessedWord = result.toString();
} // reset()

/**
 * isGuessed() returns true iff the guessedWord
 *   equals the secretWord
 * @return the boolean true iff the secret word
 * is guessed
 */
public boolean isGuessed () {
    return secretWord.equals(guessedWord);
} // isDone()

/**
 * checkProgess() returns guessedWord which
 * shows the progress of the guessing game
 * @return a String giving the guessed word
 */
public String checkProgress () {
    return guessedWord;
} // checkProgess()

/**
 * getSecretWord() returns the secret word
 * @return a String giving the secret word
 */
public String getSecretWord() {
    return secretWord;
} // getSecretWord()

/**
 * makeGuess() checks whether its char parameter
 * is contained in the secret word
 * @return a boolean representing success or failure
 * Algorithm: If the secret word contains the
 * guessed letter every occurrence of the letter is
 * replaced in guessedWord,  giving a new guessedWord.
 * If not, guessedWord remains unchanged.
 */
```

```
    public boolean makeGuess(char ch){
        StringBuffer result = new StringBuffer("");
        boolean isRight = false;
                    // If secret word contains the char
                    //  replace every occurrence of it
                    //  in the guessed word
        if (secretWord.indexOf(ch) != -1)
            for (int k = 0;k < secretWord.length();k++){
                if (secretWord.charAt(k) == ch)
                    result.append(secretWord.charAt(k));
                else if (secretWord.charAt(k) != ch)
                    result.append(guessedWord.charAt(k));
                isRight = true;
            }
                  // Otherwise, guessedWord is unchanged
        else if (secretWord.indexOf(ch) == -1) {
            result.append(guessedWord);
            isRight = false;
        }
                                // Update guessedWord
        guessedWord = result.toString();
        return isRight;
     } // makeGuess()
} // SecretWord

/*
 * File: TimeBombApplet.java
 * Author: Java, Java, Java
 * Description: This program plays the game of
 *  "time bomb."  The user is given a secret word
 *  and tries to guess its letters. Each time a
 *  correct letter is guessed, its position in the
 *  word is displayed. The time bomb goes off after
 *  the user makes six wrong guesses.

 * In this design, the applet (this class) handles
 * the user interface and a separate object, a
 * SecretWord, handles the processing of the user's
 * guesses.
 */

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

public class TimeBombApplet extends JApplet
                            implements ActionListener {

    private JLabel prompt1, prompt2; // The GUI elements
    private JTextField inputField;
```

```
    private JTextArea outputArea;
    private String inputString;
    private int numberOfGuesses = 0;
        // SecretWord object
    SecretWord secret = new SecretWord();

    /**
     * init() sets up the applet's interface which consists
     * of a JTextfield in which the user inputs a guess, and
     * a JTextArea which displays  the results of each guess.
     * An ActionListener is assigned to the JTextField.
     */
    public void init() {
Container pane = getContentPane();
        prompt1 = new JLabel("Enter a letter you think is " +
                                        "in the secret word.");
        prompt2 = new JLabel("You have six guesses to " +
                                      "figure out the word.");
        inputField = new JTextField(30);
        inputField.setEditable(true);
        inputField.addActionListener(this);
        outputArea = new JTextArea(5,30);

        pane.add("North", prompt1);
        pane.add("West", prompt2);
        pane.add("Center", inputField);
        pane.add("South", outputArea);
        setSize(600,400);
        secret.reset();
        outputArea.append("The secret word: " +
                             secret.checkProgress() +"\n");
    } // init()

    /**
     * actionPerformed() handles action events in the
     * JTextField.
     */
    public void actionPerformed(ActionEvent e) {
        numberOfGuesses++;
        int guessesLeft = 6 - numberOfGuesses;
        inputString = inputField.getText();
        inputField.setText("");
        outputArea.setText("");
        boolean guess =
                  secret.makeGuess(inputString.charAt(0));

             // Process the user's guess
        if (guess == true)
            outputArea.append ("That Guess Was Right\n");
        else
```

```
            outputArea.append ("That Guess Was Wrong\n");

        if (secret.isGuessed()) {
            outputArea.append("Very good! You guessed " +
                              "the secret word! It was " +
                          secret.getSecretWord() + "\n");
            secret.reset();
            numberOfGuesses = 0;
            outputArea.append ("The new secret word is: "
                        + secret.checkProgress() + "\n");
        } else if (numberOfGuesses >= 6) {

            outputArea.append ("Sorry, that was your last "
                            + "guess. The secret word was "
                           + secret.getSecretWord() + "\n");
            secret.reset();
            numberOfGuesses = 0;
            outputArea.append ("The new secret word is: " +
                            secret.checkProgress() + "\n");
        } else {
            outputArea.append ("You have " + guessesLeft +
                                        " guesses left.\n");
            outputArea.append ("The new string is " +
                            secret.checkProgress() + "\n");
        } // else < 6 guesses
    } // actionPerformed()
} // TimeBombApplet

<!-- ***********************************************
file: index.html
************************************************
-->

<html>
<head><title>Time Bomb Applet</title></head>
<body>
<hr>
<applet code=TimeBombApplet.class width=600 height=400>
</applet>
<hr>
<a href="TimeBombApplet.java">TimeBombApplet.java</a>
<br>
<a href="TimeBomb.java">TimeBomb.java</a>
</body>
</html>
```

16. **Challenge:** A common string processing algorithm is the global replace function found in every word processor. Write a method that takes three `String` arguments: a document, a target string, and a replacement string. The method should replace every occurrence of the target string in the document with the

replacement string. For example, if the document is "To be or not to be, that is the question" and the target string is "be" and the replacement string is "see," the result should be "To see or not to see, that is the question."

Answer:

```
/*
 * File: GlobalReplace.java
 * Author: Java, Java, Java
 * Description: This class tests the replace() method.
 */

public class GlobalReplace {

    /**
     *  replace() replaces every occurrence of tar with rep
     *   in the String doc
     * @param doc -- a String to be processed
     * @param targ -- a target String to be replaced
     * @param repl -- a replacement String
     * @return - - the modified doc.
     * Algorithm: Use indexOf() to find the location of the
     * target substring. Then replace the original string
     * with the characters to the left of the target, the
     * replacement string, and the characters to the right
     * of the target. The loop exits when indexOf() returns
     * -1, indicating that there are no further occurrences
     * of the target in the string.
     */
    public String replace(String doc, String targ,
                                          String repl){
        int p = doc.indexOf(targ);
        int targLen = targ.length();
        while (p != -1) {         // While more occurrences
            doc = doc.substring(0,p) + repl +
                                 doc.substring(p+targLen);
            p = doc.indexOf(targ);
        }
        return doc;
    } // replace()

    public static void main(String argv[]) {
        GlobalReplace replacer = new GlobalReplace();
        String result =
               replacer.replace("Mississippi", "ss", "tt");
        System.out.println(result);
    } // main()

} // GlobalReplace
```

17. **Challenge:** Design and implement an applet that plays the following game with

the user. Let the user pick a letter between 'A' and 'Z'. Then let the computer guess the secret letter. For every guess the player has to tell the computer whether it's too high or too low. The computer should be able to guess the letter within five guesses. Do you see why?

Answer:

```
/*
 * File: Guesser.java
 * Author: Java, Java, Java
 * Description: This class handles the guessing for
 *  the guessing game applet. It uses a binary search
 *  to find the user's letter in the alphabet. So its
 *  first guess will by 'M'. If that is too high, it
 *  will search between 'A' and 'L' for its next guess.
 *  It continues to divide the alphabet in half after
 *  each incorrect guess. This will guarantee that the
 *  correct letter will be found within 5 guesses.
 *
 * Guesser's state contains a pointer (low) to the
 *  first letter in the current range of the alphabet
 *  and a pointer (high) to the last letter of the
 *  alphabet.
 */

public class Guesser {
    private char guess = 'm'; //The first guess is 'm'
    private char high = (char)('z');
    private char low = (char)('a');

    /**
     * getGuess() returns the current value of guess
     * @return a char representing the letter guessed
     */
    public char getGuess(){
        return guess;
    } // getGuess()

    /**
     * reset returns Guesser to its initial state
     * @return a char representing the letter guessed
     */
    public void reset(){
        guess = 'm';
        high = (char)('z');
        low = (char)('a');
    } // reset()

    /**
     * guessTooHigh() resets the pointer to the last
```

```
     * letter in the alphabet and sets up next guess.
     */
    public void guessTooHigh () {
        high = (char)(guess - 1);
        guess = (char)((high + low) / 2);
    } // guessTooHigh()

    /**
     * guessTooLow() resets the pointer to the first
     * letter in the alphabet and sets up next guess.
     */
    public void guessTooLow () {
        low = (char)(guess + 1);
        guess = (char)((low + high) / 2);
    } // guessTooLow()
} // Guesser

/*
 * File: GuessingApplet.java
 * Author: Java, Java, Java
 * Description: This applet plays a simple guessing
 * game with the user. It lets the user pick a letter
 * between 'A' and 'Z' and then it guesses the secret
 * within five guesses. The algorithm it uses splits
 * the alphabet in half and guesses the middle letter
 * each time. So the first guess will be 'M', and the
 * user will answer "too high" or "too low" or "just
 * right."
 */

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

public class GuessingApplet extends JApplet
                        implements ActionListener {
       // The GUI components
    private JLabel prompt1, prompt2;
    private JButton tooHigh;
    private JButton tooLow;
    private JButton start;
    private JButton correct;
    private JTextArea outputArea;
    private String inputString;
      // Guesser manages the guessing task
    private Guesser guesser = new Guesser();

    /**
     * init() sets up the user interface, which
```

```
 * contains three buttons that tell the program
 * whether the guess was high, low or correct.
 * It also contains a TextArea to report the
 * progress of the game.
 */
public void init() {
    Container pane = getContentPane();
    prompt1 = new JLabel("Think of a letter " +
                            "and click Start.");
    prompt2 = new JLabel("I will then try to " +
               "guess the letter you entered.");
    outputArea = new JTextArea(5,40);
    tooHigh = new JButton("Too High");
    tooHigh.addActionListener(this);
    tooLow = new JButton("Too Low");
    tooLow.addActionListener(this);
    start = new JButton("Start");
    start.addActionListener(this);
    correct = new JButton("Correct");
    correct.addActionListener(this);
       // Create prompt pane
JPanel promptPane = new JPanel();
    promptPane.setLayout(new BorderLayout());
       // Add labels
    promptPane.add("North", prompt1);
    promptPane.add("South", prompt2);
    promptPane.add("East",start);
    pane.add("North", promptPane);
    pane.add("Center", correct);
    pane.add("East", tooHigh);
    pane.add("West", tooLow);
    pane.add("South", outputArea);
    setSize(600,400);
} // init()

/**
 * actionPerformed() handles clicks on the applet's
 * buttons.  The guesser object manages the guessing
 * task. Each time getGuess() is called, it returns
 * a new guess. When  guessTooHigh() and guessTooLow()
 * are called, guesser adjusts the range of the alphabet
 * that it must search.
 * @param e -- the ActionEvent that generated this
 *   method call
 */
public void actionPerformed(ActionEvent e) {
    outputArea.setText("");
    if (e.getSource() == start) {
        guesser.reset();
        outputArea.append("My guess is " +
```

```
                                guesser.getGuess() + "\n");
            outputArea.append("Tell me if my guess is " +
                                    "too high or low.\n");
        }
        else if (e.getSource() == tooHigh) {
            guesser.guessTooHigh();
            outputArea.append("My guess is " +
                                guesser.getGuess() + "\n");
            outputArea.append("Tell me if my guess is " +
                                    "too high or low.\n");
        }
        else if (e.getSource() == tooLow) {
            guesser.guessTooLow();
            outputArea.append("My guess is " +
                                guesser.getGuess() + "\n");
            outputArea.append("Tell me if my guess is " +
                                    "too high or low.\n");
        }
        else if (e.getSource() == correct) {
            outputArea.append("Game over. Thanks for " +
                                            "playing.\n");
            outputArea.append("To play again, think of " +
                          "a letter and press 'start'\n");
        }
    } // actionPerformed()
} // GuessingApplet

<!-- ***********************************************
file: index.html
************************************************
-->

<html>
<head><title>Guessing Game Applet</title></head>
<body>
<hr>
<applet code=GuessingApplet.class width=400 height=400>
</applet>
<hr>
<a href="GuessingApplet.java">GuessingApplet.java</a>
<br>
<a href="Guesser.java">Guesser.java</a>
</body>
</html>
```

18. **Challenge:** A *list* is a sequential data structure. Design a `List` class which uses a comma-delimited `String` — such as, "a,b,c,d,12,dog" — to implement a list. Implement the following methods for this class:

```
  void addItem( Object o );  // Use Object.toString()
```

```
  String getItem(int position);
  String toString();
  void deleteItem(int position);
  void deleteItem(String item);
  int getPosition(String item);
  String getHead();      // First element
  List getTail();        // All but the first element
  int length();          // Number of items
```

Answer:

```
/*
 * File: List.java
 * Author: Java, Java, Java
 * Description: This class uses a comma,delimited
 * string to store strings in a list. It implements a
 * variety of standard list processing methods.
 */

import java.util.*;

public class List {
       // Start with an empty list
    private StringBuffer list = new StringBuffer();

    /**
     * List() constructor creates a list using s as
     * its initial value.
     * @param s -- a comma-delimited String such as
     *   "a,b,c,12,dog"
     */
    public List(String s) {
        list.append(s);
    } // List()

    /**
     * addItem() appends a String representation of
     * the Object o to the list
     * @param o -- a reference to an Object
     */
    public void addItem(Object o) {
        list.append(o.toString() + ",");
    } // addItem()

    /**
     * getItem() returns the item stored in the nth
     * position
     * @param n -- an int giving the item's position
     * @return a String representing the nth item
```

```
 */
public String getItem(int position) {
    StringTokenizer items =
      new StringTokenizer(list.toString(), ",");
    String item;
      // Skip the first n-1 items
    for (int k = 0; k < position-1 &&
                    items.hasMoreTokens(); k++)
        item = items.nextToken();
      // Return the nth
    if (items.hasMoreTokens())
        return items.nextToken();
    else
        return "";
} // getItem()

/**
 * deleteItem() deletes the nth item in the list
 * @param n -- an int giving the item's position
 */
public void deleteItem(int position) {
    StringTokenizer items =
       new StringTokenizer(list.toString(), ",");
    StringBuffer newList = new StringBuffer();
    while (items.hasMoreTokens()) {
        position--;           // Count this item
          // Copy all but the nth item
        if (position != 0)
           newList.append(items.nextToken() + ",");
        else
           items.nextToken();   // Ignore the nth
    }
    list = newList;            // Save the new list
} // deleteItem()

/**
 * deleteItem() deletes the item matching its
 * parameter
 * @param item -- a String
 */
public void deleteItem(String item) {
    StringTokenizer items =
      new StringTokenizer(list.toString(), ",");
    StringBuffer newList = new StringBuffer();
    String listItem;
    while (items.hasMoreTokens()) {
        listItem = items.nextToken();
          // If it doesn't match item
          //   copy it to the new list
        if (!listItem.equals(item))
```

```
            newList.append(listItem + ",");
        }
        list = newList;      // Save the new list
    } // deleteItem()

    /**
     * getPosition() finds item's position in the list
     * @param item -- a String
     */
    public int getPosition(String item) {
        StringTokenizer items =
          new StringTokenizer(list.toString(), ",");
        StringBuffer newList = new StringBuffer();
        String listItem;
        int count = 0;

        while (items.hasMoreTokens()) {
            listItem = items.nextToken();
            //If it matches item, return its position
            if (listItem.equals(item))
                return ++count;
            else
                ++count;
        }
        return count;
    } // getPosition()

    public String getHead()  {
        StringTokenizer items =
          new StringTokenizer(list.toString(), ",");
        return items.nextToken();
    } // getHead()

    public String getTail() {
        StringTokenizer items =
          new StringTokenizer(list.toString(), ",");
        StringBuffer buff = new StringBuffer();
        items.nextToken();    // Ignore first item
        while (items.hasMoreTokens())
            buff.append(items.nextToken() + ",");
        return buff.toString();
    } // getTail()

    public int length() {
        StringTokenizer items =
          new StringTokenizer(list.toString(), ",");
        return items.countTokens();
    } // length()

    public String toString() {
```

```
        return list.toString();
    } // toString()

    public static void main(String argv[]) {
        List list = new List("ralph,111-1111,amy,
              222-2222,sue,333-3333,joe,444-4444");
        System.out.println("Here's the list: " +
                                list.toString());
        System.out.println("It has "  +
                    list.length() + " elements");
        System.out.println("amy is in position "  +
                       list.getPosition("amy"));
        System.out.println("Its head is "  +
                                 list.getHead());
        System.out.println("Its tail is " +
                                 list.getTail());
        System.out.println("Its 3rd item is " +
                                list.getItem(3));
        System.out.println("Deleting the 3rd item ");
        list.deleteItem(3);
        System.out.println("Here's the list: " +
                                list.toString());
        System.out.println("It has " +
                    list.length() + " elements");
        System.out.println("Deleting amy's phone ");
        list.deleteItem("222-2222");
        System.out.println("Here's the list: " +
                                 list.toString());
        System.out.println("It has "  +
                     list.length() + " elements");
        System.out.println("Inserting amy and her " +
                                  "phone again ");
        list.addItem("amy");
        list.addItem("222-2222");
        System.out.println("Here's the list: " +
                                 list.toString());
        System.out.println("It has "  +
                    list.length() + " elements");
        System.out.println("Inserting an Integer ");
        list.addItem(new Integer(100));
        System.out.println("Here's the list: " +
                                 list.toString());
        System.out.println("It has " +
                    list.length() + " elements");
    } // main()

} // List
```

19. **Challenge:** Use a delimited string to create a `PhoneList` class with an instance method to insert names and phone numbers, and a method to look up a phone

number given a person's name. Since your class will take care of looking things up, you don't have to worry about keeping the list in alphabetical order. For example, the following string could be used as such a directory:

```
mom:860-192-9876::bill g:654-0987-1234::mary lancelot:123-842-1100
```

Answer:

```
/*
 * File: PhoneList.java
 * Author: Java, Java, Java
 * Description: This class implements a phone list
 *  database using a delimited string to store the
 *  individual entries. Entries take the following form:
 *   name1:number1::name2:number2::...
 */

import java.util.*;

public class PhoneList {
        // The delimited string
    private String phoneList = "";

    /**
     * isAnEntry() returns true iff its parameter is
     * an entry of the phone list
     * @param name -- a String giving a person's name
     * @return the boolean true iff the person's name
     *   is on the list
     */
    public boolean isAnEntry (String name) {
if ( name.equals("")) return false;
        if (phoneList.indexOf(name) != -1)
            return true;
        else
            return false;
    } // isAnEntry()

    /**
     * addEntry() inserts its parameter into phone list
     * @param entry -- a String of the form name:phone
     */
    public void addEntry(String entry) {
        StringBuffer result = new StringBuffer("");
        result.append(phoneList);
        result.append(entry + "::");
        phoneList = result.toString();
    } // addEntry()
```

```
/**
 * addEntry() returns the phone number associated with
 * the name given by s or returns  "" if no such entry
 * @param name -- a String a person's name
 * @return a String giving the person's phone number
 */
public String getNumber(String name){
    StringBuffer result = new StringBuffer("");
    int pos = getPosition(name);
    int numPos = phoneList.indexOf(":", pos);
    result.append(phoneList.substring(numPos + 1,
                  phoneList.indexOf("::", numPos)));
    return result.toString();
} // getNumber()

/**
 * removeEntry() remove the entry associated with the
 * name given by the parameter
 * @param name -- a String giving a person's name
 * Algorithm: Suppose you're removing name from the
 * string
 *     ....::name:phone::...
 *           ^         ^
 *         pos1       pos2
 * Then pos1 gives the position where the name entry
 * starts and pos2 gives the entry ends. So you
 * replace the list with the concatenation of the
 * following two substrings: phoneList.substring(0,pos1)
 * and phoneList.substring(pos2 +2).
 */
public void removeEntry(String name) {
    StringBuffer result = new StringBuffer("");
    int pos1 = getPosition(name);
    int pos2 = phoneList.indexOf("::", pos1);
    result.append(phoneList.substring(0, pos1));
    result.append(phoneList.substring(pos2 + 2));
    phoneList = result.toString();
} // removeEntry()

/**
 * getPosition() returns the string index of name
 * @param name -- a String giving a person's name
 */
public int getPosition(String name) {
    return phoneList.indexOf(name);
} // getPosition()

/**
 * toString() returns the entire phone list
 */
```

```
    public String toString() {
        return phoneList;
    }

} // PhoneList

/*
 * File: PhoneListApplet.java
 * Author: Java, Java, Java
 * Description: This applet provides a user interface
 * to a phonelist. It provides buttons that let the
 * user add, remove, and lookup entries in a phone list.
 */

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

public class PhoneListApplet extends JApplet
                         implements ActionListener {

    private JLabel prompt1;           // GUI elements
    private JLabel prompt2;
    private JLabel prompt3;
    private JButton findNumber;
    private JButton addEntry;
    private JButton removeEntry;
    private JTextField inputField;
    private JTextArea outputArea;
      // The phone list
    PhoneList phonelist = new PhoneList();


    /**
     * init() creates the user interface. Three buttons
     * are assigned ActionListeners.
     */
    public void init() {
        Container pane = getContentPane();
        prompt1 = new
          JLabel("Add Entry: enter name:phone and press Add");
        prompt2 = new
          JLabel("Remove Entry :  enter name and press Remove");
        prompt3 = new
          JLabel("Lookup:  enter name and press Lookup");
        inputField = new JTextField(40);
        inputField.setEditable(true);
        outputArea = new JTextArea(5,40);
        findNumber = new JButton("Lookup");
        findNumber.addActionListener(this);
```

```
    addEntry = new JButton("Add");
    addEntry.addActionListener(this);
    removeEntry = new JButton("Remove");
    removeEntry.addActionListener(this);
    JPanel addPanel = new JPanel();
    addPanel.add(prompt1);
    addPanel.add(addEntry);
    JPanel removePanel = new JPanel();
    removePanel.add(prompt2);
    removePanel.add(removeEntry);
    JPanel lookupPanel = new JPanel();
    lookupPanel.add(prompt3);
    lookupPanel.add(findNumber);
    pane.add("Center", inputField);
    pane.add("North", lookupPanel);
    pane.add("West", addPanel);
    pane.add("East", removePanel);
    pane.add("South", outputArea);
    setSize(900,400);
} // init()


/**
 * actionPerformed() handles ActionEvents on the three
 * buttons. In each case it gets data from inputField
 * and invokes methods of the phonelist object.
 */
public void actionPerformed(ActionEvent e){
    if (e.getSource() == findNumber) {
        String name = inputField.getText();
        inputField.setText("");
        if (phonelist.isAnEntry(name)) {
            String num = phonelist.getNumber(name);
            outputArea.setText(name +
                        "'s phone number is " + num);
        } else {
            outputArea.setText(name +
                          " does not have an entry");
        }
    }
    if (e.getSource() == addEntry) {
        String entry = inputField.getText();
        inputField.setText("");
        phonelist.addEntry(entry);
        outputArea.setText(entry +
                    " has been added to your list");
    }
    if (e.getSource() == removeEntry) {
        String name = inputField.getText();
        inputField.setText("");
```

```
            if (phonelist.isAnEntry(name)) {
                phonelist.removeEntry(name);
                outputArea.setText(name +
                                   "'s entry is removed");
            } else {
                outputArea.setText(name +
                             " does not have an entry");
            }
        }
        outputArea.append("\nList: " +
                          phonelist.toString() + "\n");
    } // actionPerformed()
} // PhoneListApplet

<!-- ***********************************************
file: index.html
************************************************
-->

<html>
<head><title>PhoneList Applet</title></head>
<body>
<hr>
<applet code=PhoneListApplet.class width=900 height=400>
</applet>
<hr>
<a href="PhoneListApplet.java">PhoneListApplet.java</a>
<br>
<a href="PhoneList.java">PhoneList.java</a>
</body>
</html>
```

20. Design and implement an applet or application that displays a multiline message in various fonts and sizes input by the user. Let the user choose from among a fixed selection of fonts, sizes, and styles.

    Answer: An application solution is provided by the `ChooseFontPanel` and `ChooseFontFrame` classes given below. An applet solution can be created by attaching the `ChooseFontPanel` to the `ContentPane` of a `JApplet` object as was described in chapter 4.

```
/*
 * File: ChooseFontPanel.java
 * Author: Java Java Java  - Part of a solution to
 * exercise 7.20.
 * Description: This class defines a GUI in a JPanel
 * which allows the user to choose the font, style,
 * and size for text that is displayed in a text area.
 * The interface contains a JTextArea which contains
 * instructions displayed in a font, style, and size
```

```
 * which can be chosen by the user. The interface also
 * contains a JTextField where the user can indicate
 * choices and a JButton to cause the JTextArea to be
 * rewritten in the chosen font, style, and size.
 */

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

public class ChooseFontPanel extends JPanel
                         implements ActionListener{
    private JTextArea instructArea; // Prompt for input.
    private JTextField choiceField; // For the choices.
    private JButton setButton;      // Push to set font.
       // For instructions displayed in instructArea
    private String text;
    private String fontName = "Times"; //Choice of font
    private int styleNum = Font.PLAIN;
    private int sizeNum = 9;

    public ChooseFontPanel() {
        buildGUI();
    } // ChooseFontPanel()

    private void buildGUI() {
        text =
         "Choose from following fonts, styles, and sizes:\n";
        text = text + "Font: 1) Times 2) Courier 3) Helvetica\n";
        text = text + "Style: A) Plain B) Italic C) Bold\n";
        text = text + "Size: a) 9   b) 12   c) 14\n";
        text = text +
                "Put your 3 character choice (like: 2Ba )\n";
        text = text +
                "in the textfield and then click the button";
        instructArea = new JTextArea(8,30);
        instructArea.setFont(new Font("Times", Font.PLAIN, 9));
        instructArea.setText(text);
        instructArea.setEditable(false);
        choiceField = new JTextField(3);
        choiceField.setEditable(true);
        choiceField.setText("1Aa");
        setButton = new JButton("Change Font");
        setButton.addActionListener(this);
        setLayout(new BorderLayout());
        add("North", choiceField);
        add("Center", instructArea);
        add("South", setButton);
    } //buildGUI()
```

```
    /**
     * actionPerformed() handles clicks on setButton.
     * It inputs the data from the JTextField, converts
     * it to values for a font, style, and size.  The
     * text string is then redisplayed in instruct Area
     * in the new font, style and size.
     */
    public void actionPerformed(ActionEvent e) {
        String choice = choiceField.getText();
        char ch; // To store the chars in choice.

        if (choice.length() >= 1) { // Determine font name
            ch = choice.charAt(0);
            if (ch == '2') fontName = "Courier";
            else if (ch == '3') fontName = "Helvetica";
            else fontName = "Times"; // ch == '1' or error
        } else fontName = "Times"; // Default if choice == ""

        if (choice.length() >= 2) { // Determine font style
            ch = choice.charAt(1);
            if (ch == 'B') styleNum = Font.ITALIC;
            else if (ch == 'C') styleNum = Font.BOLD;
            else styleNum = Font.PLAIN; // ch == 'A' or error
        } else styleNum = Font.PLAIN; //If choice.length() <= 1

        if (choice.length() >= 3) { // Determine font size
            ch = choice.charAt(2);
            if (ch == 'b') sizeNum = 12;
            else if (ch == 'c') sizeNum = 14;
            else sizeNum = 9; // ch == 'a' or error
        } else sizeNum = 9; // If choice.length() <= 2

        instructArea.setFont(new Font(fontName,
                                      styleNum, sizeNum));
    } // actionPeformed()

} // ChooseFontPanel class

/*
 * File: ChooseFontFrame.java
 * Author: Java Java Java
 * Description: This program creates a ChooseFontPanel
 * and adds it to the Frame's content pane.
 */

import javax.swing.*;

public class ChooseFontFrame extends JFrame {
    public ChooseFontFrame()
    {
```

```
            getContentPane().add(new ChooseFontPanel());
        } //ChooseFontFrame() constructor

        public static void main(String args[]){
            ChooseFontFrame gframe = new ChooseFontFrame();
            gframe.setSize(400,400);
            gframe.setVisible(true);
        } // main()

    } // ChooseFontFrame class
```

Chapter 8

Inheritance and Polymorphism

1. Fill in the blanks in each of the following sentences:

   (a) A method that lacks a body is an _____ method. **Answer:** abstract

   (b) An _____ is like a class except that it contains only instance methods, no instance variables. **Answer:** interface

   (c) Two ways for a class to inherit something in Java are to _____ a class and _____ an interface. **Answer:** extend, implement

   (d) Instance variables and instance methods that are declared _____ or _____ are inherited by the subclasses. **Answer:** protected, public

   (e) An object can refer to itself by using the _____ keyword. **Answer:** this

   (f) If an applet intends to handle ActionEvents, it must implement _____ the interface. **Answer:** `ActionListener`

   (g) A _____ method is one that does different things depending upon the object that invokes it. **Answer:** polymorphic

2. Explain the difference between the following pairs of concepts:

   (a) *Class* and *interface*.

   Answer: A Class can define variables and define abstract or implemented methods. An interface can only define final variables and abstract methods.

   (b) *Stub* method and *abstract* method.

   Answer: A stub method is a partially implemented method which can be called while developing other methods. A programmer can finish implementing the stub method at a later point. An abstract method is an unimplemented method. A class extending a class with an abstract method can implement it. A class that implements an interface with an abstract method can implement the method. Classes with abstract methods cannot be instantiated.

(c) *Extending* a class and *instantiating* an object.

Answer: A class extends another class by using the `extends` keyword in the class declaration. The new class inherits the variables and methods of the the class extended. The `new` keyword can be used to instantiate an object of that belongs to a class. The object can use the public variables and public methods of the class.

(d) *Defining* a method and *implementing* a method.

Answer: A method is defined or declared in a class if either its header and body are given or if only the header of an abstract method is given. A method is implemented in a class its header and body are given in the class for either a new method or for a method that has been defined in a super class of the class.

(e) A *protected* method and a *public* method.

Answer: A public method of a class can be called in any other class. A protected method can be called only in subclasses of the class in which it is declared.

(f) A *protected* method and a *private* method.

Answer: A private method of a class can be called only in that class. A protected method can be called only in subclasses of the class in which it is declared.

3. Draw a hierarchy to represent the following situation. There are lots of languages in the world. English, French, Chinese, and Korean are examples of natural languages. Java, C, and C++ are examples of formal languages. French and Italian are considered romance languages, while Greek and Latin are considered classical languages.

   Answer: The hierarchy chart below is drawn left to right instead of top to bottom.

```
Languages
    |
    |-- Formal Languages
    |    |---- C++
    |    |----  Fortran
    |    |----  Java
    |
    |-- Natural Languages
         |
         |-- English
         |-- Japanese
         |-- Korean
         |
         |-- Classical Languages
         |    |-- Latin
         |    |-- Greek
         |
```

```
        |-- Romance Languages
             |-- French
             |-- Italian
```

4. Look up the documentation for the `JButton` class on Suns Web site:

   `http://java.sun.com/j2se/docs/`

   List the names of all the methods that would be inherited by the `ToggleButton` subclass that we defined in this chapter.

   Answer: The methods inherited by the `ToggleButton` class that are defined in the `JButton` class are: `JButton(), configurePropertiesFromAction(), getAccessibleContext(), getUIClassID(), isDefaultButton(), isDefaultCapable(), paramString(), removeNotify(), setDefaultCapable(), updateUI()`.

   In addition, the `ToggleButton` class inherits around 400 methods from the super classes of `JButton` namely: `AbstractButton, JComponent, Container, Component`, and `Object`. This list of methods includes the many of the small number of frequently used methods of `JButton`. The list of inherited methods is too long to list here.

5. Design and write a `toString()` method for the `ToggleButton` class defined in this chapter. The `toString()` method should return the `ToggleButtons` current label.

   Answer:

```
/**
 * toString() overrides the toString() method of JButton class.
 @returns the current button label of the ToggleButton variable
 * which is stored in the instance variable label1.
 */
public String toString(){
    return label1;
} // toString()
```

6. Design a class hierarchy rooted in the class `Employee` that includes subclasses for `HourlyEmployee` and `SalaryEmployee`. The attributes shared in common by these classes include the name and job title of the employee, plus the assessor and mutator methods needed by the attributes. The salaried employees need an attribute for weekly salary, and the corresponding methods for accessing and changing this variable. The hourly employees should have a pay rate and an hours-worked variable. There should be an abstract method called `calculateWeeklyPay()`, defined abstractly in the superclass and implemented in the subclasses. The salaried workers pay is just the weekly salary. Pay for an hourly employee is simply hours worked times pay rate.

   Answer: Source code for the `Employee`, `HourlyEmployee`, and `SalaryEmployee` classes is listed below.

```
   /*
 * File: Employee.java
 * Author: Java, Java, Java
 * Description: This abstract class contains instance variables for
 * the name and job title of an employee.  It contains an abstract
 * method for calculating weekly pay.
 */
public abstract class Employee {
     private String name;
     private String jobTitle;

    /**
     * Employee() creates a new employee from a pair of string
     * parameters.
     * @param aName is a String  denoting the name of the employee.
     * @param title is a String  denoting the job title of the
     *   employee.
     */
     public Employee (String aName, String title) {
         name = aName;
         jobTitle = title;
     } // Employee() constructor

    /**
     * setName() is a mutator method for the name variable
     * @param aName is a String denoting the name of an employee.
     */
    public void setName(String aName) {
       name = aName;
    } // setName()

    /**
     * getName() is an accessor method for the name variable
     * @return is name, a String denoting the name of an employee.
     */
    public String getName() {
       return name;
    } // getName()

    /**
     * setTitle() is a mutator method for the jobTitle variable
     * @param title is a String denoting the name of an employee.
     */
    public void setTitle(String title) {
       jobTitle = title;
    } // setTitle()

    /**
     * getTitle() is an accessor method for the jobTitle variable
     * @return is jobTitle, a String denoting the job title of an
```

```
     *   employee.
     */
    public String getTitle() {
       return jobTitle;
    } // getTitle()

    /**
     * toString() returns the employee as a String
     * return is the job title of the employee followed by his/her
     *  name.
     */
    public String toString() {
        return jobTitle + " " + name;
    } // toString()

    /**
     * calculateWeeklyPay() returns the amount of the employee's
     * weekly pay. This is an abstract method which must be
     * implemented by subclasses.
     * @return is a double denoting the weekly pay in dollars.
     */
    public abstract double calculateWeeklyPay();

} // Employee class

/*
 * File: HourlyEmployee.java
 * Author: Java, Java, Java
 * Description: This class extends Employee and contains instance
 * variables for the pay rate and hours worked for the employee.
 * It implements the calculateWeeklyPay() method for calculating
 * weekly pay.
 */
public class HourlyEmployee extends Employee{
     private double payRate;
     private double hours;

    /**
     * HourlyEmployee() creates a new employee from a pair of
     *   string parameters.
     * @param aName is a String  denoting the name of the employee.
     * @param title is a String  denoting the job title of the
     *   employee.
     */
     public HourlyEmployee (String aName, String title) {
         super(aName, title);
     } // HourlyEmployee() constructor

    /**
     * setRate() is a mutator method for the payRate variable
```

```
 * @param rate is a double representing the hourly pay rate
 *   of the employee.
 */
public void setRate(double rate) {
   payRate = rate;
} // setRate()

/**
 * getRate() is an accessor method for the payRtae variable
 * @return is payRate, a double denoting the hourly pay rate
 *  of the employee.
 */
public double getRate() {
   return payRate;
} // getRate()

/**
 * setHours() is a mutator method for the hours variable
 * @param theHours is a double denoting the number of hours
 *  the employeeworked the previous week.
 */
public void setHours(double theHours) {
   hours = theHours;
} // setHours()

/**
 * getHours() is an accessor method for the hours variable
 * @return is a double denoting the hours worked by the
 *  employee during the previous week.
 */
public double getHours() {
   return hours;
} // getHours()

/**
 * calculateWeeklyPay() returns the amount of the employee's
 * weekly pay. This method is an abstract method of
 * Employee which is implemented here by returning the
 * product of payRate times hours.
 * @return is a double denoting the weekly pay in dollars.
 */
public double calculateWeeklyPay(){
           return payRate*hours;
} // calculateWeeklyPay()

/**
 * main() tests the HourlyEmployee class
*/
public static void main(String args[])  {
    HourlyEmployee empl =
```

```
            new HourlyEmployee("Marsha Smith", "Programmer");
          empl.setRate(16.70);
          empl.setHours(38.5);
          System.out.println("The employee is " + empl.toString());
          System.out.println("  whose salary last week was " +
                                      empl.calculateWeeklyPay());
      } //main()

} // HourlyEmployee class

/*
 * File: SalaryEmployee.java
 * Author: Java, Java, Java
 * Description: This class extends Employee and contains an
 * instance variable for the weekly salary for the employee.
 * It implements the calculateWeeklyPay() method for calculating
 * weekly pay.
 */
public class SalaryEmployee extends Employee{
     private double salary;

    /**
     * SalaryEmployee() creates a new employee from a pair of string
     * parameters.
     * @param aName is a String  denoting the name of the employee.
     * @param title is a String  denoting the job title of the
     *   employee.
     */
     public SalaryEmployee (String aName, String title) {
         super(aName, title);
     } // SalaryEmployee() constructor

    /**
     * setSalary() is a mutator method for the salary variable
     * @param theSalary is a double representing the weekly salary
     *   of the employee.
     */
    public void setSalary(double theSalary) {
       salary = theSalary;
    } // setSalary()

    /**
     * getSalary() is an accessor method for the salary variable
     * @return is salary, a double denoting the weekly salary
     *  of the employee.
     */
    public double getSalary() {
       return salary;
    } // getSalary()
```

```
    /**
     * calculateWeeklyPay() returns the amount of the employee's
     * weekly pay. This method is an abstract method of Employee
     * which is implemented here by returning the salary variable.
     * @return is a double denoting the weekly pay in dollars.
     */
    public double calculateWeeklyPay(){
               return salary;
    } // calculateWeeklyPay()

    /**
     * main() tests the SalaryEmployee class
    */
    public static void main(String args[])  {
        SalaryEmployee empl =
                new SalaryEmployee("William Jones", "Analyst");
        empl.setSalary(439.00);
        System.out.println("The employee is " + empl.toString());
        System.out.println("  whose salary last week was " +
                                    empl.calculateWeeklyPay());
    } //main()

} // SalaryEmployee class
```

7. Design and write a subclass of `JTextField` called `IntegerField` that is used for inputting integers but behaves in all other respects like a `JTextField`. Give the subclass a public method called `getInteger()`.

   Answer: The `TestIntegerFieldFrame` class below is a GUI application that tests the given implementation of the `IntergerField` class.

```
/*
 * File: IntegerField.java
 * Author: Java, Java, Java
 * Description: This class extends JTextField and contains an
 * instance method getInteger() which returns the the contents
 * as an integer.
 */

import javax.swing.*;

public class IntegerField extends JTextField{

    /**
     * IntegerField() - The default constructor.
     */
     public IntegerField() {
         super();
     } // IntegerField() default constructor
```

```
    /**
     * IntegerField() - A constructor with a number of columns parameter.
     * @param cols denotes the number of columns of the JTextfield.
     */
     public IntegerField(int cols) {
         super(cols);
     } // IntegerField(cols) constructor

     /**
     * IntegerField() - the text constructor.
     * @param text is a String denoting the contents of the JTextfield.
     */
     public IntegerField(String text) {
         super(text);
     } // IntegerField(text) constructor

     /**
     * IntegerField() - the text and number of columns constructor.
     * @param text is a String denoting the contents of the JTextfield.
     * @param cols denotes the number of columns of the JTextfield.
     */
     public IntegerField(String text, int cols) {
         super();
     } // IntegerField(text, cols) constructor

     /**
     * getInteger() attempts to parse the contents of the JTextfield as
     * an int and returns that value. It returns -1 if the contents
     * cannot be parsed into an integer.
     */
    public int getInteger() {
        try {
            String str = getText();
            int num = Integer.parseInt(str);
            return num;
        }
        catch(Exception e) {
            return -1;
        } // catch??
    } // getInteger()

} // IntegerField class

/*
 * File: TestIntegerFieldFrame.java
 * Author: Java Java Java
 * Description: This program adds an IntegerField and
 *  JTextArea to the ContentPane of a JFrame.
 */
import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;

public class TestIntegerFieldFrame extends JFrame
                              implements ActionListener
{
    private JTextArea display;
    private IntegerField inField;
    private JButton testButton;

    public TestIntegerFieldFrame()
    {
         buildGUI();
    } // TestIntegerFieldFrame()

    private void buildGUI()
    {   Container pane = getContentPane();
display = new JTextArea(10,30);
        inField = new IntegerField("input an integer", 15);
        testButton = new JButton("Click for a test");
        testButton.addActionListener(this);
        pane.add("North", inField);
        pane.add("Center", testButton);
        pane.add("South", display);
    } //buildGUI()

    public void actionPerformed(ActionEvent e)
    {   if (e.getSource() == testButton)
        {   int num = inField.getInteger();
            display.append("num = " + num + "\n");
        } // if
    } // actionPeformed()

    public static void main(String args[]){
        TestIntegerFieldFrame gframe =
                            new TestIntegerFieldFrame();
        gframe.setSize(400,400);
        gframe.setVisible(true);
    } // main()

} // TestInteggerFieldFrame class
```

8. Implement a method that uses the following variation of the Caesar cipher. The method should take two parameters, a `String` and an int N. The result should be a `String` in which the first letter is shifted by N, the second by N + 1 , the third by N + 2 , and so on. For example, given the string ”Hello”, and an initial shift of 1, your method should return ”Igopt”. Write a method that converts its `String` parameter so that letters are written in blocks five characters long.

   Answer: The following class contains methods for encrypting and decrypting

strings using the above prescribed algorithm and a method for formatting strings into blocks of 5 letters.

```
/*
 * File: CaesarCipherVariation.java
 * Author: Java, Java, Java
 * Description: This class defines methods for encrypting and
 * decrypting strings with a starting shift for the first letter.
 * Each successive letter is shifted one additional position.
 * The class also contains a method that regroups letters into
 * blocks of 5 letters with a space between them and a main method
 * to test the other three methods.
 */

public class CaesarCipherVariation{

    /**
     * encryptCaesarVar(str, num) encrypts the letters of str
     * with a shift of num for the first letter.  Each successive
     * letter is shifted one additional position modulo 26. The
     * method is static because the class has no instance variables.
     * @param str is any string to be encrypted.
     * @param num is an int between 0 and 25 which denotes the shift
     *   for encrypting the first letter of str.
     * @return is the encryption of str.
     */
    public static String encryptCaesarVar(String str, int num) {
        String strcaps = str.toUpperCase();
        int shift = num;
        if (shift < 0) shift = -shift;  // If negative shift is given.
        StringBuffer sb = new StringBuffer();
        char ch;
        for (int k = 0; k < strcaps.length(); k++) {
            ch = strcaps.charAt(k);
            if ((ch >= 'A') && (ch <= 'Z')){
                sb.append((char)('A' + (ch - 'A' + shift) % 26));
                shift = (shift + 1) % 26;  // add 1 modulo 26
            } // if
            else   // If not a letter, just copy it.
                sb.append(ch);
        } // for
        return sb.toString();
    } // encrypytCaesarVar()

    /**
     * decryptCaesarVar(str, num) encrypts the letters of str
     * with a shift of num for the first letter.  Each successive
     * letter is shifted one additional position modulo 26.
     * @param str is any string to be decrypted.
     * @param num is an int between 0 and 25 which denotes the shift
```

```
 *   used for encrypting the first letter of str.
 * @return is the decryption of str.
 */
public static String decryptCaesarVar(String str, int num) {
    String strcaps = str.toUpperCase();
    int shift = 26 -num;
    if (shift < 0) shift = -shift;
    StringBuffer sb = new StringBuffer();
    char ch;
    for (int k = 0; k < strcaps.length(); k++) {
        ch = strcaps.charAt(k);
        if ((ch >= 'A') && (ch <= 'Z')){
            sb.append((char)('A' + (ch - 'A' + shift) % 26));
            shift = (shift + 25) % 26;
        } // if
        else
            sb.append(ch);
    } // for
    return sb.toString();
} // decrypytCaesarVar()

/**
 * fiveBlocker(str) eliminates characters that are not letters
 * in str and inserts a space after each block of five letters.
 * @param str is any string to be put into blocks of five.
 * @return is the blocked form of str.
 */
public static String fiveBlocker(String str) {
    String strcaps = str.toUpperCase();
    int numOfLetters = 0;
    StringBuffer sb = new StringBuffer();
    char ch;
    for (int k = 0; k < strcaps.length(); k++) {
        ch = strcaps.charAt(k);
        if ((ch >= 'A') && (ch <= 'Z')){
            sb.append(ch);
            numOfLetters++;
            if (numOfLetters % 5 == 0)
                 sb.append(' ');
        } // if
    } // for
    return sb.toString();
} // fiveBlocker()

public static void main(String args[]){
    String str1 =
            "Hello young programmers where ever you are!";
    System.out.println(str1);
    String str2 =
           CaesarCipherVariation.encryptCaesarVar(str1,1);
```

```
        System.out.println(str2);
        String str3 =
                     CaesarCipherVariation.fiveBlocker(str2);
        System.out.println(str3);
        String str4 =
               CaesarCipherVariation.decryptCaesarVar(str3,1);
        System.out.println(str4);
    } // main()

} // CaesarCipherVariation class
```

9. Design and implement an applet that lets the user type a document into a `JTextArea` and then provides the following analysis of the document: the number of words in the document, the number of characters in the document, and the percentage of words that have more than six letters.

   Answer: A solution is provided by the `AnalyzeText` and `AnalyzeTextApplet` classes. An `index.html` file for testing the applet is also included.

```
/*
 * File: AnalyzeText.java
 * Author: Java, Java, Java
 * Description: This class defines methods for an analysis of
 * a text string.  The number of characters in the string, the
 * number of words in the string, and the percentage of words
 * with more than 6 letter can be computed and reported.
 */

public class AnalyzeText{
    private String text;
    private int charCount = 0;
    private int wordCount = 0;
    private int longWordCount = 0;
    private double percentLongWords = 0.0;

     /**
     * AnalyzeText(str) is a constructor that assigns str as the
     *   string to be analyzed.
     * @param str is any string to be encrypted.
     */
    public  AnalyzeText(String str) {
        text = str;
    } // AnalyzeText() constructor


    /**
     * doAnalysis() does the prescribed calculations.  Words are
     * defined as any sequence of 1 or more letters. The number of
     * characters, number of words, and percentage of words with
     * more than 6 letters are calculated.
```

```
 */
public void doAnalysis() {
    String capText = text.toUpperCase();
      // All letters are now uppercase
      // Make certain variables have initial values.
    charCount = 0;
    wordCount = 0;
    longWordCount = 0;
    int currLettersInWord = 0; //For calculating the word count.
    char ch;                   // To store each text character.
    for (int k = 0; k < capText.length(); k++) {
        ch = capText.charAt(k);
        charCount++;
        if ((ch >= 'A') && (ch <= 'Z')){
            currLettersInWord++;  // since you are in a word
        } // if
        else { //If not a letter, decide if you finished a word.
          if (currLettersInWord > 0) { // just finished a word
              wordCount++;
              if (currLettersInWord > 6)
                  longWordCount++;  //just finished a long word
              currLettersInWord = 0;  // Reset for next word
          } // if currLettersInWord > 0
        } // else
    } // for
    // Check if the last letter finished a word
    if (currLettersInWord > 0) { // just finished a word
          wordCount++;
          if (currLettersInWord > 6)
              longWordCount++; // just finished a long word
          currLettersInWord = 0;  // Reset for no good reason
     } // if currLettersInWord > 0
    //  Compute percentage of long words
    percentLongWords = 100.0*longWordCount/wordCount;
} // doAnalysis()

/**
 * ReportAnalysis() returns a string reporting the number
 * of characters, number of words, and percentage of words
 * with more than 6 letters each on its own line.
 * @return is the report string.
 */
public String reportAnalysis() {
    StringBuffer sb = new StringBuffer();
    sb.append("There are "+charCount+" characters in the text.\n");
    sb.append("There are "+wordCount+" words in the text.\n");
    sb.append(percentLongWords+" percent of the words are long.");
    return sb.toString();
} // reportAnalysis()
```

```
    /**
     * main() just tests the methods
     * @parm args is not used in this main() method.
     */
    public static void main(String args[]){
        String str = "Hello young programmers where ever you are";
        System.out.println("THE TEXT IS:\n" + str);
        AnalyzeText at = new AnalyzeText(str);
        at.doAnalysis();
        System.out.println(at.reportAnalysis());
    } // main()

} // AnalyzeText class

/*
 * File: AnalyzeTextApplet.java
 * Author: Java Java Java - This applet has a prompt JLabel,
 *  an input JTextArea, an output JTextArea, and a JButton
 *  to initiate the prescribed analysis of the input text.
 * The AnalyzeText class is used to do the calculations
 * and prepare a report string.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class AnalyzeTextApplet extends JApplet
             implements ActionListener{
    // Declare instance variables
    private JLabel prompt;
    private JTextArea inputArea;
    private JButton button;
    private JTextArea outputArea;

    /**
     * init() creates the GUI componenets and adds them
     * to the content pane of the applet.
     */
   public void init()
    {        //Instantiate instance variables
        prompt =
          new JLabel("Input text in top area and click button");
        button = new JButton("Do Analysis");
        button.addActionListener(this);
        inputArea = new JTextArea(6,30);
        outputArea = new JTextArea(4,30);
            // Add the components to the applet content pane.
        Container pane = getContentPane();
        pane.add("North", prompt);
        pane.add("Center", inputArea);
```

```
        pane.add("East", button);
        pane.add("South", outputArea);
    } // init()

    /**
     * actionPerformed() handles clicks on the button.
     * @param e -- the ActionEvent the generated this system call
     */
    public void actionPerformed(ActionEvent e)
    {
        if (e.getSource() == button){
            String str = inputArea.getText();
            AnalyzeText at = new AnalyzeText(str);
            at.doAnalysis();
            outputArea.setText(at.reportAnalysis());
        } // if
     } // actionPeformed()

} // AnalyzeTextApplet class

-- File: index.html to test the AnalyzeTextApplet class --
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type"
                    CONTENT="text/html; charset=windows-1252">
<TITLE>
Analyze Text Applet

</TITLE>
</HEAD>
<BODY>
AnalyzeTextApplet will appear below in a Java enabled browser.<BR>
<APPLET
  CODEBASE = "."
  CODE     = "AnalyzeTextApplet.class"
  NAME     = "Analyze Text Applet"
  WIDTH    = 600
  HEIGHT   = 400
  HSPACE   = 0
  VSPACE   = 0
  ALIGN    = middle
>
</APPLET>
</BODY>
</HTML>
```

10. Design and implement a `Cipher` subclass to implement the following substitution cipher: Each letter in the alphabet is replaced with a letter from the opposite end of the alphabet: a is replaced with z, b with y, and so forth.

Answer: The `Reflect` class below requires access to the `Cipher` class from chapter 8 in order to compile and run.

```
/**
 * File: Reflect.java
 * Author: Java, Java, Java
 * Description: Reflect extends the Cipher class. The encode()
 * and decode() methods replace each letter between 'a' and 'z' in
 * a string with a letter reflected to a letter an equal distance
 * from the other end of the alphabet.
 */

public class Reflect extends Cipher{

    /**
    * encode() replace each letter between 'a' and 'z' in
    * word with a letter reflected to a letter an equal distance
    * from the other end of the alphabet.
    * @param word -- A plaintext string to be encoded.
    * @return -- The encoded word.
    */
    public String encode(String word){
        StringBuffer result = new StringBuffer(""); //Use a StringBuffer
        int k = 0;                // Start with the first letter of word
        while (k < word.length()){    // For each letter in word
          result.append((char)('z' - (word.charAt(k)-'a'))); //encode it
          k++;                      // go to the next letter
        } // while
        return result.toString();
    } // encode()

    /**
    * decode() replace each letter between 'a' and 'z' in
    * word with a letter reflected to a letter an equal distance
    * from the other end of the alphabet.
    * @param word -- A pciphertext string to be decoded.
    * @return -- The decoded word.
    */
    public String decode(String word){
        return encode(word);   // decode() is identical to encode();
    } // decode()

    /**
     * main() -- Tests the encode() and decode() methods of Reflect.
     * @param args -- Not used in this method.
     */
     public static void main(String args[]){
         Reflect ref = new Reflect(); // Create an instance of Reflect
         String plain = "this is a secret message"; // The plaintext
         String encrypted = ref.encrypt(plain);      // Encrypt it
```

```
            String decrypted = ref.decrypt(encrypted);  // Decrypt it
            System.out.println("Plaintext: " + plain);  //Print them out
            System.out.println("Encrypted: " + encrypted);
            System.out.println("Decrypted: " + decrypted);
        } // main()

} // Reflect class
```

11. One way to design a substitution alphabet for a cipher is to use a keyword to construct the alphabet. For example, suppose the keyword is "zebra". You place the keyword at the beginning of the alphabet, and then fill out the other 21 slots with the remaining letters, giving the following alphabet:

    ```
    Cipher alphabet:   zebracdfghijklmnopqstuvwxy
    Plain alphabet:    abcdefghijklmnopqrstuvwxyz
    ```

    Design and implement an `Alphabet` class for constructing substitution alphabets. It should have a single public method that takes a keyword `String` as an argument and returns an alphabet string. Note that an alphabet cannot contain duplicate letters, so repeated letters in a keyword like "xylophone" would have to be removed.

    Answer: An `Alphabet` class solution is listed below.

    ```
    /**
     * File: Alphabet.java
     * Author: Java, Java, Java
     * Description: This class contains a single static method that has
     * a keyword parameter and returns a cipher alphabet string.
     */

    public class Alphabet{
        /**
        * makeCipherAlph(kword) removes any duplicate letters from kword
        * and then appends letters not in kword in the order that they
        * appear in the ordinary 26 letter alphabet.
        * @param kword -- A keyword string.
        * @return -- A 26 letter permutation of the alphabet.
        */
        public static String makeCipherAlph(String kword){
            String permStr = "";    //Use a String

            // First remove duplicates from kword
            int k = 0;        // Start at begining of kword
            char ch;          // To store a candidate to add to permStr
            while (k < kword.length()){
                ch = kword.charAt(k);     // For each letter in kword
                if (permStr.indexOf(ch) < 0) // if ch is not in permStr
                    permStr = permStr + ch;  // add it to permStr
                k++;                      // go to the next letter in kword
    ```

```
        } // while

        // Next add unused letters to permStr
        String aword = "abcdefghijklmnopqrstuvwxyz";  // The alphabet
        k = 0;                        // Start at the beginning of aword
        while (k < aword.length()){
            ch = aword.charAt(k);     // For each letter in aword
            if (permStr.indexOf(ch) < 0) // if ch is not in permStr
                permStr = permStr + ch;  // add it to permStr
            k++;                      // go to the next letter in aword
        } // while

        return permStr;
    } // makeCipherAlph()

    /**
     * main() -- Tests the makeCipherAlph() method.
     * @param args -- Not used in this method.
     */
     public static void main(String args[]){
         System.out.println("zebra");
         System.out.println(Alphabet.makeCipherAlph("zebra"));
         System.out.println("zookeeper");
         System.out.println(Alphabet.makeCipherAlph("zookeeper"));

     } // main()

} // Alphabet class
```

12. Design and write a `Cipher` subclass for a substitution cipher that uses an alphabet from the `Alphabet` class created in the preceding exercise.

    Answer: The `Substitution` class solution below extends the `Cipher` class and uses the `Alphabet` class from the previous exercise.

```
/**
 * File: Substitution.java
 * Author: Java, Java, Java
 * Description: Substitution extends the Cipher class and implements
 * the the encode method with a substitution of letters prescribed
 * by a keyword. The decode() is the inverse function of encode. The
 * Alhpabet class is used to create the cipher alphabet from a keyword.
 */

public class Substitution extends Cipher{

    private String ciphAlph;    // Stores the alphabet for encoding
    private String decodeAlph;  // Stores the alphabet for decoding

    public Substitution(String kword){ // Create the cipher alphabet
```

```
        ciphAlph = Alphabet.makeCipherAlph(kword);

        // Now find the inverse function for the cipher alphabet
        StringBuffer decodeSB =
            new StringBuffer("abcdefghijklmnopqrstuvwxyz");
        char ch;               // For storing letters from ciphAlph
        for (int k = 0; k < 26; k++){
            ch = ciphAlph.charAt(k); // get k-th letter from
                // Assign k-th letter of alphabet to decodeSB
            decodeSB.setCharAt((int)(ch - 'a'), (char)('a' + k));
        } // for
        decodeAlph = decodeSB.toString();

    } // Substitution() constructor

    /**
    * encode() replace each letter between 'a' and 'z' in
    * word with a corresponding letter from ciphAlph.
    * @param word -- A plaintext string to be encoded.
    * @return -- The encoded word.
    */
    public String encode(String word){
             //Use a StringBuffer
        StringBuffer result = new StringBuffer("");
        int k = 0;         // Start with the first letter of word
        char ch;           // To store each letter of word
        while (k < word.length()){     // For each letter in word
            ch = word.charAt(k);
               // encode it
            result.append(ciphAlph.charAt((int)(ch - 'a')));
            k++;                       // go to the next letter
        } // while
        return result.toString();
    } // encode()

    /**
    * decode() replace each letter between 'a' and 'z' in
    * word with a corresponding letter from decodeAlph.
    * @param word -- A ciphertext string to be be decoded.
    * @return -- The decoded word.
    */
    public String decode(String word){
            //Use a StringBuffer
        StringBuffer result = new StringBuffer("");
        int k = 0;        // Start with the first letter of word
        char ch;          // To store each letter of word
        while (k < word.length()){    // For each letter in word
            ch = word.charAt(k);
               // decode it
            result.append(decodeAlph.charAt((int)(ch - 'a')));
```

```
            k++;                        // go to the next letter
         } // while
         return result.toString();
      } // decode()

      /**
       * main() -- Tests the encode() and decode() methods.
       * @param args -- Not used in this method.
       */
       public static void main(String args[]){
                //A Substitution object
           Substitution sub = new Substitution("zebra");
           String plain =
                   "this is a secret message"; // The plaintect
           String encrypted = sub.encrypt(plain); // Encrypt it
           String decrypted = sub.decrypt(encrypted); // Decrypt it
           System.out.println("Plaintext: " + plain); //Print them out
           System.out.println("Encrypted: " + encrypted);
           System.out.println("Decrypted: " + decrypted);
       } // main()

} // Substitution class
```

13. *Challenge:* Find a partner and concoct your own encryption scheme. Then work separately, with one partner writing `encode()` and the other writing `decode()`. Test to see that a message can be encoded and then decoded to yield the original message.

    Answer: Of course, a solution for this programming exercise depends on the encryption method chosen.

14. Design a `TwoPlayerGame` subclass called `MultiplicationGame`. The rules of this game are that the game generates a random multiplication problem using numbers between 1 and 10, and the players, taking turns, try to provide the answer to the problem. The game ends when a wrong answer is given. The winner is the player who did not give the wrong answer.

    Answer: The following implementation of `MultiplicationGame` uses a `MultiplicationPlayer` object from the class implemented in the next exercise.

```
/**
 * File: MultiplicationGame.java
 * Author: Java, Java, Java
 * Description: This class extends TwoPlayerGame and implements
 * CLUIPlayableGame The Multiplication Game is played by two players
 * who answer multiplication problems generated by the computer.
 * The first player answering incorrectly
 * loses the game.
 */
```

```
public class MultiplicationGame extends TwoPlayerGame
                                    implements CLUIPlayableGame{
      // n1 and n2 are public since MultiplicationPlayer object
      // needs to know the numbers
    public int n1,n2;
    private int product; // The correct answer
    private int answer;  // The player's answer
    private int nCorrect = 0;
    private boolean gameOver = false;

    public MultiplicationGame() {
n1 = 1 + (int)(Math.random()* 10);
n2 = 1 + (int)(Math.random()* 10);
product = n1 * n2;
    }

    public String getRules() {
        return "\n*** The Rules of Multiplication Game ***\n" +
        "1) Multiplication problems will be presented.\n" +
        "2) Players will take turns answering the questions.\n" +
        "3) Player 1 will start.\n" +
        "4) The game ends when a player gives a wrong answer\n" +
        "5) The player giving the wrong answer loses.\n";
    } // getRules()

    public String getWinner() {
        if (onePlaysNext)
            return "The winner is player 2";
        else
            return "The winner is player 1";
    } // getWinner()

    public String move(String s) {
        answer = Integer.parseInt(s);
        gameOver = answer != product;
        if (!gameOver) {
            ++nCorrect;
    n1 = 1 + (int)(Math.random()* 10);
    n2 = 1 + (int)(Math.random()* 10);
    product = n1 * n2;
            changePlayer();
    return "Good.  Your answer is correct!\n";
} else {
    return "Oops! " + n1 + " times " + n2 + " equals "
                                                + product + "\n";
} // else
    } // move()

    public boolean gameOver() {
```

```
    return gameOver;
} // gameOver()

/**
 * play() is implemented as part of the CLUIPlayableGame
 * interface. It contains the control algorithm for playing
 * MultiplicationGame game.
 * @param ui gives a reference to the UserInterface where
 * I/O is performed.
 */
public void play(UserInterface ui) {  // From CLUIPlayableGame
    ui.report(getRules());
    if (computer1 != null)
        ui.report("\nPlayer 1 is a "+computer1.toString());
    if (computer2 != null)
        ui.report("\nPlayer 2 is a "+computer2.toString() + "\n");

    while(!gameOver()) {
       IPlayer computer = null; // Assume computer is not playing
       ui.report(reportGameState());
       switch(getPlayer()) {
       case PLAYER_ONE:             // Player 1's turn
           computer = computer1;
           break;
       case PLAYER_TWO:             // Player 2's turn
           computer = computer2;
           break;
       } // cases

       if (computer != null) {      // If computer's turn
           ui.prompt(getGamePrompt());
           ui.report(move(computer.makeAMove("")));
       } else {                     // otherwise, user's turn
           ui.prompt(getGamePrompt());
           ui.report(move(ui.getUserInput()));
       }
    } // while
    ui.report(reportGameState());  // The game is now over
} //play()

/**
 * getGamePrompt() is implemented as part of the CLUIPlayableGame
 * interface. It defines the prompt presented to the user before
 * each move.
 * @return a String giving the prompt.
 */
public String getGamePrompt() {
 return "\nPlayer " + getPlayer() + ": What is " + n1 + " times "
                                                    + n2 + "? " ;
} // getGamePrompt()
```

```
    /**
     * reportGameState() is implemented as part of the CLUIPlayableGame
     * interface. It describes the current state of the game.
     * @return a String describing the game's current state.
     */
    public String reportGameState() {
        if (!gameOver())
             return "";
        else
            return "Game over! " + getWinner() +"\n";
    }   // reportGameState()


    /**
     * main() plays the Multiplication game
     * @param args[] - is not used in this implementation.
     */
    public static void main(String args[])  {
        UserInterface kb = new KeyboardReader();
        MultiplicationGame game = new MultiplicationGame();
        IPlayer computer = new MultiplicationPlayer(game);
        game.addComputerPlayer(computer);
        game.play(kb);
    } //main()

} // MultiplicationGame
```

15. Design a class called `MultiplicationPlayer` that plays the multiplication game described in the preceding exercise. This class should implement the `IPlayer` interface.

    Answer: An implementation of `MultiplicationPlayer` is listed below.

```
/*
 * File: MultiplicationPlayer.java
 * Author: Java, Java, Java
 * Description: Defines an IPlayer for MultiplicationGame.
 *  It implements the makeAMove() method in a way which it
 *  multiplies corectly for a random number of plays between
 *  one and 10 times.
 */
public class MultiplicationPlayer implements IPlayer {

    private MultiplicationGame game;
    private int numCorrectAnswers;
    private int currentTurn = 0;

    /**
     * MultiplicationPlayer() constructor is given a
```

```
     * reference to the MultiplicationGame object.
     * @param - game is a reference to MultiplicationGame
     * object.
     */
    public MultiplicationPlayer(MultiplicationGame g) {
        game = g;
        numCorrectAnswers = 1 + (int)(Math.random() * 10);
        currentTurn = 0;
    } // MultiplicationPlayer() constructor

    /**
     * makeAMove() is defined as part of the IPlayer interface. It
     *  defines a valid move in the MultiplicationGame game.
     * This version simply does the multiplication correctly for
     * a fixed number of turns which is randomly chosen in the
     * constructor.
     * @param prompt is a string containing a prompt
     * @return a string describing the move
     */
    public String makeAMove(String prompt)  {
        currentTurn++;
        System.out.print("\nMultiplicationPlayer answers ");
        if (currentTurn <= numCorrectAnswers){
            System.out.println("" + (game.n1*game.n2));
            return "" + game.n1*game.n2;
        } else {
            System.out.println("" + (1 + game.n1*game.n2));
            return "" + 1 + game.n1*game.n2;
        } // else
    } // makeAMove()

    /**
     * toString() provides a string representation of this object.
     */
    public String toString() { // returns 'MultiplicationPlayer'
        String className = this.getClass().toString();
        return className.substring(5);
    } // toString()
} // MultiplicationPlayer
```

16. Design a `TwoPlayerGame` subclass called `RockPaperScissors`. The rules of this game are that both players simultaneously make a pick: either a rock, a paper, or a scissors. For each round, the rock beats the scissors, the scissors beats the paper, and the paper beats the rock. Ties are allowed. The game is won in a best out of three fashion when one of the players wins two rounds.

    Answer: The implementation of `RockScissorsPaper` uses the class defined as a solution to the next exercise.

```
/**
```

```
 * File: RockScissorsPaper.java
 * Author: Java, Java, Java
 * Description: This class extends TwoPlayerGame and implements
 * CLUIPlayableGame The RockScissorsPaper Game is played by two
 * players who choose one of rock, scissors, or paper without
 * knowing the choice of the other player. The winner of the
 * game is determined by the rules: (1) rock breaks scissors,
 * (2) scissors cuts paper, (3) paper covers rock, and (4) if
 * players make the same choice they choose again.
 */

public class RockScissorsPaper extends TwoPlayerGame
                                implements CLUIPlayableGame{
    public static final int ROCK = 1;
    public static final int SCISSORS = 2;
    public static final int PAPER = 3;
    private int choiceOfOne = -1; // Player One's choice
    private int choiceOfTwo = -2;  // Player Two's choice
    private boolean gameOver = false;

    public RockScissorsPaper() {
    } // RockScissorsPaper() constructor

    /**
     * choiceString(choice) returns "rock", "scissors", or "paper"
     * for 1, 2, or 3.
     * @param  choice an integer 1, 2, or 3.
     * @return a String "rock", "scissors", or "paper".
     */
    public static String choiceString(int choice){
        switch(choice){
            case ROCK : return "rock";
            case SCISSORS : return "scissors";
            case PAPER : return "paper";
            default : return "invalid choice";
        } // switch()
    } // choiceString()


    public String getRules() {
        return "\n** The Rules of RockScissorsPaper Game **\n" +
        "1) Each player chooses rock, scissors, or paper.\n" +
        "2) Rock breaks scissors.\n" +
        "3) Scissors cuts paper.\n" +
        "4) Paper covers rock.\n" +
        "5) Choose again if choices were the same.\n";
    } // getRules()

    public String getWinner() {
        if (choiceOfTwo < 0) // One or both players haven't chosen yet.
```

```
        return "There is no winner yet";
    if (choiceOfOne == choiceOfTwo) // Player have same choice
        return "There is no winner yet";
    if ((choiceOfOne + 3 - choiceOfTwo) % 3 == 2)
        return "Player One is the winner!";
    if ((choiceOfOne + 3 - choiceOfTwo) % 3 == 1)
        return "Player Two is the winner!";
       // This should never be executed.
    return "There is an error in getWinner()";
} // getWinner()

public String move(String s) {
    int answer = Integer.parseInt(s);
    String str = "";  // for the return value
    if ((answer >= 1) && (answer <= 3)){  // valid choice
        if (getPlayer() == TwoPlayerGame.PLAYER_ONE)
            choiceOfOne = answer;
        else
            choiceOfTwo = answer;
          // Report choice
       str = "Player " + getPlayer() + " has chosen ";
       str = str + choiceString(answer) + "\n";
         // Valid choice so change player
       changePlayer();

        // If both choices have been made - evaluate
       if (choiceOfTwo > 0){
             // If same choice - reset variables and report.
            if ( choiceOfOne == choiceOfTwo) {
                 choiceOfOne = -1;
                 choiceOfTwo = -2;
                 str = str + "The choices were the same! ";
                 str = str + "Choose again.\n";
            } else { // there is a winner
                 gameOver = true;
                 str = str + "\n";
            } // else
        } // if both choices made
    } else {
         str = str + "Invalid choice!?!? Try again.\n";
    } // else
    return str;
} // move()

public boolean gameOver() {
    return gameOver;
} // gameOver()

/**
 * play() is implemented as part of the CLUIPlayableGame
```

```
 * interface. It contains the control algorithm for playing
 * the RockScissorsPaper game.
 * @param ui gives a reference to the UserInterface where
 * I/O is performed.
 */
public void play(UserInterface ui) {  // From CLUIPlayableGame
    ui.report(getRules());
    if (computer1 != null)
        ui.report("\nPlayer 1 is a " + computer1.toString());
    if (computer2 != null)
        ui.report("\nPlayer 2 is a " + computer2.toString() + "\n");

    while(!gameOver()) {
       IPlayer computer = null;  // Assume computers is not playing
       ui.report(reportGameState());
       switch(getPlayer()) {
       case PLAYER_ONE:             // Player 1's turn
           computer = computer1;
           break;
       case PLAYER_TWO:             // Player 2's turn
           computer = computer2;
           break;
       } // switch

       if (computer != null) {      // If computer's turn
           ui.prompt(getGamePrompt());
           ui.report(move(computer.makeAMove("")));
       } else {                     // otherwise, user's turn
           ui.prompt(getGamePrompt());
           ui.report(move(ui.getUserInput()));
       }
    } // while
    ui.report(reportGameState());  // The game is now over
} //play()

/**
 * getGamePrompt() is implemented as part of the CLUIPlayableGame
 * interface. It defines the prompt presented to the user before
 * each move.
 * @return a String giving the prompt.
 */
public String getGamePrompt() {
 return "\nPlayer " + getPlayer() + ":\nInput 1 if you choose ROCK.\n" +
     "Input 2 if you choose SCISSORS.\nInput 3 if you choose PAPER.\n" +
     "Input your choice: "  ;
} // getGamePrompt()

/**
 * reportGameState() is implemented as part of the CLUIPlayableGame
 * interface. It describes the current state of the game.
```

```
     * @return a String describing the game's current state.
     */
    public String reportGameState() {
        if (!gameOver())
             return "";
        else
            return "Game over! " + getWinner() +"\n";
    }   // reportGameState()


    /**
     * main() plays the RockScissorsPaper game
     * @param args[] - is not used in this implementation.
     */
    public static void main(String args[])  {
        UserInterface kb = new KeyboardReader();
        RockScissorsPaper game = new RockScissorsPaper();
        IPlayer computer = new RockScissorsPaperPlayer(game);
        game.addComputerPlayer(computer);
        game.play(kb);
    } //main()

} // RockScissorsPaper
```

17. Design a class called `RockPaperScissorsPlayer` that plays the game described in the preceding exercise. This class should implement the `IPlayer` interface.

    Answer: An implementation of `RockScissorsPaperPlayer` is listed below.

```
/*
 * File: RockScissorsPaperPlayer.java
 * Author: Java, Java, Java
 * Description: Defines an IPlayer for RockScissorsPaper.
 * It implements the makeAMove() method in a way which it
 * a random choice of 1, 2, o r3 is made.
 */
public class RockScissorsPaperPlayer implements IPlayer {

    private RockScissorsPaper game; // Reference is not used.

    /**
     *RockScissorsPaperPlayer() constructor is given a
     * reference to the RockScissorsPaper object.
     * @param - game is  reference to RockScissorsPaper
     * object.
     */
    public RockScissorsPaperPlayer(RockScissorsPaper g) {
        game = g;
```

```
    } // RockScissorsPaperPlayer() constructor

    /**
     * makeAMove() is defined as part of the IPlayer interface. It
     * defines a valid move in the RockScissorsPaper game.
     * This version randomly returns 1, 2, or 3.
     * @param prompt is a string containing a prompt
     * @return a string describing the move
     */
    public String makeAMove(String prompt)  {
        int myChoice = 1 + (int)(Math.random() * 3);
        System.out.print("\nWithout knowing the previous choice,");
        System.out.println(" I choose " +
           RockScissorsPaper.choiceString(myChoice) + ".");
        return "" + myChoice;
    } // makeAMove()

    /**
     * toString() provides a string representation of this object.
     */
    public String toString() { // returns 'RockScissorsPaperPlayer'
        String className = this.getClass().toString();
        return className.substring(5);
    } // toString()
} // RockScissorsPaperPlayer
```

Chapter 9

Arrays and Array Processing

1. Explain the difference between the following pairs of terms.

   (a) An *element* and an element *type*.

   Answer: An element is the actual piece of data that is stored or referenced in the array. An element type is the type of data that is stored (boolean, int, String, etc.) in the array.

   (b) A *subscript* and an *array element*.

   Answer: A subscript is the position of an element within the array. An element is an actual piece of data stored or referenced in the array.

   (c) A *one-dimensional* array and *two-dimensional* array.

   Answer: A one dimensional array is an array whose components are data of various types. A two dimensional array is an array whose components are themselves arrays.

   (d) An *array* and a *vector*.

   Answer: A Vector implements an array of objects that can grow in size as needed unlike an array object which is fixed in size.

   (e) A *insertion sort* and a *selection sort*.

   Answer: Insertion sort goes through a list of N elements N - 1 times. First the first two elements are put in the correct order. On each subsequent time through the list, one additional element is put in the correct position in the sorted part of the list. Selection sort goes also goes through the list of N elements N - 1 times. First time through it recognizes the smallest element and places it in the first spot. On each subsequent time through the list, the smallest element in the unsorted part of the list is found and put at the end of the sorted part of the list.

   (f) A *binary search* and a *sequential search*.

   Answer:A sequential search checks each element individually to see if it's equal to the value it's looking for and return the subscript of the element when it is found. A binary search can be used when the elements are in

order. It checks the number in the middle first and if it is smaller then this number it will check in the middle of the lower half of the list. It keeps narrowing down the range like this until it determines the subscript of the element it is looking for.

2. Fill in the blank:

   (a) The process of arranging an array's elements into a particular order is known as _____. **Answer:** sorting

   (b) One of the preconditions of the binary search method is that the array has to be _____. **Answer:** ordered

   (c) An _____ is an object that can store a collection of elements of the same type. **Answer:** sorted

   (d) An _____ is like an array except that it can grow. **Answer:** Vector

   (e) For an array, its _____ is represented by an instance variable. **Answer:** length

   (f) An expression that can be used during array instantiation to assign values to the array is known as an _____. **Answer:** array initializer

   (g) A _____ is an array of arrays. **Answer:** multidimensional array

   (h) A sort method that be used to sort different types of data is known as a _____ method. **Answer:** polymorphic

   (i) To instantiate an array you have to use the _____ operator. **Answer:** new

   (j) An array of objects stores _____ to the objects. **Answer:** references

3. Make each of the following array declarations.

   (a) A 4×4 array of `doubles`. **Answer:**

```
double doubArr[][] = new double[4][4];
```

   (b) A 20×5 array of `Strings`. **Answer:**

```
String strArr [][] = new String[20][5];
```

   (c) A 3×4 array of `char` initialized to '*'. **Answer:**

```
char charArr [][] =
 {{'*','*','*','*'},{'*','*','*','*'},{'*','*','*','*'}}
```

   (d) A $2 \times 3 \times 2$ array of `boolean` initialized to true. **Answer:**

```
boolean boolArr [][][] = new boolean[2][3][2];
for (int num = 0; num < boolArr.length; num ++)
  for (int num2 = 0; num2 < boolArr[num].length; num2++)
    for (int num3 = 0; num3 < boolArr[num2][num3]; num3++)
      boolArr[num2][num3] = true;
```

   (e) A 3×3 array of `Students`. **Answer:**

```
Student stuArr[][] = new Student[3][3];
```

(f) A 2×3 array of `String`s initialized to "one," "two," and so on. **Answer:**

```
String strArr[][] = new String [2][3];
strArr[0][0] = new String ("one");
strArr[0][1] = new String ("two");
strArr[0][2] = new String ("three");
strArr[1][0] = new String ("four");
strArr[1][1] = new String ("five");
strArr[1][2] = new String ("six");
```

4. Identify and correct the syntax error in each of the following.

```
(a) int arr = new int[15];
 (b) int arr[] = new int(15);
 (c) float arr[] = new [3];
 (d) float arr[] = new float {1.0,2.0,3.0};
 (e) int arr[] = {1.1,2.2,3.3};
 (f) int arr[][] = new double[5][4];
 (g) int arr[][] = { {1.1,2.2}, {3.3, 1} };
```

Answer:

```
(a) There are missing brackets after arr.
(b) There are parentheses instead of brackets around 15.
(c) It should be new float[3] instead of new [3].
(d) new float should not be there.
(e) 1.1, 2.2, and 3.3 are not integers
(f) The two declared data types are different.
(g) The initializing data are not all integers.
```

5. Evaluate each of the following expressions, some of which may be erroneous.

```
int arr[] = { 2,4,6,8,10 };
 (a) arr[4]                      (f) arr[ 5 % 2 ]
 (b) arr[ arr.length ]           (g) arr[ arr[ arr[0] ] ]
 (c) arr[ arr[0] ]               (h) arr[ 5 / 2.0 ]
 (d) arr[ arr.length / 2 ]       (i) arr[ 1 + (int) Math.random() ]
 (e) arr[ arr[1] ]               (j) arr[ arr[3] / 2 ]
```

Answer: (a) 10, (b) invalid, (c) 6, (d) 6, (e) 10, (f) 4, (g) invalid, (h) invalid, (i) 4, (j) 10

6. What would be printed by the following code segment?

```
int arr[] = { 24, 0, 19, 21, 6, -5, 10, 16};
for (int k = 0; k < arr.length; k += 2)
    System.out.println( arr[k] );
```

Answer:

```
24
19
6
10
```

7. What would be printed by the following code segment?

```
int arr[][] =
  {{24, 0, 19},{21, 6, -5},{10, 16, 3},{1, -1, 0}};
for (int j = 0; j < arr.length; j++)
    for (int k = 0; k < arr[j].length; k++)
        System.out.println( arr[j][k] );
```

Answer:

```
24
0
19
21
6
-5
10
16
3
1
-1
0
```

8. What would be printed by the following code segment?

```
int arr[][] =
  {{24,0,19},{21,6,-5},{10,16,3},{1,-1,0}};
for (int j = 0; j < arr[0].length; j++)
    for (int k = 0; k < arr.length; k++)
        System.out.println(arr[k][j]);
```

Answer:

```
24
21
10
1
0
6
16
1
19
```

```
5
3
0
```

9. What's wrong with the following code segment, which is supposed to swap the values of the `int` variables, `n1` and `n2`?

```
int temp = n1;
n2 = n1;  //  Answer: It should be n1 = n2;
n1 = temp;  //  It should be temp = n2;
```

10. Explain why the following method does not successfully swap the values of its two parameters? Hint: Think about the difference of value and reference parameters.

```
public void swapEm(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
}
```

Answer: When `swapEm(x,y)` is called, the `int` values of `x` and `y` are copied into parameters `n1` and `n2`. Then the values in `n1` and `n2` are swapped but the values in `x` and `y` remain the original values.

11. Declare and initialize an array to store the following two-dimensional table of values:

```
1   2   3   4
5   6   7   8
9  10  11  12
```

Answer:

```
int intArr[][] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

12. For the two-dimensional array you created in the previous exercise, write a nested for loop to print the values in the following order: 1 5 9 2 6 10 3 7 11 4 8 12. That is, print the values going down the columns instead of going across the rows.

Answer:

```
/* **** Print intArr column by column **** */
for (int col = 0; col < intArr[0].length; col++)
    for (int row = 0; row < intArr.length; row++)
        System.out.print(intArr[row][col] + " ");
System.out.println();
```

13. Define an array that would be suitable for storing the following values:

    (a) The GPAs of 2000 students. **Answer:**

    ```
    double GPA[] = new double [2000];
    ```

    (b) The lengths and widths of 100 rectangles. **Answer:**

    ```
    public class Rectangle {
        private int length;
        private int width;
        // ... etc.
    }
    Rectangle rectangles[] = new Rectangle[100];
    ```

    (c) A week's worth of hourly temperature measurements, stored so that it is easy to calculate the average daily temperature. **Answer:**

    ```
    double Temp[][] = new double [7][24];
    ```

    (d) A board for a tic-tac-toe game. **Answer:**

    ```
    char ticTacToeBoard[][] = new char[3][3];
    ```

    (e) The names and capitals of the 50 states. **Answer:**

    ```
    public class State {
        private String name;
        private String capital;
        // ... etc.
    }
    State states[] = new State[50];
    ```

14. Write a code segment that will compute the sum of all the elements of an array of `int`.

    Answer:

    ```
    int arrInt[] = { 25, 67, 100, -9, 20, 675, 102, 99};
    int sum = 0;
    for (int k = 0; k < arrInt.length; k++)
        sum += arrInt[k];
    ```

15. Write a code segment that will compute the sum of the elements a two-dimensional array of `int`. **Answer:**

    ```
    int arrint = new int[num1][num2]
    int sum = 0;
    for (int k = 0; k < arrint.length; k++)
        for (int j = 0; j < arrint[k].length; j++)
            sum += arrint[k][j];
    ```

16. Write a method that will compute the average of all the elements of a two-dimensional array of `float`.

    Answer:

```
/**
 * average() computes the average of a 2-D array of float
 * @param dArr -- a two-dimensional array of floats
 * @return a double giving the average of the array elements
 */
public static double average(float dArr[][]) {
    double sum = 0;
    int count = 0;
    for (int row = 0; row < dArr.length; row++)
        for (int col = 0; col < dArr[0].length; col++) {
            sum += dArr[row][col];
            count++;
        }
    return sum / count;
} // average()
```

17. Write a method that takes two parameters, an `int` array and an integer, and returns the location of the last element of the array which is greater than or equal to the second parameter.

    Answer:

```
/**
 * findAnElement() finds the location of last element
 * of arr that is >= num. It returns -1 if no such
 * number exists
 * @param arr -- an integer array
 * @param num -- a bound for the search
 * @return the last array element >= num
 */
public static int findAnElement (int arr[], int num) {
    int largest = -1;
    for (int k = 0; k < arr.length; k++) {
        if (arr[k] >= num)
            largest = k;
    }
    return  largest;
} // findAnElement()
```

18. Write a program that tests whether a 3×3 array, input by the user, is a *magic square*. A Magic square is an $N \times N$ matrix of numbers in which every number from 1 to N^2 must appear just once and every row, column and diagonal must add up to the same total — for example,

```
6 7 2
1 5 9
8 3 4
```

    Answer:

```
/* File: MagicSquareTester.java
 * Author: Java, Java, Java
 * Description: This class determines whether a 3 x 3
 * array of ints is a magic square, where a magic square
 * is an array all of whose rows, columns and diagonals
 * add up to the same sum. The algorithm works as follows.
 * Compute the sum of the first column and store this as
 * magicSum. Then, one by one, compute the sum of the
 * remaining columns, the rows, and each diagonal. After
 * each sum is  computed compare it with magicSum. If it
 * is different, return  false. If all the sums equal
 * magicSum, return true.
 *
 *  Note that an array is used to store the sums.
 */
import java.io.*;

public class MagicSquareTester  {

    /**
     * isMagic() determines if its parameter is a
     * magic square
     * @param arr[][] is the array being tested
     * @return is a boolean that is true iff the array
     *   is magic
     */
    public boolean isMagic(int arr[][]) {
          // A 3 x 3 array has 8 sums
     int sums[] = new int[8];
          // Sum the first column
       for (int k = 0; k < 3; k++)
            sums[0] += arr[k][0];
         // This gives the magic number
       int magicSum = sums[0];
         // Sum the second column
       for (int k = 0; k < 3; k++)
            sums[1] += arr[k][1];
       if (magicSum != sums[1])
            return false;
          // Sum the third column
       for (int k = 0; k < 3; k++)
            sums[2] += arr[k][2];
       if (magicSum != sums[2])
            return false;
          // Sum the first row
        for (int k = 0; k < 3; k++)
            sums[3] += arr[0][k];
        if (magicSum != sums[3])
            return false;
          // Sum the second row
```

```
        for (int k = 0; k < 3; k++)
            sums[4] += arr[1][k];
        if (magicSum != sums[4])
            return false;
          // Sum the third row
        for (int k = 0; k < 3; k++)
            sums[5] += arr[2][k];
        if (magicSum != sums[5])
            return false;
          // Sum left diagonal
        for (int k = 0; k < 3; k++)
            sums[6] += arr[k][k];
        if (magicSum != sums[6])
            return false;
          // Sum right diagonal
        for (int k = 0; k < 3; k++)
            sums[7] += arr[k][3 - k -1];
        if (magicSum != sums[7])
            return false;
          // If all tests pass, return true
        return true;
    } // isMagic()

    /**
     * print() prints an array of integers as a magic
     * square
     * @param arr[][] is the array being printed
     */
    public void print(int arr[][]) {
        for (int j = 0; j < arr.length; j++) {
            for (int k = 0; k < arr[j].length; k++)
                System.out.print(arr[j][k] + " ");
            System.out.println();
        }
    } // print()

    /**
     * main() creates an instance of MagicSquareTester and
     * uses it to test arrays of different dimensionality
     */
    public static void main(String argv[]) {
        int magic[][] = { {6,7,2}, {1,5,9}, {8,3,4} };
        int notMagic[][] = { {1,7,2}, {6,5,9}, {3,8,4} };

        MagicSquareTester tester = new MagicSquareTester();
        tester.print(magic);
        if ( tester.isMagic(magic))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");
```

```
        tester.print(notMagic);
        if ( tester.isMagic(notMagic))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");
    } // main()
} // MagicSquareTester
```

19. Revise the program in the previous exercise so that it allows the user to input the dimensions of the array, up to 4×4.

Answer:

```
/*
 * File: GenericMagicSquareTester.java
 * Author: Java, Java, Java
 * Description: This class determines whether an N x N
 * array of ints is a magic square, where a magic square
 * is an array all of whose rows, columns and diagonals
 * add up to the  same sum.  The key to making this work
 * for the generic case  is to use simple arithmetic to
 * refer to the various sums. For an  N x N array the
 * sums array must have 2N +2  elements. For  example, a
 * 4 x 4 would have 10 sums. The following scheme can be
 * used to keep track of what sums go where.
 *
 *    sums[0] ... sums[N-1]  are used for the columns
 *    sums[N] ... sums[2N-1] are used for the rows
 *    sums[2N]               is used for the left diagonal
 *    sums[2N+1]             is used for the right diagonal
 *
 * Finally, note that the isMagic(int arr[][]) does not
 * refer to the length of the array. So its algorithm can
 * work for any square array as long as you use arr.length
 * as your loop bound.
 */
import java.io.*;

public class GenericMagicSquareTester  {

    /**
     * isMagic() determines if its parameter is a magic
     * square
     * @param arr[][] is the array being tested
     * @return is a boolean that is true iff the array
     *   is magic
     */
    public boolean isMagic(int arr[][]) {
          // Sums of rows, cols, diags
        int sums[] = new int[2 * arr.length + 2];
```

```
      // Sum the first column
    for (int k = 0; k < arr.length; k++)
        sums[0] += arr[k][0];
      // This gives the magic number
    int magicSum = sums[0];
      // Sum the rest of the columns
    for (int j = 1; j < arr.length; j++) {
        for (int k = 0; k < arr.length; k++)
            sums[j] += arr[k][j];
        if (magicSum != sums[j])
            return false;
    } // for each column
      // Sum the rows
    for (int j = 0; j < arr.length; j++) {
        for (int k = 0; k < arr.length; k++)
            sums[j + arr.length] += arr[j][k];
        if (magicSum != sums[j + arr.length])
            return false;
    } // for each row
      // Sum left diagonal
    for (int k = 0; k < arr.length; k++)
        sums[2 * arr.length] += arr[k][k];
    if (magicSum != sums[2 * arr.length])
        return false;
      // Sum right diagonal
    for (int k = 0; k < arr.length; k++)
        sums[2 * arr.length + 1] +=
                    arr[k][arr.length - k -1];
    if (magicSum != sums[2 * arr.length + 1])
        return false;
      // If all tests pass, return true
    return true;
} // isMagic()

/**
 * print() prints an array of integers as a magic
 * square
 * @param arr[][] is the array being printed
 */
public void print(int arr[][]) {
    for (int j = 0; j < arr.length; j++) {
        for (int k = 0; k < arr[j].length; k++)
            System.out.print(arr[j][k] + " ");
        System.out.println();
    }
} // print()

/**
 * main() creates an instance of MagicSquareTester
 * and uses it to test arrays of different
```

```
     * dimensionality
     */
    public static void main(String argv[]) {
        int magic[][] = { {6,7,2}, {1,5,9}, {8,3,4} };
        int notMagic[][] = { {1,7,2}, {6,5,9}, {3,8,4} };
        int magic4[][] =
          {{16,3,2,13},{5,10,11,8},{9,6,7,12},{4,15,14,1}};
        int notMagic4[][] =
          {{1,7,2,6},{6,5,9,10},{3,8,4,65},{2,2,2,2}};
        int magic2[][] = {{1,1},{1,1}};

        GenericMagicSquareTester tester =
                        new GenericMagicSquareTester();

        tester.print(magic);
        if ( tester.isMagic(magic))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");

        tester.print(notMagic);
        if ( tester.isMagic(notMagic))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");

        tester.print(magic4);
        if ( tester.isMagic(magic4))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");

        tester.print(notMagic4);
        if ( tester.isMagic(notMagic4))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");

        tester.print(magic2);
        if ( tester.isMagic(magic2))
            System.out.println("This is a magic square");
        else
            System.out.println("This is NOT a magic square");
    } // main()
} // MagicSquare
```

20. Modify the `AnalyzeFreq` program so that it can display the relative frequencies of the 10 most frequent and 10 least frequent letters.

    Answer: This problem uses the `LetterFreq` class as modified in Self-Study

Exercise 9.20 (which implements the `Comparable` interface) and pairs a letter ('A') with its frequency (10). We modifiy the `AnalyzeFreq` class as previously modified in Self-Study Exercise 9.21 (so that it contains a sort method). The only modification needed for this problem is to add a `printExtremeFreqs()` method so that it reports the 10 most and 10 least frequent letters.

```
/*
 * File: LetterFreq.java
 * Author: Java, Java, Java
 * Description: This class stores the frequency of a
 * letter as an integer and the letter as a character
 * and implements Comparable.
 */
public class LetterFreq implements Comparable{
    private char letter; // A character being counted
    private int freq;    // The frequency of letter

    /**
     * LetterFreq() a constructor that sets the values of
     * the freqand letter variables.
     * @param ch is the character that the LetterFreq
     *   object will store
     * @param num is the frequency of the letter stored by
     *   the object
     */
    public LetterFreq  (char ch, int num) {
        letter = ch;
        freq = num;
    }

    public char getLetter(){
        return letter;
    }

    public int getFreq(){
        return freq;
    }

    public void incrFreq(){
        freq++;
    }

    /**
     * compareTo() the method in the Comparable interface
     * which returns +1, 0, or -1 depending on whether this
     * is greater than, equal to, or less than lf.
     * @param lf a LetterFreq parameter declared as a member
     *   of Object.
     */
     public int compareTo(Object lf) {
```

```
            LetterFreq letFreq = (LetterFreq)lf;
            if (freq < letFreq.getFreq())
                return -1;
            else if (freq > letFreq.getFreq())
                return +1;
            else return 0;
        } // compareTo()

} // LetterFreq

/*
 * File: AnalyzeFreq.java
 * Author: Java, Java, Java
 * Description: This class takes a String and analyzes the
 * relative frequencies  of all 26 letters, 'A' to 'Z'.  It
 * uses the LetterFreq class.
 */

import java.util.*;

public class AnalyzeFreq {
    private LetterFreq[] freqArr; //An array of frequencies

    /**
     * AnalyzeFreq() is a constructor that creates an array
     * of 26 LetterFreq objects representing a letters A - Z
     * the alphabet with frequencies set to zero.
     */
    public AnalyzeFreq() {
        freqArr = new LetterFreq[26];
        for (int k = 0; k < 26; k++) {
          freqArr[k] = new LetterFreq((char)('A' + k), 0);
        } // for
    } // AnalyzeFreq() constructor

    /**
     * countLetters() takes a String and analyzes it for
     * the frequency of each letter. The frequencies of the
     * letters are stored in an array of 26 LetterFreq objects
     * each representing a letter of the alphabet.  Each time
     * an instance of the letter is found the corresponding
     * integer in the array is incremented.
     * @param str is the String that is passed to the method
     *    to be analyzed
     */
    public void countLetters(String str) {
        char let;
        str = str.toUpperCase();
        for (int k = 0; k < str.length(); k++) {
            let = str.charAt(k);
```

```
        if ((let >= 'A') && (let <= 'Z')) {
             freqArr[let - 'A'].incrFreq();
        } // if
    } // for
} // countLetters()

/**
 * sort() uses the sort() method of the java.util.Arrays
 * library to sort the freqArr array into increasing order
 * of letterfrequencies.
 */
public void sort() {
    java.util.Arrays.sort(freqArr);
} // sort()

/**
 * printArray() prints the letters and their frequencies
 * stored in the freqArr array.
 */
public void printArray() {
  for (int k = 0; k < 26; k++) {
      System.out.print("letter " + freqArr[k].getLetter());
      System.out.println("  freq " + freqArr[k].getFreq());
  } //for
} // printArray()

/**
 * printExtremeFreqs() prints the letters and their
 * frequencies of the 10 most frequent and 10 least
 * frequent letters stored in the freqArr array.
 */
public void printExtremeFreqs() {
    sort();  // Sort the array into increasing order.
    System.out.println("The ten most frequent letters:");
    for (int k = 25; k >= 16; k--) {
      System.out.print("letter " + freqArr[k].getLetter());
      System.out.println("  freq " + freqArr[k].getFreq());
    } // for
    System.out.println("The ten least frequent letters:");
    for (int k = 9; k >= 0; k--) {
      System.out.print("letter " + freqArr[k].getLetter());
      System.out.println("  freq " + freqArr[k].getFreq());
    } // for
} // printExtremeFreqs()

/**
 * main() creates a string consisting of the statement of
 * Exercise 9.20 and creates an AnalyzeFreq object and
 * counts the frequencies of letters in the string.
 * The printExtremeFreqs() prints the letters and their
```

```
     * frequencies of the 10 most frequent and 10 least
     * frequent letters stored in the freqArr array.
     */
    public static void main(String argv[]){
      String str = "Modify the AnalyzeFreq program so that it";
      str = str+"can display the relative frequencies of the ten";
      str = str+"most frequent and ten least frequent letters.";
      AnalyzeFreq af = new AnalyzeFreq();
      af.countLetters(str);
      af.printExtremeFreqs();
    } // main()

} // AnalyzeFreq class
```

21. The *merge sort* algorithm takes two collections of data that have been sorted and merges them together. Write a program that takes two 25-element `int` arrays, sorts them, and then merges them, in order, into one 50-element array.

    Answer: A single class, the `Merge` class, can be used for this problem. Its main component will be the `mergeSort()` algorithm. Its `main()` method contains a simple test of the merge sort algorithm.

```
/*
 * File: Merge.java
 * Author: Java, Java, Java
 * Description: This class contains the mergeSort method,
 * which takes two sorted integer arrays and merges
 *  them, in order, into one single array.
 *
 * Algorithm: The mergesort algorithm goes through each
 *  array, element by element, putting lower element into
 *  the merged array until all the elements from both arrays
 *  have been merged.
 */

public class Merge {
    /**
     * mergeSort() merges the two arrays, arr1 and
     *     arr2 into ascending order
     * @param arr1 -- an integer array in ascending order
     * @param arr2 -- an integer array in ascending order
     * @return -- an integer array with length
     *   arr1.length + arr2.length containing the elements
     *   of arr1 and arr2 in ascending order.
     * PRE: arr1 and arr2 must be in ascending order
     * POST: the return arr is in ascending order
     * Algorithm: The while loop terminates when all the
     * elements from one or the other array have been
     * merged. After the loop it is necessary to merge the
     * remaining elements from the unfinished array into
```

```
 * the result.
 */
public int[] mergeSort (int arr1[], int arr2[]) {
    int newArr[] = new int[arr1.length + arr2.length];
               // Stores the result
    int k = 0;  // Index to arr1
    int j = 0;  // Index to arr2
    int i = 0;  // Index to newArr
            // While either array has elements
    while (k < arr1.length && j < arr2.length) {
        if (arr1[k] <= arr2[j]) //Take the lower value
            newArr[i] = arr1[k++]; // from either arr1
        else
            newArr[i] = arr2[j++];    // or arr2
        i++;        //   and increment newArr index
    } // while
      // Now merge the unfinished array
      // If arr1 is finished first
      // merge the rest from arr2
    if (k == arr1.length)
        for (int m = j; m < arr2.length; m++)
            newArr[i++] = arr2[m];
    else if (j == arr2.length) {
      // Or if arr2 is finished first, merge from arr1
        for (int m = k; m < arr1.length; m++)
            newArr[i++] = arr1[m];
    } // else if
    return newArr;     // Return the merged array
} // mergeSort()


/**
 * bubbleSort() sorts an array of integers and orders it
 * according to size
 * @param arr[] is the array of integers the method is
 *      passed to sort according to size
 */
public void bubbleSort(int arr[]) {
    int temp;
    for (int pass = 1; pass < arr.length; pass++)
        for (int pair = 1; pair < arr.length; pair++)
            if (arr[pair - 1] > arr[pair]) {
                temp = arr[pair - 1];
                arr [pair - 1] = arr [pair];
                arr[pair] = temp;
            }
} // bubbleSort()

/**
 * print() prints all the integers in the new array of 50
```

```
     * integers created by merging the two 25 integer arrays.
     */
    public void print(int arr[]) {
        for (int k = 0; k < arr.length; k++) {
            System.out.print(arr[k] + ", ");
            if ((k+1) % 10 == 0)
                System.out.println();
        } // for
    } // print()

    /**
    * main() creates a Merge instance and then calls its
    * bubbleSort() method to sort two integer arrays, which
    * contain an arbitrary number of elements. It then passes
    * the two arrays to the mergeSort() method where they are
    * merged into a single sorted array. It then displays the
    * result.
    */
    public static void main(String args[]) {
      int intArr1[] =
        {1230,1,24,3,4,5,6,7,89,9,10,101,12,13,14,152,16,17,242};
      int intArr2[] =
        {110,12,2,3,4,533,6,7,8,9,101,411,12,133,14,15,167};

       Merge merger = new Merge();
       merger.bubbleSort(intArr1);
             // Sort and display the two arrays
       merger.bubbleSort(intArr2);
       merger.print(intArr1);
       System.out.println("\n");
       merger.print(intArr2);
       System.out.println("\n");
         // Create a new array by merging the two arrays
       int intArrNew[] = merger.mergeSort(intArr1, intArr2);
       merger.print(intArrNew);
       System.out.println("\n");
    } // main()
} // Merge
```

22. **Challenge:** Design and implement a `BigInteger` class that can add and subtract positive integers with up to 25 digits. Your class should also include methods for input and output of the numbers. If you're really ambitious, include methods for multiplication and division.

    Answer: The `BigInteger` class contains data structures and methods needed to perform arithmetic on arbitrarily large integers. An `int` array is used to store the number's digits. Addition and subtraction algorithms process the arrays from right to left, performing carries and borrows as necessary. It's important to override the `toString()` method, which can be used in a wide variety of contexts

to display big integers. The following is sample output from the `BigInteger` program:

```
   big1 = 99999999999999999999999999999999999999999999999999
   big2 = 00000000000000000000000000000000000000000000000001
   ----------------------------------------------------------
   sum = 100000000000000000000000000000000000000000000000000

   big3 = 98888888888888888888888888888888888888888888888881
   big4 = 09999999999999999999888764399999999999999999999791
   ----------------------------------------------------------
   diff = 88888888888888888890001244888888888888888888889090
```

```
/*
 * File: BigInteger.java
 * Author: Java, Java, Java
 * Description: This class represents extremely long positive
 * integers, those too long to be stored in either an int or
 * a long.  The digits of  the BigInteger are stored in an int
 * array. One limitation of this class is that it only works
 * for positive integers.
 */

public class BigInteger {
    private int[] bigNum;  // The array of digits
    private int length = 0;  // This number's length
    private String bigNumStr;  // A string representation

   /**
     * BigInteger() is a constructor that takes a String and
     * enters the digits into an array of integers
     * @param s is the String passed to the method to be
     *    entered into the array
     */
    public BigInteger(String s) {
        bigNumStr = s;
        length = s.length();
        bigNum = new int[length];
        for (int k = 0; k < s.length(); k++) {
            bigNum[k] = (int) (s.charAt(k) - '0');
        }
    } // BigInteger()

   /**
     * BigInteger(int) constructor sets the size of the
     *    BigInteger from its parameter
     * @param n gives the size of the BigInteger
     */
    public BigInteger(int n) {
        bigNum = new int[n];
```

```
        length = n;
    }

    /**
     * toString() returns a String representation of this
     * BigInteger, creating a new String from the bigNum
     * array if necessary.
     */
     public String toString() {
         if (bigNumStr ==null) {
             StringBuffer buff = new StringBuffer();
             for (int k = 0; k < bigNum.length; k++)
                 buff.append(bigNum[k] + "");
                 bigNumStr = buff.toString();
         }
         return bigNumStr;
     }

     /**
      * add() computes the sum of this big integer and its
      *    parameter
      * @param  num -- a BigInteger to be added to this
      * @return -- a BigInteger giving the sum of this + num
      * Algorithm: Create a new BigIntger with room for 1
      *  digit more than the longer of this and num. Then,
      *  proceeding right to left add a digit from this and
      *  a digit from num plus a carry. Then update the carry.
      *  Since this and num can be different lengths, treat
      *  the cases where one or the other or both run out
      *  of digits as special cases.
      */
     public BigInteger add(BigInteger num) {
         int digits = Math.max(this.length, num.length);
         BigInteger result = new BigInteger(digits + 1);
         int carry = 0;
            // Pointer to rightmost digit of this integer
         int j = length -1;
           // Pointer to rightmost of num
         int k = num.length -1;

         for (int i = digits; i >= 0; i--) {
             //If both are out of digits just add the carry
             if (j < 0 && k < 0) {
                 result.bigNum[i] = carry;
             } else if (j < 0) { //If this is out of digits
                 result.bigNum[i] =
                             (carry + num.bigNum[k]) % 10;
                 carry = (carry + num.bigNum[k]) / 10;
                 k--;
             } else if ( k < 0) { //If num is out of digits
```

```
            result.bigNum[i] =
                       (carry + this.bigNum[j]) % 10;
            carry = (carry + this.bigNum[j] / 10);
            j--;
        } else {    // If both have digits left
            result.bigNum[i] =
              (carry+this.bigNum[j]+num.bigNum[k])%10;
            carry =
              (carry+this.bigNum[j]+num.bigNum[k])/10;
            j--;
            k--;
        }
    }
    return result;
} // add()

/**
 * subtract() subtracts num from this big integer
 * @param  num -- a BigInteger to be subtracted
 *   from this
 * @return -- a BigInteger giving the sum of this
 *   + num
 * Pre: this integer is >= num
 * Algorithm: Create a new BigInteger with the same
 * number of digits as this BigInteger. Then
 * proceeding from right to left, compute the
 * difference between this and num. If the difference
 * is negative, borrow from the from the next digit
 * of this BigInteger.
 */
public BigInteger subtract(BigInteger num) {
    BigInteger result = new BigInteger(length);
    int diff = 0;
      // Pointer to rightmost digit of this integer
    int j = length -1;
      // Pointer to rightmost of num
    int k = num.length -1;

    for (int i = length-1; i >= 0; i--) {
        if ( k < 0) {   // If num is out of digits
            result.bigNum[i] = this.bigNum[j];
            j--;
        } else {     // If both have digits left
            diff = this.bigNum[j] - num.bigNum[k];
            if (diff < 0) { //If a borrow is necessary
               diff += 10;   //borrow from next digit
               this.bigNum[j-1] -= 1;
            }
            result.bigNum[i] = diff;
           j--;
```

```
                k--;
            }
        }
        return result;
    } // subtract()

} // BigInteger
```

23. **Challenge:** Design a data structure for this problem: As manager of Computer Warehouse, you want to keep track of the dollar amount of purchases made by those clients that have regular accounts. The accounts are numbered from `0, 1, ..., N`. The problem is that you don't know in advance how many purchases each account will have. Some may have one or two purchases. Others may have 50 purchases.

    Answer: The `Account` class uses an array of `double` to store purchase amounts. Each account also has a unique account number, which is generated by a `static` integer defined in the class.

```
/*
 * File: Account.java
 * Author: Java, Java, Java
 * Description: This class defines a data structure for
 *  an account that keeps track of purchases made by
 *  individual customers. The accounts are numbers 0, 1, ...
 *  An array is used to store purchase amounts.
 */
import java.util.*;

public class Account {
    private static final int MAXPURCHASES = 100;
                // Maximum number of purchases per account
    private static int nAccounts;     // Number of accounts

    private int accountNum;  // This object's account number
    private double purchases[] = new double[MAXPURCHASES];
           // This stores the purchases on
           // the account as an array of doubles
    private int numPurchases = 0;
           // This keeps tracks of the number of purchases on
           // this account

    /**
     * Account() constructor gives this instance a unique
     * account number by incrementing the static variable
     * nAccounts.
     */
    public Account() {
        accountNum = nAccounts++;
```

```
    } // Account()

    /**
     * addPurchase() adds a purchase to the this account.
     * @param newPurchase - a double representing the value
     *   of the purchase
     */
    public void addPurchase(double newPurchase) {
         purchases[numPurchases] = newPurchase;
         numPurchases++;
    } // addPurchase()

    /**
     * toString() returns a string representation of the
     *  purchases that have been added to the account.
     */
    public String toString() {
        StringBuffer result = new StringBuffer();
        result.append("Account Number " + accountNum +
                  " has " + numPurchases + " purchases.\n");
        if (numPurchases != 0) {
            result.append("They are in the amounts of:\n");
            for (int k = 0; k < numPurchases; k++)
                  result.append("$" + purchases[k] + "\n");
        }
        return result.toString();
    } // acctToString()

    /**
     * acctToString() is static and returns a string
     * representation of the total number of accounts that
     * have been created.
     */
    public static String acctToString() {
        return "There are a total of " + nAccounts +
                                          " accounts.\n";
    } // acctToString()
} // Account
```

24. An *anagram* is a word made by rearranging the letters of another word. For example, “act” is an anagram of “cat”, and “aegllry” is an anagram of “allergy.” Write a Java program that accepts two words as input and determines if they are anagrams.

    Answer: The `Anagram` class implements the `areAnagrams()` method to determine if two strings are anagrams of each other. Two strings are anagrams of each other if, after sorting each of them, they are equal.

```
/*
 * File: Anagram.java
```

```
 * Author: Java, Java, Java
 * Description: This class analyzes Strings and determines
 * if they are anagrams of each other. To determine if
 * two strings are anagrams, sort both strings and see
 * if they are equal. A version of bubblesort is used
 * for the sort.
 */
public class Anagram {
    /**
     * areAnagrams() is a method that analyzes two
     * Strings and determines if they are anagrams of each
     * other. It sorts the letters of the strings and test
     * if the results are equal.
     * @param str1 is one of the Strings to analyze
     * @param str2 is the second String to analyze
     * @return is the boolean true if the Strings  are
     * anagrams
     *       and false otherwise
     */
    public boolean areAnagrams (String str1, String str2) {
        String s1 = stringSort(str1);
        String s2= stringSort(str2);
        return s1.equals(s2);
    } // areAnagrams()

    /**
     * stringSort() sorts a string of characters into
     * alphabetical order. It uses the bubblesort algorithm
     * @param s -- the string to be sorted
     * @return a string whose letters are in alphabetical
     * order
     */
    public String stringSort(String s) {
        StringBuffer buff = new StringBuffer(s);
        for (int pass = 1; pass < buff.length(); pass++)
            for (int pair = 1; pair < buff.length(); pair++)
                if (buff.charAt(pair-1) > buff.charAt(pair)) {
                    char temp = buff.charAt(pair-1);
                    buff.setCharAt(pair-1, buff.charAt(pair));
                    buff.setCharAt(pair, temp);
                }
        return buff.toString();
    } // stringSort()

    public static void main(String argv[]) {
        Anagram tester = new Anagram();
        if (tester.areAnagrams("act", "cat"))
            System.out.println("act and cat are anagrams");
        if (tester.areAnagrams("act", "cate"))
            System.out.println("act and cate are anagrams");
```

```
    } // main()

} // Anagram
```

25. **Challenge:** An *anagram dictionary* is a dictionary that organizes words together with their anagrams. Write a program that lets the user enter up to 100 words (in a `JTextField`, say). After each word is entered, the program should display (in another `JTextField` perhaps) the complete anagram dictionary for the words entered. Use the following sample format for the dictionary. Here the words entered by the user were: felt, left, cat, act, opt, pot, top.

```
act:  act cat
eflt:  felt left
opt:   opt pot top
```

Answer: This program uses four classes. (1) The `AnagramApplet` class serves as a graphical user interface. In accepts a string of input in one `JTextArea` and displays the resulting anagram dictionary in a second `JTextArea`. (2) A modified version of the `Anagram` class from the previous exercise is used to determine when two words are anagrams of each other. For this problem we must add a constructor and a `toString()` method to this class. (3) The `Dictionary` class uses a `java.util.Vector` to store entries in dictionary form. (4) The `Entry` class defines an individual dictionary entry, which consists of an anagram (e.g., eflt) and a list of words that match that anagram.

```
/*
 * File: Dictionary.java
 * Author: Java, Java, Java
 * Description: This class stores the anagram dictionary as
 *  a vector of Entries
 */
import java.util.*;

public class Dictionary {
    private Vector entries = new Vector();


    /**
     * Dictionary() breaks the String into tokens and inserts
     *  each token into the dictionary.
     * @param s is the String the dictionary is made from
     */
    public Dictionary (String s) {
        StringTokenizer words = new StringTokenizer(s);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            insert(word);
```

```
        }
    } // Dictionary

    /**
     * insert() adds the String to an entry if it matches an
     *  Entry already in the dictionary and creates a new
     * Entry if there are no matches
     * @param s is the String being inserted
     */
    public void insert(String s) {
        int k = 0;
        boolean entryfound = false;
        while (!entryfound && k < entries.size()) {
            Entry o = (Entry)entries.elementAt(k);
            if (o.matches(s)) {
                o.insert(s);
                entryfound = true;
            }
            else
                k++;
        }

        if (!entryfound) {
            Entry e = new Entry(s);
            entries.addElement(e);
        }
    } // insert()
    /**
     * toString() returns the Dictionary as a String
     */
    public String toString() {
        StringBuffer result = new StringBuffer("");
        for (int k = 0; k < entries.size(); k++)
          result.append(entries.elementAt(k).toString()+"\n");
        return result.toString();
    } // toString()


} // Dictionary

/*
 * File: Entry.java
 * Author: Java, Java, Java
 * Description: This class stores an entry in the Dictionary
 *  as a Vector
 */
import java.util.*;

public class Entry {
    Anagram anagram;
```

```
    private Vector words = new Vector();

    /**
     * Entry() creates a new Entry
     * @param s is the String that the new entry is created
     * from
     */
    public Entry (String s) {
        anagram = new Anagram(s);
        words.addElement(s);
    } // Entry()

    /**
     * toString() returns the Entry as a String
     */
    public String toString() {
        StringBuffer result =
            new StringBuffer(anagram.toString() + ": ");
        for (int k = 0; k < words.size(); k++) {
            Object o = words.elementAt(k);
            result.append(o.toString() + ", ");
        }
        return result.toString();
    } // toString()

    /**
     * matches() checks whether a one String is an anagram
     *  of the Entry
     * @param s is the String being tested
     */
    public boolean matches(String s) {
       return anagram.matches(s);
    } // matches()

    /**
     * insert() adds a new String to the Vector which the
     * entry's words are stored in
     * @param s is the String being added to the Vector
     */
    public void insert(String s) {
        words.addElement(s);
    } // insert()
} // Entry

/*
 * File: Anagram.java
 * Author: Java, Java, Java
 * Description: This class analyzes Strings and
 *  determines if they are anagrams of each other. To
 *  determine if two strings are anagrams, sort both
```

```
 *  strings and see if they are equal. A version of
 *  bubblesort is used for the sort.
 */
public class Anagram {
     private String anagram;

    /**
     * Anagram() creates a new anagram from a String.
     * @param s is the String
     */
     public Anagram (String s) {
         anagram = stringSort(s);
     } // Anagram()

    /**
     * toString() returns the anagram as a String
     */
    public String toString() {
        return anagram;
    } // toString()

    /**
     * areAnagrams() is a method that analyzes two
     * Strings and determines if they are anagrams
     * of each other. It sorts the letters of the
     * strings and test if the results are equal.
     * @param str1 is one of the Strings to analyze
     * @param str2 is the second String to analyze
     * @return is the boolean true if the Strings
     *     are anagrams and false otherwise
     */
    public boolean areAnagrams (String str1, String str2) {
        String s1 = stringSort(str1);
        String s2= stringSort(str2);
        return s1.equals(s2);
    } // areAnagrams()

    /**
     * matches() is a method that analyzes a String and
     * determines  if it is an anagram of the String
     * stored in the Anagram object
     * @param s is the String being analyzed
     * @return is the boolean true if the Strings are
     *    anagrams and false otherwise
     */
    public boolean matches(String s) {
       return areAnagrams(s, anagram);
    } // matches()

    /**
```

```
     * stringSort() sorts a string of characters into
     * alphabetical order. It uses the bubblesort algorithm
     * @param s -- the string to be sorted
     * @return a string whose letters are in alphabetical
     *   order
     */
    public String stringSort(String s) {
        StringBuffer buff = new StringBuffer(s);
        for (int pass = 1; pass < buff.length(); pass++)
          for (int pair = 1; pair < buff.length(); pair++)
            if (buff.charAt(pair-1) > buff.charAt(pair)) {
              char temp = buff.charAt(pair-1);
              buff.setCharAt(pair-1, buff.charAt(pair));
              buff.setCharAt(pair, temp);
            } // if
        return buff.toString();
    } // stringSort()

} // Anagram

/*
 * File: AnagramApplet.java
 * Author: Java, Java, Java
 * Description: This applet provides a GUI for
 *  the Dictionary class. It lets the user type text into
 *  a JTextArea and then makes an anagram dictionary
 *  out of the String entered which is then displayed.
 */
import java.util.*;
import javax.swing.*;
import java.awt.event.*;

public class AnagramApplet extends JApplet
                          implements ActionListener {
    private JLabel prompt;
    private JTextArea inputArea;
    private JButton button;
    private static JTextArea outputArea;

    /**
     * init() sets up the applet's interface
     */
    public void init(){
        prompt =
          new JLabel("Enter a String to make a Dictionary:");
        inputArea = new JTextArea(3,30);
        inputArea.setEditable(true);
        outputArea = new JTextArea(10,30);
        button = new JButton("Make Dictionary");
        button.addActionListener(this);
```

```
        getContentPane().add("North", prompt);
        getContentPane().add("Center", inputArea);
        getContentPane().add("East", button);
        getContentPane().add("South", outputArea);
        setSize(400,400);
    } // init()

    /**
     * actionPerformed() gets the text from the input
     * TextAreaand makes it into an anagram dictionary
     * which it displays
     */
    public void actionPerformed(ActionEvent e){
        outputArea.setText("");
        String inputString =
                    new String(inputArea.getText());
        Dictionary dict = new Dictionary(inputString);
        outputArea.setText(dict.toString());
    } // actionPerformed()
} // AnagramApplet
```

26. The Acme Trucking company has hired you to write software to help dispatch its trucks. One important element of this software is knowing the distance between any two cities that it services. Design and implement a `Distance` class that stores the distances between cities in a two-dimensional array. This class will need some way to map a city name, "Boise," into an integer that can be used as an array subscript. The class should also contain methods that would make it useful for looking up the distance between two cities. Another useful method would tell the user the closest city to a given city.

    Answer: The `Distance` class uses a two-dimensional array to store distances between pairs of cities. A one dimensional array is used to store the names of the cities. Given any two city names, the lookup algorithm converts the names to integers, and then uses the integers as indices into the distance table. The distance between the two cities will be distance stored at `distances[j][k]`, where `j` and `k` are the cities' indices.

```
/* File: Distance.java
 * Author: Java, Java, Java
 * Description: This class analyzes the distances
 * between different cities. The main data structure for
 * this program is a two-dimensional table storing the
 * distances between pairs of cities. The cities,
 * represented by numbers between 0 and MAXCITIES-1, will
 * serve as indices to the table. The table's diagonal will
 * contain all 0s, since that represents all the distances
 * between a city and itself. The  names of cities are
 * stored in a one-dimensional array.
 */
```

```
public class Distance {
    private static final int MAXCITIES = 5;
    private double[][] distances =
         {{0,   3000,2000,2900,2950}, // Distances
             {3000,0   ,1000,105 ,150 }, // Table
             {2000,1000,0   ,900 ,1000},
             {2900,105 ,900 ,0   ,100 },
             {2950,150 ,1000,100 ,0   }};

    private String[] cityNames =
      {"Boise","Boston","Chicago","Hartford","New York"};

   /**
     * getCityIndex() gets the index of a city that is
     * stored in the array
     * @param s is the String passed to the method that is
     *   the name of the city
     * @return is the index of the city returned as an
     *   integer
     */
    public int getCityIndex (String s) {
        for (int k = 0; k <= MAXCITIES; k++) {
        if (cityNames[k].equals(s))
            return k;
        }
        return -1;
    } // getCityIndex()

   /**
     * getDistance() analyzes two cities entered into the
     * array and determines the distance between them
     * @param s1 and s2 are the Strings passed to the
     *   method that are the names of the cities
     * @return is the distance between the cities returned
     *   as an double
     */
    public double getDistance(String s1, String s2) {
      return distances[getCityIndex(s1)][getCityIndex(s2)];
    } // getDistance()

   /**
     * getNearestCity() analyzes determines the city that
     * is the nearest to the city passed to the method
     * @param s is the name of the city passed to the
     *   method
     * @return is the name of the city that is closest to
     *   the city passed to the method
     * Algorithm: Initialize a variable called
     * closestDistanceand another one called closest City.
     * Then go through each pairing of this city with some
```

```
     * other city in the table  - all of these will occur
     * in the same row. If a pairing has a distance that is
     * less than closestDistance, make it the new closest
     * city.
     */
    public String getNearestCity (String s) {
          // Initial to a big value
        double closestDistance = 10000;
          // Get this city's index
        int cityIndx = getCityIndex(s);
        int closestCity = cityIndx;
        for (int k = 0; k < MAXCITIES; k++) {
                      // For each city in the table
            if (k != cityIndx) { //If k is not this city
                double distance = distances[cityIndx][k];
                          // Get the distance
                if (distance < closestDistance) {
                         // If this distance is shorter
                    closestDistance = distance;
                        //  Make it the new shortest
                    closestCity = k;
                        //  And remember its city index
                } // if shorter
            } // if
        } // for
        return cityNames[closestCity];
    } // getNearestCity()

    public static void main(String argv[]){
      Distance distance = new Distance();
      System.out.println("The nearest city to Boise is "
                       + distance.getNearestCity("Boise"));
      System.out.println("The nearest city to New York is "
                    + distance.getNearestCity("New York"));
      System.out.println("The nearest city to Hartford is "
                    + distance.getNearestCity("Hartford"));
      System.out.println("The nearest city to Boston is "
                     + distance.getNearestCity("Boston"));
      System.out.println("The distance between Boston and "
        +"Boise is "+distance.getDistance("Boston","Boise"));
      System.out.println("The distance between Boston and "
        +"Hartford is "+distance.getDistance("Boston","Hartford"));
        System.out.println("The distance between Boston and "
        +"New York is "+distance.getDistance("Boston","New York"));
        System.out.println("The distance between Hartford and "
        +"Hartford is "+distance.getDistance("Hartford","Hartford"));
    } // main()
} // Distance
```

27. Rewrite the `main()` method for the `WordGuess` example so that it allows the

user to input the number of players and whether each player is a computer or a human. Use a `KeyboardReader`.

Answer: The entire `WordGuess` and `WordGuesser` classes that include the changes referred to in Chapter 9 are included below. The `WordGuesser` class includes a 3 parameter constructor that was not explicitly described in chapter 9. To compile and run this program, `WordGuess` must have access to the `WordGuesser, ComputerGame, Player, IGame, CLUIPlayableGame, UserInterface`, and `KeyboardReader` classes and interfaces.

```
/*
 * File: WordGuess.java
 * Author: Java, Java, Java
 * Description:  WordGuess  extends the ComputerGame class
 * and implements the CLUIPlayableGame interface, which
 * enables any number of players to play using a command-line
 * interface.  It makes extensive use of Java's inheritance and
 * polymorphism mechanisms.
*/
public class WordGuess extends ComputerGame
                               implements CLUIPlayableGame {
    private String secretWord;   // The word being guessed
        // The state of the current guesses
    private StringBuffer currentWord;
       // A string of the previous guesses
    private StringBuffer previousGuesses;
    private int unguessedLetters;   // Unused letters

    /**
     * WordGuess() default constructor initializes the game.
     * Sets number of players to one.
     */
    public WordGuess() {
        super(1);
        init();
    } // WordGuess()

    /**
     * WordGuess(m) a constructor which initializes the game.
     * Sets number of players to m.
     */
    public WordGuess(int m) {
        super(m);
        init();
    } // WordGuess(m)

    /**
     * init() initializes the game.
     * Used in both constructors
     */
    public void init() {
```

```
        secretWord = getSecretWord();
        currentWord = new StringBuffer(secretWord);
        previousGuesses = new StringBuffer();
        for (int k = 0; k < secretWord.length(); k++)
            currentWord.setCharAt(k,'?');
        unguessedLetters = secretWord.length();
    } // init()

    /**
     * getPreviousGuesses() returns a string containing
     *  all previous letter guesses
     * @return a String containing the guesses
     */
    public String getPreviousGuesses() {
        return previousGuesses.toString();
    } // getPreviousGuesses()

    /**
     * getCurrentWord() returns the current state of
     * the guessed word
     */
    public String getCurrentWord() {
        return currentWord.toString();
    } // getCurrentWord()

    /**
     * getSecretWord() returns a secret word
     */
    private String getSecretWord() {
        int num = (int)(Math.random()*10);
        switch (num)
        {   case 0: return "SOFTWARE";
            case 1: return "SOLUTION";
            case 2: return "CONSTANT";
            case 3: return "COMPILER";
            case 4: return "ABSTRACT";
            case 5: return "ABNORMAL";
            case 6: return "ARGUMENT";
            case 7: return "QUESTION";
            case 8: return "UTILIZES";
            case 9: return "VARIABLE";
            default: return "MISTAKES";
        } //switch
    } // getSecretWord()

    /**
     * guessLetter() returns true if the letter is in
     * the secret word and returns false otherwise
     * @param letter is the letter being guessed
     */
```

```
private boolean guessLetter(char letter) {
    previousGuesses.append(letter);
    if (secretWord.indexOf(letter) == -1)
        return false; // letter is not in secretWord
    else // find positions of letter in secretWord
    {   for (int k = 0; k < secretWord.length(); k++)
        {   if (secretWord.charAt(k) == letter)
            {   if (currentWord.charAt(k) == letter)
                    return false; ////already guessed
                currentWord.setCharAt(k,letter);
                unguessedLetters--; //one less to find
            } //if
        } //for
        return true;
    } //else
} //guessLetter()

/**
 * getRules() is overridden from the TwoPlayerGame
 * class. It returns a String giving the games rules.
 * @return a String describing the game
 */
public String getRules() {
    return "\n*** The Rules of Word Guess ***\n" +
    "(1) The game generates a secret word.\n" +
    "(2) Two players alternate taking moves.\n" +
    "(3) A move consists of guessing a letter in " +
                                    "the word.\n" +
    "(4) A player continues guessing until a letter "
                                   + "is wrong.\n" +
    "(5) The game is over when all letters of the " +
                                "word are guessed\n" +
    "(6) The player guessing the last letter of the "
                                    + "word wins.\n";
} //getRules()

/**
 * gameOver() is defined as abstract in TwoPlayerGame
 * and implemented here. A WordGuess game is over
 * when all the letters in the word have been guessed
 * @return true if the game is over and false otherwise
 */
public boolean gameOver() { // From TwoPlayerGame
    return (unguessedLetters <= 0);
} // gameOver()

/**
 * getWinner() is defined as abstract in TwoPlayerGame
 *  and implemented here. It defines who wins WordGuess.
 * @return a String describing the winner
```

```
 */
public String getWinner() {  // From TwoPlayerGame
    if (gameOver())
        return "Player " + getPlayer();
    else return "The game is not over.";
} // getWinner()

/**
 * reportGameState() is implemented as part of
 * the CLUIPlayableGame interface. It describes
 * the current state of the game.
 * @return a String describing the game's current
 *   state.
 */
public String reportGameState() {
    if (!gameOver())
      return "\nCurrent word " + currentWord.toString()
        + " Previous guesses " + previousGuesses +
        "\nPlayer " + getPlayer() + " guesses next.";
    else
      return "\nThe game is now over! The secret word is "
      + secretWord + "\n" + getWinner() + " has won!\n";
} // reportGameState()

/**
 * getGamePrompt() is implemented as part of the
 * CLUIPlayableGame interface. It defines the prompt
 * presented to the user before each move.
 * @return a String giving the prompt.
 */
public String getGamePrompt() {
    return "\nGuess a letter that you think is in the "
                                    + "secret word: ";
} // getGamePrompt()

/**
 * submitUserMove() defines one move of the WordGuess
 * game.
 * @param s is a String giving the move, in this case
 *   a single letter at s.charAt(0) which is the
 *   guessed letter.
 * @return a message describing the outcome of the move
 */
public String submitUserMove(String s) {
    char letter = s.toUpperCase().charAt(0);
    if (guessLetter(letter)) {   //if correct
        return "Yes, the letter " + letter +
                               " IS in the secret word\n";
     } else {
         changePlayer();
```

```
          return "Sorry, "+letter+" is NOT a " +
                      "new letter in the secret word\n";
      }
} // submitUserMove()

/**
 * play() is implemented as part of the CLUIPlayableGame
 * interface. It contains the control algorithm for
 * playing the WordGuess game.
 * @param ui gives a reference to the UserInterface where
 *   I/O is performed.
 */
public void play(UserInterface ui) {
    ui.report(getRules());
    ui.report(listPlayers());
    ui.report(reportGameState());

    while(!gameOver()) {
      WordGuesser p = (WordGuesser)player[whoseTurn];
      if (p.isComputer()){
        ui.report(submitUserMove(p.makeAMove(getGamePrompt())));
      } else {              // otherwise, user's turn
          ui.prompt(getGamePrompt());
          ui.report(submitUserMove(ui.getUserInput()));
      } // else
      ui.report(reportGameState());
    } // while
} //play()

/**
 * main() plays the WordGuess game
*/
public static void main(String args[])  {
    KeyboardReader kb = new KeyboardReader();
    int num = -1;
    while (num <= 0) {
        kb.prompt("Input a positive integer for "
                        + "the number of players:");
        num = kb.getKeyboardInteger();
    } // while
    WordGuess game = new WordGuess(num);
    int choice;
    for (int k = 0; k < num; k++){
      choice = - 1;
      while ((choice < 0)||(choice > 1)) {
        kb.prompt("Choose type of player for " +
                                    "player " + k);
        kb.prompt("\nInput 0 for human or 1 " +
                                   "for computer:");
        choice = kb.getKeyboardInteger();
```

```
            } // while
            if (choice == 0)
              game.addPlayer(new WordGuesser(game, k,
                                        Player.HUMAN));
            else
              game.addPlayer(new WordGuesser(game, k,
                                      Player.COMPUTER));
          } // for
          game.play(kb);
      } //main()
  } //WordGuess class

  /*
   * File: WordGuesser.java
   * Author: Java, Java, Java
   * Description: Defines an Player for the WordGuess game.
   * appropriate It implements the makeAMove() method in a
   * way that is for a n-player WordGuess game. In this
   * case the Player chooses a random letter, which is not
   * a "good" way to play the game.
   */
  public class WordGuesser extends Player {
      private WordGuess game;

      /**
       * WordGuesser() constructor is given a reference to
       * the WordGuess game.
       * @param a reference to the WordGuess game
       */
      public WordGuesser (WordGuess game, int id, int kind) {
          super(id, kind);
          this.game = game;
      }

      /**
       * makeAMove() is defined as part of the Player
       * interface. It defines a valid move in the
       * WordGuess game. This version simply picks a
       * random letter from the alphabet.
       * @param prompt is a string containing a prompt
       * @return a string describing the move
       */
      public String makeAMove(String prompt)  {
          String usedLetters = game.getPreviousGuesses();
          char letter;
          do {  // Pick one of 26 letters
            letter = (char)('A'+(int)(Math.random()*26));
          } while (usedLetters.indexOf(letter) != -1);
          return "" + letter;
      }
```

```
    /**
     * toString() provides a string representation of
     * this object.
     */
    public String toString() { // returns 'WordGuesser'
        String className = this.getClass().toString();
        return className.substring(5);
    }
} // WordGuesser
```

28. Write a smarter version of the `WordGuesser` class that knows which letters of the English language are most frequent. *Hint*: Rather than use random guesses, store the players guesses in a string in order of decreasing frequency: "ETAONRISHLGCMFBDGPUKJVWQXYZ".

    Answer: The `SmartGuesser` class listed below is a modification of the `WordGuesser` class with the major changes in the `makeAMove()` method. Some obvious changes need to be made in the `play()` and `main()` methods the `WordGuess` class in order to test this class.

```
/**
 * File: SmartGuesser.java
 * Author: Java, Java, Java
 * Description: Defines a Player for the WordGuess game.
 * It implements the makeAMove() method in a way that is
 * appropriate for a n-player WordGuess game. In this case
 * the Player chooses the most frequently used letter in
 * English among the unguessed letter in this game.
 */
public class SmartGuesser extends Player {
    private WordGuess game;
    private final String ENGLFREQ =
                          "ETAONRISHLGCMFBDGPUKJVWQXYZ";

    /**
     * SmartGuesser() constructor is given a reference to
     * the WordGuess game.
     * @param a reference to the WordGuess game
     */
    public SmartGuesser (WordGuess game, int id, int kind) {
        super(id, kind);
        this.game = game;
    }

    /**
     * makeAMove() is defined as part of the Player
     * interface. It defines a valid move in the WordGuess
     * game. This version  picks the most frequent letter
     * in the English language that has not yet been guessed.
```

```
     * @param prompt is a string containing a prompt
     * @return a string describing the move
     */
    public String makeAMove(String prompt)  {
        String usedLetters = game.getPreviousGuesses();
        char letter;
        int freqIndex = 0;
        do {  // Pick the most frequent unguessed letter
            letter = ENGLFREQ.charAt(freqIndex);
            freqIndex++;
        } while ((usedLetters.indexOf(letter) != -1) &&
                                        (freqIndex < 26));
        return "" + letter;
    } // makeAMove()

    /**
     * toString() provides a string representation of this
     * object.
     */
    public String toString() {
        if (kind == Player.HUMAN) return "Human User";
        else return "SmartGuesser";
    } // toString()
} // SmartGuesser
```

29. Write a `CLUI` version of the `SlidingTilePuzzle`. You will need to make modifications to the `SlidingTilePuzzle` class.

    Answer: The modified `SlidingTilePuzzler` class and the `TilePuzzlePlayer` class listed below provide an answer to the exercise.

```
/*
 * File: SlidingTilePuzzle.java
 * Author: Java, Java, Java
 * Description: A modification of the SlidingTile puzzle in
 * the text to implement the CLUIPlayableGame interface
 * instead of GUIPlayable. The puzzle consists of 7 tiles.
 * Three are labelede R and three are labeled  L. The seventh
 * tile is blank. The user can move a tile into the blank or
 * can have a tile jump over one or two other tiles into the
 * blank space. The goal is to rearrange the puzzle so that
 * all the Ls are to the left of all the Rs and the blank is
 * in the middle.
 */

public class SlidingTilePuzzle extends ComputerGame
                              implements  CLUIPlayableGame {
    private char puzzle[] = {'R','R','R',' ','L','L','L'};
       // The goal of the puzzle
    private String solution = "LLL RRR";
```

```
private int blankAt = 3;
private boolean userGaveUp = false;

/**
 * SlidingTilePuzzle() default constructor sets up a
 * 1-player game.
 */
public SlidingTilePuzzle() { super(1); }

/**
 * gameOver() is abstract in the ComputerGame class
 * and implemented here. The sliding tile puzzle is
 * finished when the puzzles arrangement matches
 * the solution variable.
 * @return true is returned if the puzzle is solved.
 */
public boolean gameOver() { // True if puzzle solved
    StringBuffer sb = new StringBuffer();
    sb.append(puzzle);
    return (sb.toString().equals(solution) ||
                                        userGaveUp);
} // gameOver()

/**
 * getWinner() is abstract in the ComputerGame class
 * and implemented here. It returns a string reporting
 * whether the puzzle is solved.
 */
public String getWinner() {
    if (userGaveUp)
      return "The puzzle is not solved, you gave up.\n";
    else if (gameOver())
      return "\n Very Nice!You solved the puzzle!\n";
    else
      return "\nGame not over. Make another move.\n";
} // getWinner()

/**
 * reportGameState() reports the current state of the
 * game as a String
 * @param a String representing the current game state.
 */
public String reportGameState() {
  StringBuffer sb = new StringBuffer();
  sb.append(puzzle);
  sb.append(" - The state of the SlidingTilesPuzzle\n");
  sb.append("0123456 -- number labels for the " +
                                "positions of tiles.\n");
    return sb.toString();
} // reportGameState()
```

```
/**
 * getGamePrompt() returns the prompt for this game.
 */
public String getGamePrompt() {
  String str = "To move a tile, input the number " +
                            "label of its position.\n";
  str = str + " or input -1 to give up the game as lost:";
  return str;
} //prompt()

/**
 * submitUserMove() is from the CLUIPlayableGame
 * interface. It takes a user move and processes it and
 * returns a response.
 */
public String submitUserMove(String usermove) {
    int tile = Integer.parseInt(usermove);
    if (tile == -1){
        userGaveUp = true;  // And exit the method
        return "You have given up.\n\n";
    } // if
       // Assume tile is a tile position.
    char ch = puzzle[tile];
    String comment;  // To store the return string.
    if (ch=='L' && (blankAt==tile-1 ||
                                    blankAt==tile-2)){
        swapTiles(tile,blankAt);
        comment = "That move is legal.\n\n";
    } else if (ch=='R' && (blankAt==tile+1 ||
                                    blankAt==tile+2)){
        swapTiles(tile,blankAt);
        comment = "That move is legal.\n\n";
    } else
        comment = "That's an illegal move.\n\n";
    return comment;
} // submitUserMove()

/**
 * swapTiles() reverses the labels on two tiles, one
 * of which is the blank tile.
 * @param t1 an int giving the labeled tile
 * @param bl an int representing the blank tile
 */
private void swapTiles(int ti, int bl) {
    char ch = puzzle[ti];
    puzzle[ti] = puzzle[bl];
    puzzle[bl] = ch;
    blankAt = ti;   // Reset the blank
} // swapTiles()
```

```
/**
 * getRules() overrides a method of the ComputerGame
 * class. It contains the control algorithm for
 * playing the SlidingTilePuzzle game.
 * @return - a string that describes the rules of
 *   the puzzle.
 */
public String getRules() {
    StringBuffer sb = new StringBuffer();
    sb.append("The rules of this puzzle are:\n");
    sb.append("(1) L-tiles can only move left.\n");
    sb.append("(2) R-tiles can only move right.\n");
    sb.append("(3) A tile can move one space into " +
                              "the blank space.\n");
    sb.append("(4) A tile can jump over one tile " +
                           "into the blank space.\n");
    sb.append("(5) Tiles can make no other moves.\n");
    sb.append("(6) The winning position is: LLL_RRR\n");
    return sb.toString();
} // getRules()

/**
 * play() is implemented as part of the CLUIPlayableGame
 * interface. It contains the control algorithm for
 * playing the SlidingTilePuzzle game.
 * @param ui gives a reference to the UserInterface
 *   where I/O is performed.
 */
public void play(UserInterface ui) {
    ui.report(getRules());
    ui.report(listPlayers());
    ui.report(reportGameState());

    while(!gameOver()) {
      Player p = player[whoseTurn];
      if (p.isComputer()){
        ui.report(submitUserMove(p.makeAMove(getGamePrompt())));
      } else {              // otherwise, user's turn
        ui.prompt(getGamePrompt());
        ui.report(submitUserMove(ui.getUserInput()));
      } // else
      ui.report(reportGameState());
    } // while
    ui.report(getWinner());
} //play()

/**
 * main() plays the SlidingTilePuzzle game
*/
```

```
    public static void main(String args[])  {
        KeyboardReader kb = new KeyboardReader();
        SlidingTilePuzzle game = new SlidingTilePuzzle();
        game.addPlayer(new
                  TilePuzzlePlayer(game, 0, Player.HUMAN));
        game.play(kb);
    } //main()

} //SlidingTilePuzzle class

/*
 * File: TilePuzzlePlayer.java
 * Author: Java, Java, Java
 * Description: Defines a Player for the SlidingTileGame
 * game. Assumes that the SlidingTilePuzzle game will
 * always be played by a single human user.  The constructor
 * will be used in the play() method implemented as part of
 * the CLUIPlayable interface It implements the makeAMove()
 *  method with trivial body.
 */
public class TilePuzzlePlayer extends Player {
    private SlidingTilePuzzle game;

    /**
     * TilePuzzlePlayer() constructor is given a reference
     * to the WordGuess game.
     * @param a reference to the WordGuess game
     */
    public TilePuzzlePlayer (SlidingTilePuzzle game,
                                     int id, int kind) {
        super(id, kind);
        this.game = game;
    } // TilePuzzlePlayer() constructor

    /**
     * makeAMove() is defined as part of the Player
     * interface. A trivial implementation assumes the
     * method will not be called.
     * @param prompt is a string containing a prompt
     * @return the string "makeMove() trivially
     *   implemented"
     */
    public String makeAMove(String prompt)  {
        String str = "makeAMove() trivially implemented";
        return str;
    } // makeAMove()

    /**
     * toString() provides a string representation of
     * this object.
```

```
         */
        public String toString() {
            if (kind == Player.HUMAN) return "Human User";
            else return "TilePuzzlePlayer";
        } // toString()
    } // TilePuzzlePlayer class
```

Chapter 10

Exceptions: When Things Go Wrong

1. Explain the difference between the following pairs of terms.

   (a) *Throwing an exception* and *catching an exception*.

   Answer: *Throwing an exception* occurs when an exceptional condition is detected. It uses a `throw` statement. *Catching an exception* is the process of handling an exception.

   (b) *Try block* and *catch block*.

   Answer: A *try block* includes statements that might throw an exception. A *catch block* includes code that handles a thrown exception.

   (c) *Catch block* and *finally block*.

   Answer: A *catch block* is executed when a particular type of exception is throws. It includes the code to handle that type of exception. A *finally block* contains statements that should be executed whether or not an exception is thrown.

   (d) *Try block* and *finally block*.

   Answer: A *try block* contains statements that may throw an exception. *finally block*. A *finally block* contains statements that should be executed whether or not an exception is thrown.

   (e) *Dynamic scope* and *static scope*.

   Answer: The *static scope* of a program refers to the program's source code definitions. For example, a statement is contained within the static scope of a method, if it is contained in the method definition. *Dynamic scope* refers to the order in which statements are executed when the program is run. For example, if a method is called from the `main()` method, that method is contained in the dynamic scope of `main()`.

   (f) *Dialog box* and *top-level window*.

Answer: A *top-level window* may exist on its own, whereas a *dialog box* can only be created from a top-level window.

(g) *Checked* and *unchecked* exception.

Answer: A *checked* exception must either be caught or declared within the method in which it may occur. An *unchecked* exception needn't be caught.

(h) *Method stack* and *method call*.

Answer: The *method stack* is a data structure that keeps tracks of the current history of method calls, where a *method call* is a statement that invokes a particular method. When an exception occurs, Java typically prints a trace of the method stack, showing how that erroneous statement was reached.

2. Fill in the blank:

(a) _____ an exception is Java's way of signaling that some kind of abnormal situation has occurred.

Answer: Throwing

(b) The only place that an exception can be thrown in a Java program is within a _____. **Answer: try block**

(c) The block of statements placed within a catch block are generally known as an _____. **Answer: exception handlers**

(d) To determine a statement's _____ scope, you have to trace the program's execution. **Answer: dynamic**

(e) To determine a statement _____ scope, you can just read its definition. **Answer: static**

(f) When a method is called, a representation of the method call is place on the _____. **Answer: method stack**

(g) The root of Java's exception hierarchy is the _____ class. **Answer: Exception**

(h) A _____ exception must be either caught or declared within the method in which it might be thrown. **Answer: checked**

(i) An _____ exception may be left up to Java to handle. **Answer: unchecked**

3. Compare and contrast the four different ways of handling exceptions within a program.

Answer: Generally, there are four ways to handle an exception: (1) Let Java handle it. This way would be recommended if your handling of the exception does not improve significantly upon Java's default handling of it. (2) Fix the problem that led to the exception and resume the program. This approach is warranted only if the error can be fixed and the program resumed, or in the case of a failsafe program. (3) Report the problem and resume the program. This would be the preferred technique for a failsafe program in which the detected error cannot really be fixed. (4) Print an error message and terminate the program. Most erroneous conditions reported by exceptions are difficult or impossible to fix. This approach would usually be used during program development.

4. Suppose you have a program that asks the user to input a string of no more than five letters. Describe the steps you'd need to take in order to design a `StringTooLongException` to handle cases where the user types in too many characters.

   Answer: Create a subclass of `Exception` named `StringTooLongException`. In the method that reads the user's input, code a `try..catch` block. If the user inputs a string longer than 5 letters, throw a `StringTooLongException` in the `try` portion and handle it in the `catch` class.

5. Exceptions require more computational overhead than normal processing. Explain.

   Answer: When an exception is thrown, Java must suspend normal execution. It was create some kind of data structure to store the program's state and then locate and branch to the catch clause that handles the exception. All of this requires time.

6. Suppose the following `ExerciseExample` program is currently executing the if statement in `method2()`. Draw a picture of the `method call stack` that represents this situation.

```
public class ExerciseExample {
    public void method1(int M) {
        try {
            System.out.println("Entering try block");
            method2(M);
            System.out.println("Exiting try block");
        } catch (Exception e) {
            System.out.println("ERROR: " + e.getMessage());
        }
    } // method1()

    public void method2(int M) {
        if (M > 100)
            throw new ArithmeticException(M + " is too large");
    }

    public static void main(String argv[]) {
        ExerciseExample ex = new ExerciseExample();
        ex.method1(500);
    }
} // ExerciseExample
```

   Answer: The method call stack would contain the following entries, from bottom to top:

```
main()
method1()
method2()
```

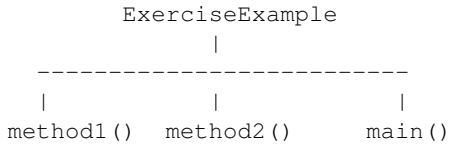
7. Repeat the previous exercise for the situation where the program is currently executing the second `println()` statement in `method1()`.

   Answer: The method call stack would contain the following entries, from bottom to top:

   ```
   main()
   method1()
   ```

8. Draw a hierarchy chart that represents the static scoping relationships among the elements of the `ExerciseExample` program.

   Answer:

9. What would be printed by the `ExerciseExample` program when it is run?

   Answer:

   ```
   The output would be:
       Entering try block
       ERROR: 500 is too large
   ```

10. What would be printed by the `ExerciseExample` program, if the statement in its main method were changed to `ex.method1(5)`?

    Answer:

    ```
    The output would be:
        Entering try block
        Exiting try block
    ```

11. Consider again the `ExerciseExample` program. If the exception thrown were `Exception` rather than `ArithmeticException`, explain why we would get the following error message: `java.lang.Exception must be caught, or it must be declared....`

    Answer: An `ArithmeticException` is a subclass of `RuntimeException` and therefore it is an *unchecked exception*. Unchecked exceptions do not have to be caught or declared in the methods in which they are thrown. On the other hand, an `Exception` include subclasses that are *checked exceptions*, and must therefore be caught or declared in `method2()`.

12. Write a `try/catch` block which throws an `Exception` if the value of variable `X` is less than zero. The exception should be an instance of `Exception` and when it is caught, the message returned by `getMessage()` should be "ERROR: Negative value in X coordinate."

    Answer:

```
try {
    if (X < 0)
        throw new Exception("ERROR: Negative value in X coordinate");
} catch (Exception e) {
    System.out.println("ERROR: " + e.getMessage());
}
```

13. Look at the `IntFieldTester` program and the `IntField` class definition. Suppose the user inputs a value that's greater than 100. Show what the method call stack would look like when the `IntField.getInt()` method is executing the `num > bound` expression.

    Answer:

```
The method call stack would keep track of the method calls leading
to the error. The getInt() method would be on the top:
          Top: IntField.getInt()
               IntFieldTester.actionPerformed()
               IntFieldTester.main()
```

14. As a continuation of the previous exercise, show what the program's output would be if the user output a value greater than 100.

    Answer:

```
 java.lang.Exception.IntOutOfRangeException: The input value
 exceeds the bound 100
    at IntField.getInt(IntField.java:25)
    at IntFieldTester.actionPerformed(IntFieldTester.java:35)
    at IntFieldTester.main(IntFieldTester.java:45)
```

15. As a continuation of the previous exercise, modify the `IntOutOfRangeException` handler so that it prints the message call stack. Then show what it would print.

    Answer: It would print what is shown in the previous exercise. The change to `IntOutOfRangeException` would be to add a call to the `printStackTrace()` method.

```
public class IntOutOfRangeException extends Exception {

    public IntOutOfRangeException(int Bound) {
        super("The input value exceeds the bound " + Bound);
        printStackTrace();
    }
}
```

16. Define a subclass of `RuntimeException` named `InvalidPasswordException`, which contains two constructors. The first constructor takes no parameters and an exception thrown with this constructor should return "ERROR: invalid password" when its `getMessage()` is invoked. The second constructor takes a

single `String` parameter. Exceptions thrown with this constructor should return the constructor's argument when `getMessage()` is invoked.

Answer:

```
/*
 *  File: InvalidPasswordException.java
 *  Author: Java, Java, Java
 *  Description: This Exception subclass is invoked when the
 *   user inputs an invalid password.
 */

public class InvalidPasswordException extends Exception {

    public InvalidPasswordException() {
        super("Error: invalid password");
    }

    public InvalidPasswordException(String s) {
        super(s);
    }
}
```

17. Extend the `IntField` class so that it will constrain the integer `JTextField` to an `int` between both a lower and upper bound. In other words, it should throw an exception if the user types in a value lower than the lower bound or greater than the upper bound.

Answer:

```
/*
 * File: IntField.java
 * Author: Java, Java, Java
 * Description:  This version of IntField has both an upper
 * and lower bound associated with it. The user must type
 * a number within these bounds into the field. A programmer
 * defined exception, IntOutOfRangeException, is thrown if
 * the bound is exceeded. Note that upperBound and
 * lowerBound are set initially to MAX and MIN possible
 * values respectively. This allows the field to accept any
 * numeric input if its first constructor is used.
 */

import javax.swing.*;

public class IntField extends JTextField {
    private int upperBound = Integer.MAX_VALUE;
    private int lowerBound = Integer.MIN_VALUE;

    /**
     * IntField() constructor merely invokes its superclass
```

```
     *  constructor, in this way inheriting all the properties
     *  of a JTextField.
     */
    public IntField (int size) {
        super(size);
    }

    /**
     * This IntField() constructor sets the field's two bounds.
     */
    public IntField(int size, int max, int min) {
        super(size);
        upperBound = max;
        lowerBound = min;
    }

    /**
     * getInt() converts the field's text to an int and returns it.
     * This version also checks that the field's bounds are
     *  not exceeded and raises an exception if one or the
     *  other is. The method also makes clear that
     *  a NumberFormatException could be thrown. This would
     *  happen if the user types a value that is not a valid integer.
     * @return an int representing the integer typed into the text field
     */
    public int getInt() throws NumberFormatException,
                                IntOutOfRangeException {
        int num = Integer.parseInt(getText());
        if (num > upperBound)
            throw new IntOutOfRangeException(upperBound);
        else if (num < lowerBound)
            throw new IntOutOfRangeException(lowerBound);
        return num;
    } // getInt()

} // IntField
```

18. Design Issue: One of the preconditions for the `insertionSort()` method (Text Figure 9.13) is that its array parameter not be null. Of course this precondition would fail if the array were passed a null array reference. In that case Java would throw a `NullPointerException` and terminate the program. Is this an appropriate way to handle that exception?

    Answer: It is appropriate to terminate the program in this case because it is not clear how the program could recover from this error. The program should be modified to so that it checks that the array reference is not null before calling `InsertionSort()`.

19. With respect to the previous exercise, suppose you decide that it is more appropriate to handle the `NullPointerException` by presenting an error dialog.

Modify the method to accommodate this behavior.

```
/*
 *  File: Sort.java
 *  Author: Java, Java, Java
 *  Description: This class implements the Insertion Sort
 *  algorithm. This version makes Sort a subclass of JFrame
 *  thereby giving it a simple GUI interface. This is
 *  necessary in order to open a message dialog when a
 *  NullPointerException is thrown in main().
 *
 *  Note: This example is somewhat contrived, but it
 *  illustrates the separation of exception handling from
 *  exception throwing, and for a GUI application, the best
 *  way to report an exception is to use some kind of message
 *  dialog.
 */

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

public class Sort extends JFrame {

    /**
     * insertionSort() sorts the values in arr into ascending
     * order
     * Pre: arr is not null.
     * Post: The values arr[0]...arr[arr.length-1] will be
     *  arranged in ascending order.
     */
    public void insertionSort(int arr[]) {
        if (arr == null){
            String errstr =
                 "ERROR: InsertionSort() parameter is null";
            throw new NullPointerException(errstr);
        }  // if

        // arr is not null so do an insertion sort
        int temp;    // Temporary variable for insertion
          // begin with second element
        for (int k = 1; k < arr.length; k++) {
            temp = arr[k]; //Remove an element from the array
            int i;
                   // For all array elements larger than temp
                   //  Move array element one place right
            for (i = k - 1;  i >= 0 && arr[i] > temp; i--)
                arr[i + 1] = arr[i];
            arr[i + 1]  = temp;
        } // for
```

```
    } // insertionSort()

    /**
     * print() prints the values in an array
     * @param arr -- an array of integers
     */
    public void print(int arr[]) {
               //For each integer
        for (int k = 0; k < arr.length; k++)
            System.out.print( arr[k] + " \t ");
               // Print it
        System.out.println();
    } // print()

    /**
     *  main() creates an instance of this (Sort) class and
     *  sets the size and visibility of its JFrame. An
     *  anonymous class is used to create an instance of the
     *  WindowListener class, which handles the window close
     *  events for the application.
     */

    public static void main(String args[]) {
        Sort sorter = new Sort();
        int intArr[] = { 21, 20, 27, 24, 19 };
        try {
            sorter.setSize(400, 300);
            sorter.setVisible(true);
            sorter.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                     // Quit the application
                    System.exit(0);
                }
            });
            sorter.print(intArr);
            sorter.insertionSort(intArr);
            sorter.print(intArr);
            // Now cause an exception
            intArr = null;
            sorter.insertionSort(intArr);
        } catch (Exception e) {
            String estr = "Null array in insertionSort()"
            JOptionPane.showMessageDialog(sorter, estr);
            System.exit(0);
        }
    } // main()
} //Sort
```

20. Design Issue: Another possible way to design the `sequentialSearch()` method (Text Figure 9.18) would be to have it throw an exception when its key

is not found in the array. Is this a good design? Explain.

Answer: No. Failing to find the key in a search is not really an exceptional condition. It is something that is expected to happen with a frequency depending on the type of search. Think of how many times your Web searches have failed. Exceptions should be reserved for truly exceptional conditions.

Chapter 11

Files and Streams: Input/Output Techniques

1. Explain the difference between each of the following pairs of terms:

   (a) `System.in` and `System.out`

   Answer: `System.in` is an input stream, which is a stream of data from an input source (usually the keyboard) into the program. `System.out` is an output stream, which is a stream of data from the program out to some destination device (usually the screen).

   (b) *File* and *directory*.

   Answer: A *file* is a sequence of bits meant to be interpreted as logically connected data. A *directory* is a collection of files.

   (c) *Buffering* and *filtering*.

   Answer: *Buffering* is the practice of moving large portions of data into and out of memory at once to avoid the inefficiency of transferring the data strictly as needed. *Filtering* is the process of performing some modification of data as it is transferred through a stream.

   (d) *Absolute* and *relative path name*.

   Answer: An *absolute* path begins from the root of the directory structure. A *relative* path begins from the current location within the directory tree.

   (e) *Input stream* and *output stream*.

   Answer: An *input* stream a stream of data from an input source (the keyboard, a file, a socket) into the program. An *output* stream is a stream of data from the program out to some destination device (the screen, a file, a socket).

   (f) *File* and *database*.

   Answer: A *file* is a collection of data. A *database* is collection of related files.

(g) *Record* and *field*.

Answer: A *field* is a chunk of data. A *record* is a collection of logically related fields.

(h) *Binary file* and *text file*.

Answer: The only difference between a binary file and a text file is interpretation. A *binary file* is meant to be interpreted as a sequence of bits, while a *text file* is meant to be interpreted as a sequence of ASCII characters.

(i) *Directory* and *database*.

Answer: The data within a collection of related files is a *database*, while a *directory* refers to a file that contains the names (not the data) of other files.

2. Fill in the blanks:

   (a) Unlike text files, binary files do not have a special _____ character. **Answer:** End of File (eof)

   (b) In Java, the `String` array parameter in the `main()` method is used of _____. **Answer:** command line arguments

   (c) _____ files are portable and platform independent. **Answer:** Java or Text

   (d) A _____ file created on one computer can't be read by another computer. **Answer:** binary

3. Arrange the following kinds of data into their correct hierarchical relationships: `bit`, `field`, `byte`, `record`, `database`, `file`, `String`, `char`.

   Answer: From highest to lowest: database, file, record, field, String, char, byte, bit

4. In what different ways can the following string of 32 bits be interpreted?

   ```
   00010101111000110100000110011110
   ```

   Answer: There are an unlimited number of ways in which a sequence of bits can be interpreted. This is why it is necessary to specify file formats!

5. When reading a binary file, why is it necessary to use an infinite loop that's exited only when an exception occurs?

   Answer: Since binary files do not contain an EOF (End of File) character, there is no way to know when the end of a binary file will be encountered. We can only react when an exception (signaling an empty stream) occurs.

6. Is it possible to have a text file with 10 characters and 0 lines? Explain.

   Answer: Yes. A file with 10 characters and 0 lines can exist if there is no newline (`\n`) character, but only a *eof* character following the 10 data characters.

7. In reading a file, why is it necessary to attempt to read from the file before entering the read loop?

   Answer: In reading a file, we must read from the file prior to entering the read loop in order to handle the case where the file is empty.

8. When designing binary I/O, why is it especially important to design the input and output routines together?

   Answer: When designing binary I/O, the interpretation of the data is the most important issue. The routines that write the data must use the same "format" at the routines that read the data. In order to ensure this symmetry, both the input and output routines should be designed together.

9. What's the difference between ASCII code and UTF code?

   Answer: The characters that can be represented in UTF code are a superset of those that can be represented in ASCII code. UTF contains additional characters which allow for internationalization. However, UTF also takes up 2 bytes per character, as opposed to 1 byte per character for ASCII representation.

10. Could the following string of bits possibly be a Java object? Explain.

    ```
    00010111000111101010101000011100100010
    11010010010101010010101001000001000000111
    ```

    Answer: Yes, the string of bits could indeed represent a Java object. Java objects can be "serialized" and written to a file, just like any other type of data. After all, anything a computer can do is dependent on some interpretation of 0's and 1's!

11. Write a method, which could be added to the `TextIO` program, that reads a text file and prints all lines containing a certain word. This should be a `void` method that takes two parameters: the name of the file and the word to search for. Lines not containing the word should not be printed.

    Answer: The `FindString` class below contains only the specified method and a `main()` method to test it.

```
/*
* File: FindString.java
* Author: Java, Java, Java
* Description: This program searches a text file for a target
* string, printing every line containing the string. The target
* and the filename are supplied as arguments to the method
* displayMatches().
*/

import java.io.*;

public class FindString{
```

```
    /**
    * displayMatches() searches line-by-line for a target string
    * in a text file. Every line containing the target is printed,
    * with its line number. The number of matching lines is also
    * reported.
    * @param target -- the String being searched for
    * @param filename -- the file being searched
    */
    public void displayMatches(String target, String filename){
        int lineCount = 0, matchCount = 0;
        try {
            BufferedReader inStream =
                new BufferedReader(new FileReader(filename));
            String line = inStream.readLine();
            while (line != null) {
                lineCount++;
                if (line.indexOf(target) >= 0) {
                //If line contains target
                    System.out.println(lineCount + ": " + line);
                    matchCount++;
                } // if
                line = inStream.readLine();
            } // while
            inStream.close();
            System.out.println("Matching Lines : " + matchCount);
        } catch (Exception e) {
             e.printStackTrace();
        } // catch
    } // displayMatches()

    /**
    * main() creates an instance of the FindString class and
    *  uses it to search a text file for a given target string.
    * @param args -- the String array is not used.
    */
    public static void main(String args[]){
        FindString app = new FindString();
        app.displayMatches("and", "testfile.txt");
    } // main()

} // FindString
```

12. Write a program that reads a text file and reports the number of characters and lines contained in the file.

    Answer: Design: The `displayStats()` method in the following `FileLineCounter` class reads one line of the file at a time. After reading each line it increments a line counter and it adds the length of the line to the character counter. It produces the following output:

```
    Lines: 49
    Chars: 1572

/* File: FileLineCounter.java
* Author: Java, Java, Java
* Description: This program counts the lines and characters
* contained in a file. The filename is supplied as an
* argument to the method displayStats().
*/

import java.io.*;

public class FileLineCounter {
     /**
     * displayStats() reads the named file and counts its lines
     *  and characters.  Results are printed to System.out
     * @param filename -- a String giving the name of the file
     */

     public void displayStats(String filename){
         int lineCount = 0;
         int charCount = 0;
         try {
             BufferedReader inStream =
                 new BufferedReader(new FileReader(filename));
             String line = inStream.readLine();
             while(line != null){
                 lineCount++;
                 charCount += line.length();
                 line = inStream.readLine();
             } // while
             inStream.close();
             System.out.println("Lines: " + lineCount);
             System.out.println("Chars: " + charCount);
         } // try
         catch (Exception e){
             System.out.println("FILE ERROR");
             e.printStackTrace();
         } // catch
     } // displayStats()

     /**
     * main() creates an instance of this class and then calls
     * the method displayStats() withargument "testfile.txt".
     * @param args -- an array of String which is not used
     */
     public static void main(String args[]){
         FileLineCounter app = new FileLineCounter();
         app.displayStats("testfile.txt");
     } // main()
```

```
} // FileLineCounter
```

13. Modify the program in the previous exercise so that it also counts the numbers of words in the file. (*Hint*: The `StringTokenizer` class might be useful for this task.)

    Answer: The solution to this exercise requires only minor additions to the `displayStats()` method. Declare an additional counter variable, `wordCount`. Then for each line, create a `StringTokenizer` instance using the line as the initial value. Then you can use the `StringTokenizer.countTokens()` method to count the words in the line. In short, make the following modifications to `displayStats()`:

```
import java.io.*;
import java.util.*;

public class FileLineCounter2 {

    public void displayStats(String filename){
        ...
        int wordCount = 0;
        try {
            ...
            while(line != null) {
                StringTokenizer st = new StringTokenizer(line);
                wordCount += st.countTokens();
                ...
            }
        ...
        System.out.println("Words: " + wordCount);
        }
        ...
    } // displayStats()
```

14. Modify the `ObjectIO` program so that it allows the user to designate a file and then input `Student` data with the help of a GUI. As the user inputs data, each record should be written to the file.

    Answer: Design: The interface should give the user the ability both to write data to the file and to read the file, if for no other reason than to verify that the program is correctly writing the data. To simplify data entry, we allow the user to input one complete student record as a delimited string – Name,Year,GPA. The program then uses a `StringTokenizer` to break the string into its individual fields.

    The interface should enable to user to append multiple records to the file. This means the file's output stream must remain open through multiple write operations. To facilitate this we want to declare the output stream as a class variable. Once the output stream is open, it should not be closed until the user elects to read the file.

The `Student` class remains unchanged from the one defined in this chapter, and is therefore not shown here.

```
/*
 * File: ObjectIO.java
 * Author: Java, Java, Java
 * Description: This class provides a graphical user interface for
 *  data entry of Student records.  The interface consists of two
 *  text fields, one for entering the name of the data file and
 *  the other for entering a student record. The student record
 *  is entered as a comma-delimited string -- Name,Year,GPA.  One
 *  control button enables the user to append one record at a time
 *  to the named datafile. The other lets the user read all the
 *  records in the data file.
 *
 * Design Issue: Since we wish to append multiple records to the
 *  data file we cannot open and close the files output stream for
 *  each record. Instead, the output stream is open, then a bunch of
 *  records are written to the file. The stream is closed only when
 *  the user chooses to read the file, after with its output stream
 *  must be reopen again before appending new records. See the
 *  actionPerformed() method for details.
 */
import javax.swing.*;         // Swing components
import java.awt.*;
import java.io.*;
import java.awt.event.*;
import java.util.*;           // String Tokenizer

public class ObjectIO extends JFrame implements ActionListener {
    private JTextArea display = new JTextArea();
    private JButton read = new JButton("Read From File"),
                    write = new JButton("Append to File");
         // A JTextField for a File name
    private JTextField fileNameField = new JTextField(10);
        // A prompt for the File name
    private JLabel prompt = new JLabel("Filename:",JLabel.RIGHT);
        // A JTextField for the Student Record data
    private JTextField studentRecord = new JTextField(24);
        // A prompt for the StudentRecord data
    private JLabel prompt2 =
        new JLabel("Student Record as: name,year,gpa",JLabel.RIGHT);
        // A panel for the
    private JPanel commands = new JPanel();
    private FileInputStream inStream;
    private FileOutputStream outStream;

    /**
     * ObjectIO() sets up the GUI using a BorderLayout, with controls
     *  at the North and a display text area in the center.
```

```
 */
public ObjectIO () {
    super("ObjectIO Demo");    // Set window title
    read.addActionListener(this);
    write.addActionListener(this);
          // Create control panel
    commands.setLayout( new GridLayout(3,2,1,1));
    commands.add(prompt);
    commands.add(fileNameField);
    commands.add(prompt2);
    commands.add(studentRecord);
    commands.add(read);
    commands.add(write);
    display.setLineWrap(true);
    this.getContentPane().setLayout(new BorderLayout () );
    this.getContentPane().add("North",commands);
    this.getContentPane().add( new JScrollPane(display));
    this.getContentPane().add("Center", display);
 } // ObjectIO

/**
 * readRecords() reads and displays Student records from
 * the named file. Note  that it opens and closes a
 * FileInputStream each time it is invoked.
 * @param fileName -- a string giving the name of the file
 */
 private void readRecords(String fileName) {
     try {      // Open a stream
         FileInputStream inStream =
             new FileInputStream(fileName);
         display.setText("Name\tYear\tGPR\n");
         try {
             while (true) { // Create an infinite loop
                   // Create a student instance
                 Student student = new Student();
                   //  and have it read an object
                 student.readFromFile(inStream);
                   //  and display it
                 display.append(student.toString()+"\n");
             } // while
         } catch (EOFException e) { //Until EOF exception
         }  // Do nothing and move on to
         inStream.close();    // Close the stream
     } catch (FileNotFoundException e) {
       display.append("IOERROR:File NOT Found:" +
                                        fileName+"\n");
     } catch (IOException e) {
       display.append("IOERROR: "+e.getMessage()+"\n");
     } catch (ClassNotFoundException e) {
       display.append("ERROR:Class NOT found" +
```

```
                                    e.getMessage()+"\n");
      }
} // readRecords()

/**
 * appendRecord() writes the current value of the
 * studentRecord textfield to the named file. Note
 * that this method does not open or close the
 * FileOutputStream, which is declared, instead,
 * as a class variable.
 * @param fileName - a string giving the name of the file
 */
private void appendRecord(String fileName) {
  try {
    StringTokenizer record =
      new StringTokenizer(studentRecord.getText(), ",;:/");

      String name = record.nextToken(); // Name
      int year =
        Integer.parseInt(record.nextToken()); // Year
      double gpr =
        Double.parseDouble(record.nextToken()); //GPA
        // Create the Student object
      Student student = new Student(name, year, gpr);
        //and write the new student in display JTextArea
      display.append("Output: "+student.toString()+"\n");
             //  and write the new student to the file
      student.writeToFile(outStream) ;
    } catch (IOException e) {
      display.append("IOERROR: "+e.getMessage()+"\n");
    }
} // appendRecord()

/**
 * actionPerformed() handles the user's button actions.
 * If  the read button was clicked, it closes the output
 * stream, if it was already open, and then invokes the
 * readRecords() method to read the named file.  If the
 * write button was clicked, it opens the output stream,
 * if necessary, and  invokes the appendRecord() method.
 * @param evt -- the ActionEvent that led to this
 *   invocation
 */
public void actionPerformed(ActionEvent evt) {
    try {     // Get the file name
    String fileName = fileNameField.getText();
    if (evt.getSource() == read) {
        if (outStream != null) {
            outStream.close();
            outStream = null;   // Reset to null
```

```
            }
            readRecords(fileName);
        }
        else if (evt.getSource() == write) {
            if (outStream == null)   // Open a stream
                outStream =
                      new FileOutputStream(fileName,true);
            appendRecord(fileName);
        }
        } catch (IOException e) {}
    } // actionPerformed()

    /**
     * main() creates and instance of this class and opens
     * its window.
     */
    public static void main(String args[]) {
        ObjectIO io = new ObjectIO();
        io.setSize( 425,200);
        io.setVisible(true);
        io.addWindowListener(new WindowAdapter() {
                // Quit the application
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()

}//ObjectIO
```

15. Write a program that will read a file of `int`s into memory, sort them in ascending order, and output the sorted data to a second file.

    Answer:Design: It is not known beforehand how many integers the input file contains. One way of discovering the number of integers is to read through the file to count them. Then create an array of the correct size and read through the file a second time and store them in the array. The sort() method of the java.util.Arrays class can be used sort the array.

```
/* File: IntFileSorter.java
 * Author: Java, Java, Java
 * Description: This program reads a file of integer
 * data into an array sorts it with Arrays.sort(),
 *  and then outputs the data to a second file.
 */

import java.io.*;
import java.util.*;

public class IntFileSorter {
```

```
    // The number of ints in "inintfile.dat"
private int numInts;
   // The array to store the integers
private int[] intArr;


/**
 * writeInts() writes the data to the named file
 * @param filename - name of the destination file
 */
private void writeInts(String fileName){
    try {
        FileOutputStream fos =
            new FileOutputStream(fileName);
        DataOutputStream outStream =
            new DataOutputStream (fos);
        for(int j=0; j < numInts; j++) {
            // Output to the file
            outStream.writeInt(intArr[j]);
        } // for
    } // try
    catch (IOException e){
        System.out.println("IOERROR!");
        e.printStackTrace();
    } // catch
} // writeInts()

/**
* countInts() reads the ints from the named file
* and counts them storing the number of ints in
* the instance variable numInts.
* @param filename - string giving the name of a file
*/
private void  countInts(String fileName){
    try{
        numInts = 0;
        FileInputStream fis =
            new FileInputStream(fileName);
        DataInputStream inStream =
            new DataInputStream(fis);
        try{
            while(true) {
                int n = inStream.readInt();
                // int was read so count it
                numInts++;
            } // while
        } // try
        catch (EOFException e) {}
        finally{
            inStream.close();
```

```
        } // finally
    } // try
    catch (IOException e){
        System.out.println("IOERROR!");
        e.printStackTrace();
    } // catch
} // countInts()

/**
* readInts() reads the ints from the named file
* and stores them in the array intArr. It is
* assumed that numInts stores the number of ints
* that are in the file.  The ints are printed to
* System.out as they are read.
* @param filename -- a string giving the name
*    of a file
*/
private void  readInts(String fileName){
    try{   // Assign memory
        intArr = new int[numInts];
        FileInputStream fis =
            new FileInputStream(fileName);
        DataInputStream inStream =
            new DataInputStream(fis);
        try{
            int index = 0;
            while(true) {
                intArr[index] = inStream.readInt();
                // if read is successful, write
                // the int read to standard out
                System.out.print( intArr[index] + " ");
                index++;  // and increase index
            } // while
        } // try
        catch (EOFException e) {}
        finally{
            inStream.close();
            Arrays.sort(intArr); // Sort the array
        } // finally
    } // try
    catch (IOException e){
        System.out.println("IOERROR!");
        e.printStackTrace();
    } // catch
} // readInts()

/** NOT PART OF THE SOLUTION - USED TO CREATE
*    the file inintfile.dat of integers to be sorted.
* createIntFile() stores 100 unsorted ints in the
* named file.
```

```
     * @param filename - string giving the name of a file
     */

     private void createIntFile(String fileName){
         try {
             FileOutputStream fos =
                 new FileOutputStream(fileName);
             DataOutputStream outStream =
                 new DataOutputStream (fos);
             for(int j=0; j < 100; j++) {
                 int n = 371*j%503;
                 // Output to standard out
                 System.out.print(n + " ");
                 outStream.writeInt(n);
             } // for
         } // try
         catch (IOException e){
             System.out.println("IOERROR!");
             e.printStackTrace();
         } // catch
     } // createIntFile()

     /**
     * main() creates an instance of this class and uses it
     *  to read, sort and write a file of integer data.
     * @param -- args - a String array which is not used
     */
     public static void main(String args[]){
         IntFileSorter app = new IntFileSorter();
         // app.createIntFile("inintfile.dat");
         app.countInts("inintfile.dat");
         System.out.println("\nreading from input file");
         app.readInts("inintfile.dat");
         app.writeInts("outintfile.dat");
         System.out.println("\n reading from output file");
         app.readInts("outintfile.dat");
     } // main()

} // IntFileSorter
```

16. Write a program that will read two files of `int`s, which are already sorted into ascending order, and merge their data. For example, if one file contains 1, 3, 5, 7, 9, and the other contains 2, 4, 6, 8, 10, then the merged file should contain 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

    Answer: Design: One solution to the problem is to create a method that reads an integer from each file. Then create a loop which writes the smaller element to the output file and reads a new integer from the file that integer came from until one of the files reaches an end of file. Finally, the remaining integers in one of the files need to be read and written to the output file.

```
/* File: IntFileMergeSorter.java
 * Author: Java, Java, Java
 * Description:  This program contains a static method
 * that merges two files of sorted integer data, creating
 * a thirdfile whose data is also sorted into increasing
 * order.
 *
 */

import java.io.*;
import java.util.*;

public class IntFileMerger {

    /**
     * merge() merges the increasing integer values in inFile1
     * and inFile2 as they are read and writes them to outFile.
     * @param inFile1 - name for a file of increasing integers
     * @param inFile2 - name for a file of increasing integers
     * @param outFile - a string giving the name of a file
     */
    public static void merge(String inFile1, String inFile2,
                                              String outFile){
      try{
            // Open the 3 files and create 3 streams
        FileInputStream fis1 = new FileInputStream(inFile1);
        DataInputStream inStream1 = new DataInputStream(fis1);
        FileInputStream fis2 = new FileInputStream(inFile2);
        DataInputStream inStream2 = new DataInputStream(fis2);
        FileOutputStream fos = new FileOutputStream(outFile);
        DataOutputStream outStream = new DataOutputStream(fos);

              // declare int and boolean variables for merging
        int num1 = 0;  // Use for an int from inStream1
        int num2 = 0;  // Use for an int from inStream2
        boolean moreValues1 = false; //num1 has no value to write
        boolean moreValues2 = false; //num2 has no value to write

                // Read first value from inStream1
        try{
          num1 = inStream1.readInt();
          moreValues1 = true;
        } catch (EOFException e) { //inStream1 has no int values
        } // catch()

                // Read first value from inStream2
        try{
          num2 = inStream2.readInt();
          moreValues2 = true;
        } catch (EOFException e) { //inStream2 has no int values
```

```
} // catch()

        // Write smallest value and read another until
        // all values from one file have been written
while (moreValues1 && moreValues2) {
  if (num1 <= num2) {  // then num1 is smaller or equal
    outStream.writeInt(num1);  // Write num1 to file
    System.out.print(num1 + " ");  //Report to System.out
    try{  // Try to read another value from inStream1
      num1 = inStream1.readInt();
    } catch (EOFException e) {
        //EOF means no more values in file
      moreValues1 = false;
    } // catch()

  } else { // num2 is smaller
    outStream.writeInt(num2);  // Write num2 to file
    System.out.print(num2 + " "); //Report to System.out
    try{  // Try to read another value from inStream2
      num2 = inStream2.readInt();
    } catch (EOFException e) {
        //EOF means no more values in file
      moreValues2 = false;
    } // catch()
  } // else
} //while both files have values

  //If only inStream1 has more values write them to file
while(moreValues1){
  outStream.writeInt(num1);  // Write num1 to file
  System.out.print(num1 + " ");  //Report to System.out
  try{  // Try to read another value from inStream1
    num1 = inStream1.readInt();
  } catch (EOFException e) {
    moreValues1 = false; //EOF means no more values
  } // catch()
} // while only inStream1 still has values

 //If only inStream2 has more values write them to file
while(moreValues2){
  outStream.writeInt(num2);  // Write num1 to file
  System.out.print(num2 + " "); //Report to System.out
  try{  // Try to read another value from inStream1
    num2 = inStream2.readInt();
  } catch (EOFException e) {
    moreValues2 = false; //EOF means no more values
  } // catch()
} // while only inStream2 still has values

    // Close all three streams
```

```
    inStream1.close();
    inStream2.close();
    outStream.close();
  } catch(IOException e){
    System.out.println("IOERROR!");
    e.printStackTrace();
  } // catch()
} // merge()

 /** NOT PART OF SOLUTION ****
 * writeInts() writes the data from an array of integers
 * to the named file. This method can be used to create
 * files of sorted integers to test the merge() method.
 * @param arr -- an array of (increasing) int values
 * @param filename -- the name of the destination file
 */
public static void writeInts(int[] arr, String fileName){
  try{
    FileOutputStream fos = new FileOutputStream(fileName);
    DataOutputStream outStream = new DataOutputStream(fos);
    for(int j=0; j < arr.length; j++) {
      outStream.writeInt(arr[j]);  // Write to file
        //report to System.out
      System.out.print(arr[j] + " ");
    } // for
    outStream.close();
  }  catch (IOException e){
    System.out.println("IOERROR!");
    e.printStackTrace();
  } // catch()
} // writeInts()

 /** NOT PART OF SOLUTION ****
 * readInts() reads and displays a sequence of integers
 * from the named file. This method can be used to verify
 *  that the merge() method has written the correct file.
 * @param filename -- the name of the destination file
 */
public static void readInts(String fileName){
  int num = 0; // For holding integers read
  try{
    FileInputStream fis = new FileInputStream(fileName);
    DataInputStream inStream = new DataInputStream(fis);
    try{ // read from file until EOF reached
      while(true) {
        num = inStream.readInt();  // Read from file
          //report to System.out
        System.out.print(num + " ");
      } // while
    } catch (EOFException e) {
```

```
          inStream.close();
        } //catch() EOF
      }  catch (IOException e){
        System.out.println("IOERROR!");
        e.printStackTrace();
      } // catch()
    } // readInts()

    /**
     * main() creates two files of sorted integers and merges
     * the integers in the two files and writes them to a
     * third file of sorted integers. The integers are read
     * from the merged file to verify that they have been
     * written correctly.
     * @param -- args, a String array not used in this
     *    implementation.
     */
    public static void main(String args[]){
      // Create first file of increasing sequence of integers
        int[] arr1 = {3,11,43,50,55,67,71};
        IntFileMerger.writeInts(arr1, "inTest1.dat");
        System.out.println(" inTest1.dat written");

        int[] arr2 = {5,9,53,54,55,66,77,78,79};
        IntFileMerger.writeInts(arr2, "inTest2.dat");
        System.out.println(" inTest2.dat written");

        System.out.println("Merging Files");
        IntFileMerger.merge("inTest1.dat", "inTest2.dat",
                                              "outTest.dat");
        System.out.println("\noutTest.dat written");
        IntFileMerger.readInts("outTest.dat");
        System.out.println("\nDone reading data from outTest.dat");
    } // main()
} // IntFileMerger
```

17. Suppose you have file of data for a geological survey, such that each record consists of a longitude, a latitude, and an amount of rainfall, all represented by `doubles`. Write a method to read this file's data and print them on the screen, one record per line. The method should be `void` and it should take the name of the file as its only parameter.

    Answer: The `GeoData` class below implements the method specified in this exercise and also the method specified in the next exercise.

```
/*
 * File: GeoData.java
 * Author: Java, Java, Java
 * Description: This program writes and reads a data file
 *  containing rainfall records at various geographical
```

```
 * locations. Each record consists of a latitude, longitude
 * and rainfall amount.
 */

import java.io.*;

public class GeoData {

    /**
     * getGeoData() reads data from the named file
     * @param fileName -- the name of the data file
     */
    public static void getGeoData(String fileName){
        try{
            DataInputStream inStream =
                new DataInputStream(new FileInputStream(fileName));
            System.out.println("Reading data from: " + fileName);
            System.out.println("Long.    Lat.    Rain    ");
            try{
                while(true){
                    double longitude = inStream.readDouble();
                    double latitude = inStream.readDouble();
                    double rainfall = inStream.readDouble();
                    String record = new String("");
                    record += "Longitude: " + longitude + "\t";
                    record += "Latitude: " + latitude + "\t";
                    record += "Rainfall: " + rainfall + "\t";
                    System.out.println(record);
                } // while
            } // try
            catch (EOFException e) {}
            finally {
                inStream.close();
            } // finally
        }
        catch (IOException e){
    System.out.println("IOERROR!");
    e.printStackTrace();
        }
    } // getGeoData()

    /**
     * writeGeoData() writes data to the named file
     * and reports it to System.out as it writes.
     * @param fileName -- the name of the output file
     */
    public static void writeGeoData(String fileName) {
        java.util.Random gen = new java.util.Random();
        try{
          DataOutputStream outStream =
```

```
            new DataOutputStream (new FileOutputStream(fileName));
          System.out.println("Writing data to: " + fileName);
          System.out.println("Long.    Lat.    Rain    ");
          for(int j=0; j < 1000; j++){
            double longitude = (gen.nextDouble() * 360) - 180;
            double latitude = (gen.nextDouble() * 360) - 180;
            double rainfall = gen.nextDouble() * 20;
            outStream.writeDouble(longitude);
            outStream.writeDouble(latitude);
            outStream.writeDouble(rainfall);
            System.out.print(longitude + ' ');
            System.out.print(latitude + ' ');
            System.out.println(rainfall + ' ');
          } // for
        } // try
        catch (IOException e){
          System.out.println("IOERROR!");
          e.printStackTrace();
        } // catch()
    } // writeGeoData()

    /**
     * main() creates an instance of this class and writes data
     * to a file and then reads the data from that file.
     * @param -- args, a String array not used in this
     *   implementation.
     */
    public static void main(String args[]) {
        GeoData.writeGeoData("geo.dat");
        GeoData.getGeoData("geo.dat");
    } // main()
} // GeoData
```

18. Suppose you have the same data as in the previous exercise. Write a method that will generate 1000 records of random data and write them to a file. The method should be void and should take the file's name as its parameter. Assume that longitudes have values in the range $+/-$ 0 to 180 degrees, latitudes have values in the range $+/-$ 0 to 90 degrees, and rainfalls have values in the range 0 to 20 inches.

    Answer: See the solution to the previous exercise.

19. Design and write a file copy program that will work for either text files or binary files. The program should prompt the user for the names of each file and copy the data from the source file into the destination file. It should not overwrite an existing file, however. (*Hint*: read and write the file as a file of `byte`.)

    Answer: Design: Since bytes are more general than characters, a program that reads bytes can be used to copy either type of file.

    ```
    /*
    ```

```
 * File: FileCopier.java
 * Author: Java, Java, Java
 * Description: This program has a static method that
 * copies a data file, byte by byte. The destination file
 * must be a new file or the copying is not done.
 */

import java.io.*;

public class FileCopier {
    /**
     * If a file named dest does not exist then
     * copyFile() creates the input and output streams
     * from the given names and reads bytes from
     * the input and writes them to the output.
     * @param src -- a String, a name of the input file
     * @param dest -- a String, a name of the output file
     */
    public static void copyFile(String src, String dest){
        //
        if ((new File(dest)).exists()){
          System.out.println("Destination file exists. Aborting.");
          return;
        } // if

       // destination file does not exist so copy to it.
        try{
          DataInputStream inStream =
            new DataInputStream(new FileInputStream(src));
          DataOutputStream outStream =
            new DataOutputStream (new FileOutputStream(dest));
          try{
            while(true){
               byte currentByte = inStream.readByte();
               outStream.writeByte(currentByte);
            } // while
          } // try
          catch (EOFException e) {}
          finally{
            inStream.close();
            outStream.close();
          } // catch()
        } // try
        catch(IOException e){
          System.out.println("IOERROR!");
          e.printStackTrace();
        } // catch()
    } // copyFile()

    /**
```

```
     * main() -- Uses the copyFile() method to try to copy a
     * text file, then to try to copy an integer file, and
     * finally to copy the text file to an output file with
     * the same name as in the first case.
     * @param args -- a String array not used here.
     */
    public static void main (String args[]){
           // Remove the output files if they exist
        new File("outText.txt").delete();
        new File("outInt.dat").delete();
        System.out.println("Try copying textFile.txt " +
                                          "to outText.txt");
        FileCopier.copyFile("textFile.txt", "outText.txt");
        System.out.println("Try copying intFile.dat " +
                                           "to outInt.dat");
        FileCopier.copyFile("intFile.dat", "outInt.dat");
        System.out.println("Try copying textFile.txt " +
                                       "to outText.txt again");
        FileCopier.copyFile("textFile.txt", "outText.txt");
     } // main()
} // FileCopier
```

20. Design a class, similar to `Student` to represent an `Address`, consisting of street, city, state and zip code. This class should contain its own `readFromFile()` and `writeToFile()` methods.

    Answer:

```
/* File: Address.java
 * Author: Java, Java, Java
 * Description: This class defines a address record,
 * consisting of street, city, stat and zip code.
 * It contains methods to read and write files containing
 * such objects. It implements the Serializable interface,
 * which enables it to read and write objects from and
 * into sequential data files.
 */

import java.io.*;

public class Address implements Serializable {
    String Street, City, State, Zip;

    /**
     * Address() is the default constructor
     */
    public Address() {} // Address

    /**
     * Address() constructs an object from several
```

```
 * string parameters, each representing a field
 * of an address record
 */
public Address(String street, String city, String state,
                        String zip) {
    Street = street;
    City = city;
    State = state;
    Zip = zip;
} // Address()

/**
 * writeToFile() outputs this object to the named stream
 * @param outStream -- a reference to an output stream
 */
public void writeToFile(FileOutputStream outStream)
                         throws IOException {
    ObjectOutputStream ooStream =
        new ObjectOutputStream(outStream);
    ooStream.writeObject(this);
    ooStream.flush();
} // writeToFile()

/**
 * readFromFile() reads an object from inStream
 * @param inStream -- reference to an input stream
 */
public void readFromFile(FileInputStream inStream)
        throws IOException, ClassNotFoundException {
    ObjectInputStream oiStream =
        new ObjectInputStream(inStream);
    Address a = (Address) oiStream.readObject();
    this.Street = a.Street;
    this.City = a.City;
    this.State = a.State;
    this.Zip = a.Zip;
} // readFromFile()

/**
 * toString() displays this object as an address record
 * @return
 */
public String toString(){
    return(Street + "\n" + City + ", " + State +
                               "\t" + Zip + "\n");
} // toString()

/**
 * main() tests this class's read and write routines.
 * It first writes several address records to a file,
```

```
     * and then reads them back in. It prints both what it
     * reads and what it writes.
     */
    public static void main(String args[]) throws IOException,
            FileNotFoundException, ClassNotFoundException {
        FileOutputStream out =
            new FileOutputStream("address.dat");
        System.out.println("Writing addresses to address.dat");
        for (int k = 0; k < 5; k++) {
            Address a = new Address(k + " M Street","New York",
                                    "NY", "10001");
            a.writeToFile(out);
            System.out.println(a.toString());
        }
        FileInputStream in = new FileInputStream("address.dat");
        System.out.println("Reading addresses from address.dat");
        for (int k = 0; k < 5; k++) {
            Address a = new Address();
            a.readFromFile(in);
            System.out.println(a.toString());
        }
    } // main()

} // Address
```

21. Using the class designed in the previous exercise, modify the `Student` class so that it contains an `Address` field. Modify the `ObjectIO` program to accommodate this new definition of `Student` and test your program.

    Answer: Very few modifications need to be made. Inside the class definition of Student we must add:

    ```
    private Address address;
    ```

    inside the readFromFile method we must add:

    ```
    this.address = (Address) s.address;
    ```

    and the body of the toString() method becomes:

    ```
    return name + "\t" + year + "\t" + gpa + "\n" + address;
    ```

    The constructor method must also be changed to accommodate the addition of the address field. Otherwise, since `writeObject()` and `readObject()` will output or input an entire object, no changes need be made to the `ObjectIO` class. The `Serializable` designation will take care of the details.

22. Write a program called `Directory`, which provides a listing of any directory contained in the present directory. This program should prompt the user for the

name of a directory. It should then print a listing of the directory. The listing should contain the following information: the full path name of the directory, and the file name, length, and last modified date, and a read/write code for each file. The read/write code should be an 'r' if the file is readable and a 'w' if the file is writeable, in that order. Use a '-' to indicate not readable or not writeable. For example, a file that is readable but not writable will have the code "r-". Here's an example listing:

```
Listing for directory: myfiles
  name          length modified   code
  index.html    548    129098     rw
  index.gif     78     129190     rw
  me.html       682    128001     r-
  private.txt   1001   129000     --
```

Note that the `File.lastModified()` returns a `long`, which gives the modification time of the file. This number can't easily be converted into a date, so just report its value.

Answer: The program shown here produced the following directory listing when run in the directory containing the program itself:

```
NAME                    LENGTH    MODIFIED      CODE
Directory.java          2172      937973197000  rw
Directory.java~         1366      937972501000  rw
Directory.class         1563      937972831000  rw
```

```
/*
 * File: Directory.java
 * Author: Java, Java, Java
 * Description: This program prints a directory listing
 * given the name of a directory on the system.
 */

import java.io.*;

public class Directory  {
    /**
     * getCode() constructs a rw code for the given file, showing
     * whether the file is readable and/or writeable
     * @param f -- the File that is being characterized
     * @return a String of the form rw or r- or -w or --
     */
    public static String getCode(File f){
        String code = "";
        if(f.canRead())
            code += "r";
        else
```

```
            code += "-";
        if(f.canWrite())
            code += "w";
        else
            code += "-";
        return code;
    } // getCode()

    /**
     * listDirectory() gets the names of all the files in the
     * named directory and prints their names, lengths, last
     * modified dates,  and read/write characterization.
     * @param dirname -- a String giving the name of the directory
     */
    public static void listDirectory(String dirname){
        File dir = new File(dirname);
        System.out.println("Listing for the directory: " + dirname);
        String banner = "NAME\t\t\tLENGTH\t\t\tMODIFIED\t\t\tCODE";
        System.out.println(banner);
        String[] fileList = dir.list();
        for(int j = 0; j < fileList.length; j++){
            File f = new File(dir, fileList[j]);
            System.out.print(f.getName() + "\t\t\t");
            System.out.print(f.length() + "\t\t\t");
            System.out.print(f.lastModified() + "\t\t\t");
            System.out.print(getCode(f) + "\n");
        }
    } // listDirectory()

    /**
     * main() prompts the user to input the name of a directory and
     *  then calls the listDirectory() method.
     * @param args -- a String array not used here.
     */
    public static void main(String args[]){
        try {
            BufferedReader br =
                new BufferedReader( new InputStreamReader(System.in));
            System.out.print("Input the name of a directory:");
            String dName = br.readLine();
            Directory.listDirectory(dName);
        } catch(IOException e){
            System.out.println("IOERROR!");
            e.printStackTrace();
        } // catch()
    } // main()

} // Directory
```

23. **Challenge:** Modify the `OneRowNimGUI` class listed in Figure 4.25 so that the

user can save the position of the game to a file or open the position from a file. Add two new `JButtons` to the GUI interface. Use the `Student` serialization example as a model for your input and output streams.

Answer: Changes are needed in both `OneRowNim` and `OneRowNimGUI`. The `Student` class can be used as a model of how to make `OneRowNim` Serializable with new methods `writeToFile()` and `readFromFile()`. The methods for writing and reading files containing `Student` data in the `ObjectIO` class can be used as a model for methods that are called in `OneRowNimGUI` class when the new buttons are clicked to save or open a `OneRowNim` game position. The dialog methods in `JFileChooser` are used to choose the file names to write to or read from.

```
/*
 * File: OneRowNim.java
 * Author: Java Java Java
 * Description: This version of OneRowNim includes several constructor
 * methods and several instance methods and it uses an if-else structure
 * in its takeSticks() method.
 * NOTE MODIFICATIONS FOR FILE IO: The class is declared as implementing
 * Serializable. import java.io.*; has been added at the start of this file.
 * New methods writeToFile() and readFromFile() have been implemented.
 */
import java.io.*;

public class OneRowNim implements Serializable
{   private int nSticks = 7;
    private int player = 1;

    public OneRowNim()
    {
    } // OneRowNim() constructor

    public OneRowNim(int sticks)
    {   nSticks = sticks;
    }  // OneRowNim() constructor2

    public OneRowNim(int sticks, int starter)
    {   nSticks = sticks;
        player = starter;
    }  // OneRowNim() constructor3

    public boolean takeSticks(int num)
    {   if (num < 1) return false;       // Error
        else if ( num > 3) return false; // Error
        else              // this is a valid move
        {   nSticks = nSticks - num;
            player = 3 - player;
            return true;
        } // else
```

```
} // takeSticks()

public int getSticks()
{   return nSticks;
} // getSticks()

public int getPlayer()
{   return player;
} // getPlayer()

public boolean gameOver()
{   return (nSticks <= 0);
} // gameOver()

public int getWinner()
{   if (nSticks < 1) return getPlayer();
    else return 0;  //game is not over
} // getWinner()

public void report()
{   System.out.println("Number of sticks left: " +
                                        getSticks());
    System.out.println("Next turn by player " +
                                        getPlayer());
}   // report()

/**
 * writeToFile() writes this object's data to a
 * designated file. Note the use of an  ObjectOutputStream,
 * which enables us to write Serializable  objects to the
 * stream using writeObject.
 * @param outStream -- the OutputStream associated with
 *   the file
 */
public void writeToFile(FileOutputStream outStream)
                                        throws IOException{
    ObjectOutputStream ooStream =
                        new ObjectOutputStream(outStream);
    ooStream.writeObject(this);
    ooStream.flush();
} // writeToFile()

/**
 * readFromFile() reads this object's data from a
 * designated file. Note the use of the ObjectInputStream,
 * which enables us to read a Serializable object from
 * the stream.
 * @param inStream -- the InputStream associated with
 *   the file
 */
```

```
    public void readFromFile(FileInputStream inStream) throws
            IOException, ClassNotFoundException {
        ObjectInputStream oiStream =
                              new ObjectInputStream(inStream);
        OneRowNim g = (OneRowNim)oiStream.readObject();
        this.nSticks = g.nSticks;
        this.player = g.player;
    } // readFromFile()

    /**
     * toString() overrides the Object.toString() method
     */
    public String toString() {
      String str ="Number of sticks left: "+getSticks()+"\n";
      str = str+"Next turn by player "+getPlayer()+"\n";
      return str;
    } // toString

} // OneRowNim class

/*
 * File: OneRowNimGUI.java
 * Author: Java Java Java
 * Description: This application plays One Row Nim via a GUI
 * interface.
 * NOTE MODIFICATIONS FOR FILE IO:Two new JButtons as instance
 * variables are added and they are initialized in buildGUI().
 * import java.io.*; has been added at the start of this file.
 * New saveNimToFile() and readNimFromFile() methods are
 * implemented and are called in the actionPerformed()
 * when the new buttons are clicked.
 */
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class OneRowNimGUI extends JFrame
                     implements ActionListener
{   private JTextArea display;
    private JTextField inField;
    private JButton goButton;
    private JButton saveButton;
    private JButton readButton;
    private OneRowNim game;

    public OneRowNimGUI(String title)
    {   game = new OneRowNim(21);
        buildGUI();
        setTitle(title);
```

```
    pack();
    setVisible(true);
} // OneRowNimGUI()

private void buildGUI()
{   Container contentPane = getContentPane();
    contentPane.setLayout(new BorderLayout());
    display = new JTextArea(20,30);
    display.setText("Let's play Take Away. There are " +
                    game.getSticks() + " sticks.\n" +
                   "Pick up 1,2, or 3 at a time.\n" +
                                      "You go first.\n");
    inField = new JTextField(10);
    goButton = new JButton("Take Sticks");
    goButton.addActionListener(this);
    saveButton = new JButton("Save to File");
    saveButton.addActionListener(this);
    readButton = new JButton("Read from File");
    readButton.addActionListener(this);
    JPanel inputPanel = new JPanel();
    inputPanel.add(new
              JLabel("How many sticks do you take:"));
    inputPanel.add(inField);
    inputPanel.add(goButton);
    inputPanel.add(saveButton);
    inputPanel.add(readButton);
    contentPane.add("Center", display);
    contentPane.add("South", inputPanel);
} // buildGUI

private void userMove()
{   int userTakes = Integer.parseInt(inField.getText());
    if (game.takeSticks(userTakes))
      display.append("You take " + userTakes + ".\n");
    else
      display.append("You can't take " + userTakes +
                                       ". Try again\n");
} // userMove()

private void computerMove()
{   if (game.gameOver()) return;
    if (game.getPlayer() == 2)
    {   game.takeSticks(1); // Temporary strategy
        display.append("I take one stick. ");
    } // if
} // computerMove()

private void endGame()
{      // Disable button and textfield
    goButton.setEnabled(false);
```

```
    inField.setEnabled(false);
    if (game.getWinner() == 1)
      display.append("Game over. You win. Nice game.\n");
    else
      display.append("Game over. I win. Nice game.\n");
} // endGame()

private void saveNimToFile() {
    JFileChooser chooser = new JFileChooser();
    int result = chooser.showSaveDialog(this);
    if (result == JFileChooser.APPROVE_OPTION){
        File  theFile = chooser.getSelectedFile();
        String fileName = theFile.getName();
        try {  // Open a stream
           FileOutputStream outStream =
             new FileOutputStream( fileName );
           // tell game to write it's data
           game.writeToFile(outStream) ;
           outStream.close();
           display.append("Game written to " +
                                      fileName + "\n");
        } catch (IOException e) {
           display.append("IOERROR: " + e.getMessage()
                                             + "\n");
        } // catch()
    } //if
} // saveNimToFile()

private void readNimFromFile() {
    JFileChooser chooser = new JFileChooser();
    int result = chooser.showOpenDialog(this);
    if (result == JFileChooser.APPROVE_OPTION){
      File  theFile = chooser.getSelectedFile();
      String fileName = theFile.getName();
      try {
        // Open a stream
        FileInputStream inStream =
            new FileInputStream(fileName);
        try {
          // read an object
          game.readFromFile(inStream);
          //  and display it
          display.setText("Game read from file " +
                                 fileName + "\n");
          display.append(game.toString());
            // Enable button for moves
          goButton.setEnabled(true);
            // Enable textfield
          inField.setEnabled(true);
        } catch (IOException e) {
```

```
            }
            inStream.close();  // Close the stream
         } catch (FileNotFoundException e) {
             display.append("IOERROR:File NOT Found:" +
                                    fileName + "\n");
         } catch (IOException e) {
             display.append("IOERROR:" + e.getMessage()
                                              + "\n");
         } catch (ClassNotFoundException e) {
             display.append("ERROR:Class NOT found" +
                                 e.getMessage() + "\n");
         } // catch()

        } //if
    } // readNimFromFile()

    public void actionPerformed(ActionEvent e)
    {   if (e.getSource() == goButton) {
            userMove();
            computerMove();
            int sticksLeft = game.getSticks();
            display.append("There are " + sticksLeft +
                                      " sticks left.\n");
            if (game.gameOver()) endGame();
        } else  if (e.getSource() == saveButton) {
            saveNimToFile();
            display.append("There are " + game.getSticks()
                                    + " sticks left.\n");
        } else  if (e.getSource() == readButton) {
            readNimFromFile();
            display.append("It is your turn to play.\n");
        } // if
    } // actionPerformed()

    public static void main(String args[])
    {
        new OneRowNimGUI("One Row Nim");
    }
} // OneRowNimGUI
```

24. **Challenge:** In Unix systems there's a program named `grep` that can be used to list the lines in a text file that contain a certain string. Write a Java version of this program that prompts the user for the name of the file and the string to search for.

    Answer: Design: The program must read each line of the file. It can then use the `String.indexOf()` method to determine if the search string occurs in the file. Here's the output you should receive if you run this program on the `Grep.java` source file searching for the string "class".

```
12: public class Grep {
Matching Lines : 1
```

A `Buffered Reader` will be used to read user input. Obviously, a `KeyboardReader` could have been used.

```
/*
 * File: Grep.java
 * Author: Java, Java, Java
 * Description: This program searches a text file for a
 * target string, printing every line containing the
 * string. The target and the filename are read from the
 * command line as user input.
 */

import java.io.*;

public class Grep {
    /**
     * displayMatches() searches line-by-line for a target
     * string in a text file. Every line containing the
     * target is printed, with its line number.  The
     *  number of matching lines is also reported.
     * @param target -- the String being searched for
     * @param filename -- the file being searched
     */
    public static void displayMatches(String target,
                                      String filename){
        int lineCount = 0, matchCount = 0;
        try {
            BufferedReader inStream =
                new BufferedReader(new FileReader(filename));
            String line = inStream.readLine();
            while (line != null) {
                lineCount++;
                    // If line contains target
                if (line.indexOf(target) >= 0) {
                    System.out.println(lineCount + ": " + line);
                    matchCount++;
                } // if
                line = inStream.readLine();
            } // while
            inStream.close();
            System.out.println("Matching Lines : " + matchCount);
        } catch (Exception e) {
            e.printStackTrace();
        } // catch()
    } // displayMatches()

    /**
```

```
     * main() prompts the user for a file name and a string to
     * be used as a target for a search in the specified file.
     * a text file for a given target string. Both the target
     * and filenameare supplied as command line arguments.
     * @param args -- a String array unused in this method
     */
    public static void main(String args[]){
        try {
          BufferedReader br =
            new BufferedReader( new InputStreamReader(System.in));
          System.out.print("Input the name of a text file:");
          String fName = br.readLine();
          System.out.print("Input a string to search for:");
          String str = br.readLine();
          Grep.displayMatches(str, fName);
        } catch(IOException e){
          System.out.println("IOERROR!");
          e.printStackTrace();
        } // catch()
    } // main()

} // Grep
```

25. **Challenge:** Write a program in Java named `Copy` to copy one file into another. The program will prompt the user for two file names, `filename1` and `filename2`. Both `filename1` and `filename2` must exist or the program should throw a `FileNotFoundException`. Although `filename1` must be the name of a file (not a directory), `filename2` may be either a file or a directory. If `filename2` is a file, then the program should copy `filename1` to `filename2`. If `filename2` is a directory, then the program should simply copy `filename1` into `filename2`. That is, it should create a new file with the name `filename1` inside the `filename2` directory, copy the old file to the new file, and then delete the old file.

    Answer: Design: The `validateAndCopy()` method should take the names of the source and destination files as its parameters. It can create `File` instances using these names. Then it can use some of the `File` methods to test for existence, file type and so on. If a test fails, it should throw an exception. If both file names are valid, it should proceed to copy the files.

```
/*
 * File: Copy.java
 * Author: Java, Java, Java
 * Description: This program copies a source file
 * to a destination file, where the user is prompted
 * for names of the files as input from the keyboard.
 *
 *  The program observes the following constraints:
 *    (1) The sourcefile must exist or a
 *       FileNotFoundException is thrown.
```

```
 *    (2) The sourcefile must be the name of a file,
 *        not a directory. The destination file may
 *        be either a directory or a file.
 *    (3) If the destination is a directory, the
 *        sourcefile should be copied into it.
 *    (4) If the destination file is not a directory,
 *        it must not already exist.
 */

import java.io.*;

public class Copy {

    /**
     * copyFile() reads bytes from the file src and
     *  writes them to the file dest.
     * @param src -- reference to the source file
     * @param dest -- reference to the destination file
     */
    public static void copyFile(File src, File dest){
        try{

            DataInputStream inStream =
              new DataInputStream(new FileInputStream(src));
            DataOutputStream outStream =
              new DataOutputStream (new FileOutputStream(dest));
            try {
                while(true) {
                    byte currentByte = inStream.readByte();
                    outStream.writeByte(currentByte);
                } // while
            } // try
            catch (EOFException e) {}
            finally {
                inStream.close();
                outStream.close();
            } // finally
        } catch (IOException e) {
        } // catch()

    } // copyFile()

    /**
     * validateAndCopy() validates the supplied file names.
     * If no exception is thrown, it performs the desired
     * copy operation.
     * @param source -- a String giving the name of the source
     * file, which should be the name of an existing file.
     * @param dest -- a String giving the name of the
     * destination file, which should be either an existing
```

```
         * directoryor the name of a nonexistent file
         */
        public static void validateAndCopy(String source, String dest) {
            try {
                File file1 = new File(source);
                File file2 = new File(dest);
                if (!(file1.exists())) {
                    String str1 = file1.getName() + " Not Found";
                    throw new FileNotFoundException(str1);
                } // if file1 does not exist
                if (!(file1.isFile())) {
                    String str2 = file1.getName()+" must be a file!";
                    throw new IOException(str2);
                } // if file1 is not a file
                if (file2.exists() && !file2.isDirectory()) {
                    String str3 = file2.getName()+" must be a directory";
                    throw new IOException(str3);
                } // if file2 exists and is not a directory
                if (!file2.exists())
                    copyFile(file1, file2);
                else
                    copyFile(file1, new File(file2, source));
            } catch (IOException ex) {
                System.out.println(ex.getMessage());
                ex.printStackTrace();
            } // catch()
        } // validateAndCopy()

        /**
         * main() Copies source file to the destination file.
         * The user is prompted for the file names and calls the
         * the static method validateAndCopy().
         * @param args -- a String array not used here.
         */
        public static void main (String args[]) {
          try {
            BufferedReader br =
              new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Input the name of a source file:");
            String inFile = br.readLine();
            System.out.print("Input name for destination file or directory:");
            String outFile = br.readLine();
            Copy.validateAndCopy(inFile, outFile);
          } catch(IOException e){
            System.out.println("IOERROR!");
            e.printStackTrace();
          } // catch()
        } // main()

    } // Copy
```

Chapter 12

Recursive Problem Solving

1. Explain the difference between the following pairs of terms:

   (a) *Iteration* and *recursion*.

   Answer: *Iteration* employs loop structures, such as for, while and do-while structures, to perform a repetitive algorithm. *Recursion* employs a recursive method to perform repetition, where a recursive method is one that calls itself.

   (b) *Recursive method* and *recursive definition*.

   Answer: A *recursive method* is a method that calls or invokes itself, usually with a different argument, as a means of performing some repetitive algorithm. A *recursive definition* is a definition of a concept which involves an integer value, say N, where the definition of the concept for larger values of N is defined in terms of the the concept with smaller values of N. The concept for the smallest values of N have an explicit definition.

   (c) *Base case* and *recursive case*.

   Answer: A *base case* is that part of a recursive definition or method that does not contain a recursive call. It serves to limit or bound the process of repetition. A *recursive case* is that part of a recursive definition or method that does involve a recursive call.

   (d) *Head* and *tail*. **Answer:** The *head* of a string or an array is the first element in the string or the array. The *tail* of a string or an array is all the elements (possibly none) except the head.

   (e) *Tail* and *nontail* recursive.

   Answer: A method is *tail recursive* if the only recursive call in the method is the last statement executed by the method. This needn't be the last statement contained in the method definition. A method is *nontail* recursive if it is not tail recursive, that is, execution of the recursive call is followed by the execution of other statements or there are more than one recursive call made.

2. Describe how the *method call stack* is used during a method call and return.

   Answer: The *method call stack* is a structure that is used to keep track of method calls and returns during program execution. When a method is called, a block containing information about that method call is placed on the top of *method call stack*. Among other things the information on the stack includes the method call's arguments, local variables, and the address to which the program should return when the method call is finished. A stack is a last-in-first-out structure, so the last method called will be on the top of the stack and will be the first block removed from the stack when a return statement is executed. When a return is executed, the return address is retrieved from the top block on the method call stack, and then that block is removed from the stack.

3. Why is a recursive algorithm generally less efficient than an iterative algorithm?

   Answer: Recursive algorithms use method calls to effect repetition. A loop, such as a for loop, does not. Method calls involve more computational overhead than a for loop, because a block representing the method call must be placed on the method call stack. This causes recursive algorithms to be somewhat less efficient than iterative ones.

4. A tree, such as a maple tree or pine tree, has a recursive structure. Describe how a tree's structure displays *self-similarity* and *divisibility*.

   Answer: The tree can be divided into branches (divisibility) each of which has the same general structure and appearance as the tree itself (self-similarity).

5. Write a recursive method to print each element of an array of `double`.

   Answer:

```
/**
 * printArr() prints each element of the array dArr
  * starting at first. To  print the array named myArray,
 * you would call this method with:  printArr(myArr, 0);
 * Algorithm: In addition to the array parameter we
 * need a second parameter to keep track of the
  * recursion. So first will start at 0 and increment on each
 *  recursive call until it reaches dArr.length.
 * @param dArr -- the array of double to be printed
 * @param first -- the index of the first element to be printed
 * Pre: 0 <= first <= dArr.length -1
 * Post: none
 */
public void printArr (double dArr[], int first) {
    if (first != dArr.length) {
        System.out.print(dArr[first] + " ");
        printArr(dArr, first + 1); // Recursive case
    }
    else
        System.out.println(); // Base case
} // printArr
```

6. Write a recursive method to print each element of an array of `double`, from the last to the first element.

   Answer:

```
/**
 * printRev() prints each element of the array dArr starting
 *   at last. To print the array named myArray, you would
 *   call this method with: printRev(myArr, myArr.length-1);
 * Algorithm: In addition to the array parameter we need a
 *  second parameter to keep track of the recursion. So last
 *  will start at dArr.length-1 and decrement on each
 * recursive call until it reaches 0.
 * @param dArr -- the array of double to be printed
 * @param last -- the index of the last element to be printed
 * Pre: 0 <= last <= dArr.length -1
 * Post: none
 */
public void printRev (double dArr[], int last) {
    if (last >= 0) {
        System.out.print(dArr[last] + " ");
        printRev(dArr, last - 1);  // Recursive case
    }
    else
        System.out.println();  // Base case
}  // printRev()
```

7. Write a recursive method that will concatenate the elements of an array of `String` into a single `String` delimited by blanks.

   Answer:

```
/**
 * concat() concatenates each element of the array strArr
 *  into a single string.  To call it, use: concat(myStrArr, 0);
 * Algorithm: In addition to the array parameter we need
 *  a second parameter to keep track of the recursion. So
 *  first will start at 0 and increment on each recursive call
 * until it reaches strArr.length.
 * @param strArr -- the array of strings to be concatenated
 * @param first -- index of first element to be concatenated
 * Pre: 0 <= first <= dArr.length -1
 * Post: none
 */
public String concat (String strArr[], int first) {
    if (first != strArr.length) {    // recursive case
        return strArr[first] + " " + concat(strArr,first + 1);
    }
    else
        return " ";  // base case
} // concat()
```

8. Write a recursive method that is passed a single `int` parameter, $N \geq 0$, and prints all the odd numbers between 1 and *N*.

   Answer:

```
/**
 * printOdds() prints the odd numbers between N and 0
 * Algorithm: The parameter N serves as the starting
 *  point  of the sequence, which is bounded by 0. N
 *  is decremented on each recursion.
 * @param N -- the starting point of the sequence
 * Pre: N >= 0
 * Post: none
 */
public void printOdds (int N) {
    if (N < 0 )
        System.out.println();  // base case
    else if (N % 2 != 0) {
        System.out.print(N + " ");
        printOdds(N - 1);  // recursive case
    } else
        printOdds(N - 1);  // recursive case
} // printOdds()
```

9. Write a recursive method that takes a single `int` parameter $N \geq 0$ and prints the sequence of even numbers between *N* down to 0.

   Answer:

```
/**
 * printEvens() prints the even numbers between N and 0
 * Algorithm: The parameter N serves as the starting point
 *  of the sequence, which is bounded by 0. N is
 *  decremented on each recursion.
 * @param N -- the starting point of the sequence
 * Pre: N >= 0
 * Post: none
 */
public void printEvens (int N) {
    if (N < 0 )
        System.out.println();    // base case
    else if (N % 2 == 0) {
        System.out.print(N + " ");
        printEvens(N - 1);  // recursive case
    } else
        printEvens(N - 1);   // recursive case
} // printEvens()
```

10. Write a recursive method that takes a single `int` parameter $N \geq 0$ and prints the multiples of 10 between 0 and *N*.

    Answer:

```
/**
 * printTens() prints the multiples of 10 between 0 and N.
 *  Call this method with: printTens(20);
 *  Algorithm: The sequence starts at 0 and is bounded
 *  by the parameter N.  N is decremented on each
 *  recursion. However, in order to print 10 * N starting
 *  at 0, note how we place the recursive call BEFORE
 *  the print statement. That way printing doesn't start
 *  until N reaches 0, and we get the sequence 0, 10, 20, ...
 * @param N -- the end point of the sequence
 * Pre: N >= 0
 * Post: none
 */
public void printTens (int N) {
    if (N >0)
        printTens(N - 1);   // Recursive case
    if (N % 10 == 0)
        System.out.print(N + " ");
} // printTens()
```

11. Write a recursive method to print the following geometric pattern:

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

Answer:

```
/**
 * printPattern1() prints a triangular pattern that
 * consists of 1 asterisk in the first row, 2 in the
 *  second, 3 in the third, and so on, with all lines left
 *   justified. The parameter N determines the number
 *  of rows and the number of asterisks per row.
 * Algorithm: In order to print 1 asterisk in the first row,
 *  we have recurse down until N is 1, then start printing.
 * The rows with 2, 3 and more asterisks will be printed
 *  after the row with 1, because they  recursion behaves
 * in a last-in-first-out fashion. Note that we use a for-loop
 *   to print the asterisks themselves.
 * @param N -- the number of rows in the pattern
 * Pre: N >= 0
 * Post: none
 */
public void printPattern1 (int N) {
    if (N <= 0)
        return;                     // Base case
```

```
        else {
            printPattern1(N-1);  // Recursive case
            for (int j = 0; j < N; j++) // print after recursion
                System.out.print("# ");
            System.out.println();
        }
    } // printPattern1()
```

12. Write recursive methods to print each of the following patterns.

```
   Pattern 2           Pattern3
---------------     ---------------

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

Answer:

```
    /**
     * printPattern2() prints a triangular pattern that
     * consists of N asterisks in the first row, N-1 in the
     * second, and so on down to 1 asterisk in the Nth
    *  row, with all rows right justified.
     * Algorithm: In this case printing precedes the recursive
     * call. But the K parameter controls the recursion, which
     * is bounded when K > N. K determines what row it is
     * and how many leading spaces to print. For loops
     *  are used to print the spaces and asterisks.
     * @param K -- the number of spaces in each row
     * @param N -- the number of rows in the pattern and
     *         number # per row
     * Pre: N >= 0 and 0 < K <= N
     * Post: none
     */
    public void printPattern2 (int K, int N) {
        if (K > N)
            return;  // Base case
        else {  // Recursive case
            for (int j = 1; j < K; j++)
                System.out.print("  "); //Print leading spaces
            for (int j = K; j <= N; j++)
                System.out.print("# ");  //  Print #s
            System.out.println();
```

```
        printPattern2(K+1, N);
    } // else
} // printPattern2()

/**
 * printPattern3() prints a triangular pattern that
 * consists of N asterisks in the first row, N-1 in the
 * second, and so on down to 1 asterisk in the Nth row,
 * with all rows left justified.
 * Algorithm: In this case printing precedes the
 * recursive call, and the recursion is bounded by 0.
 * There's no need for a second parameter here
 *  because there are no leading blanks that vary in
 *  each row as in pattern2.
 * @param N -- the number of rows in the pattern and
 *      number of # per row
 * Pre: N >= 0
 * Post: none
 */
public void printPattern3 (int N) {
    if (N < 0)
        return;  // Base case
    else {
        for (int j = 0; j < N; j++) // Recursive case
            System.out.print("# ");
        System.out.println();
        printPattern3(N - 1);
    }
} // printPattern3()
```

13. Write a recursive method to print all multiples of M up to M * N.

    Answer:

```
/**
 * printMultiples() prints the multiples of M between 0
 * and M * N. Call this method with: printMultiples(5, 20)
 * to get 0, 5, 10, ..., 100
 * Algorithm: The algorithm has two parameters, M and N
 * N is decremented on each recursion and printing
 * FOLLOWS the recursive call. So printing doesn't start until
 *  N reaches 0, and we get the sequence 0, 5, 10, 15, 20, ...
 * @param N -- the end point of the sequence
 * @param M -- the multiple by which each term is multiplied
 * Pre: N >= 0
 * Post: none
 */
public void printMultiples (int M, int N) {
    if (N >= 0) {
        printMultiples(M, N - 1); // Recursive case
```

```
            System.out.print(N * M + " ");
        }
        else
            System.out.println();  // Base case
    } // printMultiples()
```

14. Write a recursive method to compute the sum of grades stored in an array.
    Answer:

```
/**
 * sum() computes the sum of the values in an array
 * of double.
 * Algorithm: In addition to the array parameter we
 *  need a length parameter to keep track of the
 *  recursion. The array is summed from last down
 *  to first. Recursion stops when length == 0.
 * @param array -- array of double to be summed
 * @param length -- the length of the array
 * @return -- a double equal to the sum of the
 * elements of array
 * Pre: 0 <= length <= array.length -1
 * Post: The return value equals the sum of the
 * array elements
 */
public double sum(double array[], int length) {
  if (length == 0)     // Base case
    return 0;
  else
    return array[length-1] + sum(array,length - 1);
} // sum()
```

15. Write a recursive method to count the occurrences of a substring within a string.
    Answer:

```
/**
  * countSubStrs() counts occurrences of substr in
  *str starting at N.
  * Algorithm: Use str.indexOf(substr, N) to determine
  * if str contains substr, N's new value on each
  * recursion is 1+ str.indexOf(substr, N). Note that
  * there are two base cases, one when N runs off the
  * end of the string and two when there are no further
  * occurrences of substr.
  * @param substr -- the substring to be counted
  * @param str -- the string to be searched
  * @param N -- the starting location in the str,
  * should initially be 0.
  *
  */
```

```
public int countSubStrs(String substr,String str,int N) {
 if (N >= str.length())
   return 0;  // Base case, end of str
 else if (str.indexOf(substr, N) == -1)
   return 0;  // Base case, no more substr
 else      // Recursive case
   return 1 +
     countSubStrs(substr,str,str.indexOf(substr,N)+1);
} // countSubStrs()
```

16. Write a recursive method to remove the HTML tags from a string.

    Answer:

```
/** ex12.16
 * removeHTML() removes all HTML tags from its parameter,
 *  where an HTML tag is any sequence of characters between
 *  <  and > .
 * Algorithm: Use indexOf() to find the HTML tag. If there
 * are none, return the original string. If one is found,
 * remove it and return the resulting string. A new value
 * of str is passed on to the next level orecursion, which
 *  isstopped  when the parameter has no HTML tags.
 * @param str -- the string to be searched for HTML tags
 * PRE: If str contains a <, it contains a matching >
 * POST: The result contains no HTML tags.
 */
public String removeHTML(String str) {
   int p1 = str.indexOf("<");
   int p2 = str.indexOf(">");
   if (p1 == -1)
       return str;
   else {
       String tempS = str.substring(0,p1) +
                               str.substring(p2 + 1);
       return removeHTML(tempS);
   } // else
} // removeHTML()
```

17. Implement a recursive version of the `Caesar.decode()` method.

    Answer: Recursive versions of both `encode()` and `decode()` are provided in the implementation of the `Caesar` class given below.

```
/*
 *  File: Caesar.java
 *  Author: Java, Java, Java
 *  Description: This class provides an implementation of
 *  the Caesar cipher algorithm by implementing the encode()
 *  and decode() methods. A Caesar cipher translates each
 *  letter of the alphabet into another letter that is so
```

```
 *  3 letters ahead, wrapping around the end of the alphabet
 *  if necessary. For example 'a' would become 'd', and
 *   'z' would become 'c'.
 *  NOTE
 *  This version defines encode() and decode() recursively.
 */

public class Caesar extends Cipher {
    /**
     * encode(String word) -- recursively performs a Caesar
     *  shift on word where the shift is given as a parameter
     * Pre:  word != NULL
     * Post: each letter in word has been shifted
     */
    public String encode(String word) {
        if (word.length() == 0)
            return "";    // Base case
        else {
            char ch = word.charAt(0); //  Get first character
            //  Perform caesar shift
            ch = (char)('a' + (ch -'a'+ 3) % 26);
            //  Do rest of the word
            return ch + encode(word.substring(1));
        } // else
    } // encode()

    /*
     * decode(String word) ---  performs a reverse Caesar
     *  shift on word where the shift is fixed at 3.
     * Pre:  word != NULL
     * Post: each letter in word has been shifted by 26-3
     */
     public String decode(String word) {
        if (word.length() == 0)
            return "";  // Base case
        else {
            char ch = word.charAt(0); // Get first character
            //  Perform reverse caesar shift
            ch = (char)('a' + (ch - 'a' + 23) % 26);
             //  Do rest of the word
            return ch + decode(word.substring(1));
        } // else
     } // decode()

} // Caesar
```

18. The Fibonacci sequence (named after the Italian mathematician Leonardo of Pisa, ca. 1200) consists of the numbers `0,1,1,2,3,5,8,13,...` in which each number (except for the first two) is the sum of the two preceding numbers. Write a recursive method `fibonacci(N)` that prints the *N*th Fibonacci

number.

Answer:

```
/*
 * File: Fibonacci.java
 * Author: Java, Java, Java
 * Description: The  recursive method  fibonacci(N)
 * computes the Nth Fibonacci number.   The Fibonacci
 * sequence begins with 0 and 1 and each successive
 * term is the sum of the two previous terms.
 */

public class Fibonacci {

   /**
    * fibonacci() returns the first Nth Fibonacci number
    * @param N -- the index in the fibonacci sequence
    * @return -- the Nth value in the sequence
    */
    public int fibonacci(int N) {
        if (N == 0 || N == 1)
            return N;
        else
           return fibonacci(N - 1) + fibonacci(N - 2);
    } // fibonacci()

   /**
    * main() creates an instance of this class and tests its
    *  fibonacci() method on values from 0 to 20
    */
   public static void main(String argv[]) {
       Fibonacci tester = new Fibonacci();
       for (int k = 0; k <= 20; k++){
           System.out.print("The " + k + "th Fibonacci number" );
           System.out.println(" is "+ tester.fibonacci(k));
       } // for
   } // main()

} // Fibonacci class
```

19. Write a recursive method to rotate a `String` by *N* characters to the right. For example, `rotateR("hello", 3)` should return "llohe."

    Answer: The following class contains methods that solve this exercise and also the next exercise.

```
/*
 * File: Rotater.java
 * Author: Java, Java, Java
 * Description: This class implements methods to rotate strings.
```

```
 */

public class Rotater {

   /** ex12.19
    * rotateR() recursively rotates the characters in string
    * parameter N characters to the right.
    * Algorithm: Each level of recursion rotates by 1 character.
    *  To rotate right, take the last character and concatenate
    *   to the left of the remainder. So "hello" goes to "ohell".
    * @param s -- the String to be rotated  -- e.g., hello
    * @param N -- the number of rotations   -- e.g., 3
    * @return -- the rotated string         -- e.g., llohe
    */
    public String rotateR(String s, int n) {
        if (n <= 0)
            return s;
        else
            return rotateR( s.substring(s.length()-1) +
                        s.substring(0,s.length()-1), n - 1 );
    } // rotateR()

   /** ex12.20
    * rotateL() recursively rotates the characters in string
    * parameter N characters to the left.
    * Algorithm: Each level of recursion rotates by 1 character.
    *  To rotate left, take the first character and concatenate
    *  to the right of the remainder. So "ello" goes to "elloh".
    * @param s -- the String to be rotated  -- e.g., hello
    * @param N -- the number of rotations   -- e.g., 3
    * @return -- the rotated string         -- e.g., lohel
    */
    public String rotateL(String s, int n) {
        if (n <= 0)
            return s;
        else
            return rotateL( s.substring(1) +
                                s.substring(0,1), n - 1 );
    } // rotateR()

   /**
    * main() creates an instance of this class and tests
    * the rotateR() and rotateL() methods.
    */
   public static void main(String argv[]) {
       String str = "hello";
       Rotater r = new Rotater();
       for (int k = 0; k <= 5; k++){
           System.out.print("Rotate L/R by " + k + " gives ");
           System.out.print(r.rotateL(str, k) + " ");
```

```
            System.out.println(r.rotateR(str, k));
        } // for
    } // main()

} // Rotater class
```

20. Write a recursive method to rotate a `String` by *N* characters to the left. For example, `rotateL("hello", 3)` should return "lohel."

    Answer: The `rotateL()` method appears in the `Rotater` class of the solution of the previous exercise.

21. Write a recursive method to convert a `String` representing a binary number to its decimal equivalent. For example, `binTodecimal("101011")` should return the `int` value 43.

    Answer:

```
/* ex12.21
 * binToDecimal() takes a string of bits "101011" and
 *  treats it like a binary number, converting it to
 *  its decimal equivalent.
 * Algorithm: The bit string is processed from right
 * to left. It uses the fact that 11001 which equals
 * 25 is equivalent to 1 + 2 * 12, that is it is the
 * rightmost bit plus twice the value of the remaining
 * leftmost bits (1100 which is decimal 12).
 * @param bitString -- a String of 0s and 1s
 * @return -- int giving the decimal value of bitStr.
 * Pre: 0 <= bitString.length() and bitString contains
 *    only 0s and 1s
 * Post: return value equals the decimal value of bitStr.
 */
public int binToDecimal(String bitStr) {
    if (bitStr.length() == 0)  // Base case
      return 0;
    else { // Get the rightmost bit
      char ch = bitStr.charAt(bitStr.length() - 1);
      int binValue = 0;
      if (ch == '1')    // Get the bit's value
        binValue = 1; // Return 1 + 2 * (left substring)
      return binValue + 2 *
        binToDecimal(bitStr.substring(0,bitStr.length() - 1));
    } // else
} // binToDecimal()
```

22. A palindrome is a string that is equal to its reverse — "mom," "i," "radar" and "able was i ere i saw elba." Write a recursive `boolean` method that determines whether its `String` parameter is a palindrome.

    Answer: The `Palindrome` class below has a recursive `reverse()` method that is used in a non-recursive `isPalindrome()` method.

```
/*
 * File: Palindrome.java
 * Author: Java, Java, Java
 * Description: This class tests whether strings are
 * palindromes. A palindrome is a string that is
 * spelled the same backwards and forwards. Examples:
 * radar, mom, i, dad, maam, able was i ere i saw elba.
 */

public class Palindrome {

   /**
    * isPalindrome() returns true if its parameter is
    * a palindrome
    * @param s -- a string to test
    * @return -- boolean true if s is a palindrome
    */
    public boolean isPalindrome(String s) {
       // Just compare with its reverse
        return s.equals(reverse(s));
    } //

   /**
    * reverse() recursively reverses its string parameter
    * @param s -- the string to be reversed
    * @return a string giving s in reverse order
    * Pre:  s != null
    * Post: reverse(s) = the characters of s in
    *  reverse order
    */
   public String reverse(String s) {
      if (s.length() <= 1)  // Base case
         return s;
      else                  // Recursive case
         return reverse( s.substring(1)) + s.charAt(0);
   }

   /**
    * main() creates an instance of this class and tests its
    * isPalindrome() method on several strings.
    */
   public static void main(String argv[]) {
     Palindrome tester = new Palindrome();
     String[] strArr = new String[4];
     strArr[0] = "mama";
     strArr[1] = "radar";
     strArr[2] = "able";
     strArr[3] = "able was I ere I saw elba";
     for (int k = 0; k < strArr.length; k++){
       if (tester.isPalindrome(strArr[k]))
```

```
            System.out.println(strArr[k] + " is a palindrome");
          else
            System.out.println(strArr[k] + " is NOT a palindrome");
        } // for
      } // main()

} // Palindrome
```

23. **Challenge:** Incorporate a `drawBinaryTree()` method into the `RecursivePatterns` program. A level-1 binary tree has two branches. At each subsequent level, two smaller branches are grown from the endpoints of every existing branch. The geometry is easier if you use 45 degree angles for the branches. The Figure 12.36 in the text shows a level-4 binary tree drawn upside down.

    Answer: In general, the following set of instructions can be used to draw a basic binary tree:

    ```
    Draw the trunk branches
    Draw a smaller left subtree
    Draw a smaller right subtree
    ```

    The idea here is that the left and right subtrees are just smaller versions of the tree itself. That is where the recursion comes in. As in our other recursive graphics we use parameters to control the location and size of the drawing, and the number of levels of recursion.

    For a binary tree, all of these parameters must change in value at each level of recursion. The level decreases by 1. It is bounded by 0. The size will become half the original size at every level. The location of the level k-1 subtrees will be at the ends of the branches of the level k subtree. This leads to the following definition of the `drawBinaryTree()` method:

    ```
    /**
     * drawBinaryTree() draws a Binary Tree recursively
     * @param g -- graphics object
     * @param level -- the number of levels of recursion
     * @param p --  reference Point (location) of the pattern
     * @param h -- the height of the Tree
     * Description: Draw two branches. Then if the recursion
     *  level is > 0, draw two binary trees with height h/2, one
     *  from the end of the left branch and one from the end
     * of the right branch.
     */
     void drawBinaryTree(Graphics g, int level, Point p, int h) {
            // No matter what level, draw two branches
        g.drawLine(p.x, p.y, p.x - h, p.y + h);
        g.drawLine(p.x, p.y, p.x + h, p.y + h);
        if (level > 0) {
            Point newP1 = new Point(p.x - h, p.y + h);
    ```

```
            Point newP2 = new Point(p.x + h, p.y + h);
                // draw 1/2 size tree at branch end
            drawBinaryTree(g, level - 1, newP1, h/2);
                // draw 1/2 size tree  at branch end
            drawBinaryTree(g, level - 1, newP2, h/2);
        } // if
    } // drawBinaryTree
```

To modify the Recursive Patterns project (from Chapter 12), the `drawBinaryTree()` method should be added to the `Canvas.java` class. In addition to the method itself, you should define new class constants to give the initial size and location of the tree, and you must modify the `paintComponent()` method which calls the new method. You must also modify `RecursivePatterns.java`, adding the new pattern as a choice box menu item. These changes are shown below. For completeness, the index.html file that is needed to view the applet is also listed:

```
/*
 * File: Canvas.java
 * Author: Java, Java, Java
 * Description: This subclass of JPanel performs all
 *  the drawing functions for the RecursivePatterns applet.
 *  Whenever the user selects a pattern, the applet invokes
 *  the setPattern() method and then calls its repaint()
 *  method. Since the Canvas is contained within the applet,
 *  it too will be repainted, by its paintComponent() method,
 *  which draws the currently selected pattern.
 */

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

public class Canvas extends JPanel {
    public static final int WIDTH=400, HEIGHT=400;
    private final int HBOX=10, VBOX=50;
    private final int BOXSIDE=200, BOXDELTA=10;
    private final int HTREE=200, VTREE=50, TREEHGT = 100;
          // Initial gasket points
    private final Point gasketP1 = new Point(10, 280);
    private final Point gasketP2 = new Point(290, 280);
    private final Point gasketP3 = new Point(150, 110);
    private int pattern = 0 ;     // Current pattern
    private int level = 4;          // Current level

    /**
     *  Canvas() constructor sets its size
     */
    public Canvas() {
        setSize(WIDTH, HEIGHT);
```

```
}

/**
 *  setPattern() is invoked by the applet to set the
 *   current drawing pattern and level
 *  @param pat -- the pattern to be drawn
 *  @param lev -- the number of levels of recursion
 */
public void setPattern(int pat, int lev) {
    pattern = pat;
    level = lev;
}

/**
 *  paintComponent() is invoked automatically whenever
 *  the Canvas needs to be painted -- e.g., when its
 *  containing applet is painted
 *  @param g -- the Canvas graphics object.
 */
public void paintComponent(Graphics g) {
    g.setColor(getBackground());
        // Redraw the panel's background
    g.drawRect(0, 0, WIDTH, HEIGHT);
    g.setColor(getForeground());
    switch (pattern) {
    case 0:
        drawGasket(g, level, gasketP1, gasketP2, gasketP3 );
        break;
    case 1:
        Point pt1 = new Point(HBOX, VBOX);
        drawBoxes(g, level, pt1, BOXSIDE, BOXDELTA );
        break;
    case 2:
        Point pt2 = new Point(HTREE, VTREE);
        drawBinaryTree(g, level, pt2, TREEHGT);
        break;
    } // switch
} // paintComponent()

/**
 * drawGasket() --- recursively draws the Sierpinski gasket
 *  pattern, with points p1, p2, p3, representing the vertices
 *  of its enclosing triangle.
 * @param p1 -- a Point for a vertex of the enclosing triangle
 * @param p2 -- a second Point of the enclosing triangle
 * @param p3 -- a third Point of the enclosing triangle
 * @param level (>= 0) recursion parameter (base case:  0)
 */
private void drawGasket(Graphics g, int lev, Point p1,
                        Point p2, Point p3) {
```

```
    g.drawLine(p1.x, p1.y, p2.x, p2.y);  // Draw a triangle
    g.drawLine(p2.x, p2.y, p3.x, p3.y);
    g.drawLine(p3.x, p3.y, p1.x, p1.y);
    if (lev > 0) {// If morelevels, draw 3 smaller gaskets
        Point midP1P2 =
            new Point( (p1.x + p2.x) / 2, (p1.y + p2.y) / 2 );
        Point midP1P3 =
            new Point( (p1.x + p3.x) / 2, (p1.y + p3.y) / 2 );
        Point midP2P3 =
            new Point( (p2.x + p3.x) / 2, (p2.y + p3.y) / 2 );
        drawGasket(g, lev - 1, p1, midP1P2, midP1P3);
        drawGasket(g, lev - 1, p2, midP1P2, midP2P3);
        drawGasket(g, lev - 1, p3, midP1P3, midP2P3);
    }
} // drawGasket()

/**
 * drawBoxes() --- recursively draws a pattern of nested
 *  squares with loc as the top left corner of outer square
 *  and side being the length square's side.
 * @param loc is the top-left Point of the outer square
 * @param side is the length of the square's side
 * @param level (>= 0) is recursion parameter (base case: 0)
 * @param delta is amount  a side is adjusted at each level
 */
private void drawBoxes(Graphics g, int level, Point loc,
                       int side, int delta) {
    g.drawRect(loc.x, loc.y, side, side );
    if (level > 0) {
        Point newLoc = new Point( loc.x + delta, loc.y + delta);
        drawBoxes(g, level - 1, newLoc, side - 2 * delta, delta);
    }
} // drawBoxes()

/**
 * drawBinaryTree() draws a Binary Tree recursively
 * @param g -- graphics object
 * @param level -- the number of levels of recursion
 * @param p --  reference Point (location) of the pattern
 * @param h -- the height of the Tree
 * Description: Draw two branches. Then if the recursion
 *  level is > 0, draw two binary trees with height h/2, one
 *  from the end of the left branch and one from the end
 * of the right branch.
 */
 void drawBinaryTree(Graphics g, int level, Point p, int h) {
         // No matter what level, draw two branches
     g.drawLine(p.x, p.y, p.x - h, p.y + h);
     g.drawLine(p.x, p.y, p.x + h, p.y + h);
     if (level > 0) {
```

```
            Point newP1 = new Point(p.x - h, p.y + h);
            Point newP2 = new Point(p.x + h, p.y + h);
                // draw 1/2 size tree at branch end
            drawBinaryTree(g, level - 1, newP1, h/2);
                // draw 1/2 size tree  at branch end
            drawBinaryTree(g, level - 1, newP2, h/2);
        } // if
    } // drawBinaryTree

} // Canvas

/*
 * File: RecursivePatterns.java
 * Author: Java, Java, Java
 * Description: This applet provides a GUI interface for
 *  for the Canvas class, which draws recursive patterns
 *  selected by the user. Two JComboBoxes are provided,
 *  one to select a pattern, and one to select a level of
 *  recursion. Whenever the user makes a selection, the
 *  itemStateChanged() method invokes canvas.setPattern()
 *  to draw the selected pattern.
 */

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

public class RecursivePatterns extends JApplet
                             implements ItemListener  {
        // Pattern choices
     private String choices[] = {"Sierpinski Gasket",
                            "Nested Boxes", "Binary Tree"};
    private JComboBox patterns =
        new JComboBox(choices);
    private JComboBox levels =
        new JComboBox(); //Level choices
    private Canvas canvas =
        new Canvas();  // Drawing panel
    private JPanel controls = new JPanel();

    /**
     * init() sets up the applet's interface. Two JComboBoxes
     *   are created and given items. Note the use the border
     *   around the drawing canvas. Each JComboBox
     *  is assigned the applet itself as its ItemListener.
     */
    public void init() {
        for (int k=0; k < 10; k++) // Add 10 levels
            levels.addItem(k + "" );
```

```
        // Initialize the menus
        patterns.setSelectedItem(choices[0]);
        levels.setSelectedItem("4");
        TitledBorder bdr =
           BorderFactory.createTitledBorder("Canvas");
        canvas.setBorder(bdr);
        controls.add(levels);    // Control panel for menus
        controls.add(patterns);
              // Add the controls
        getContentPane().add(controls,"North");
              // And the drawing panel
        getContentPane().add(canvas,"Center");
              // Register the menus with a listener
        levels.addItemListener( this );
        patterns.addItemListener( this );
        setSize(canvas.WIDTH,
                 canvas.HEIGHT+controls.getSize().width);
    } // init()

    /**
     * itemStateChanged() is the only method of ItemListener
     *  interface. It is invoked whenever user makes a selection
     *  in a JComboBox. In this case the selections from both
     * JComboBoxes are passed along to canvas.setPattern().
     * Repainting applet causes all its contained components
     *  to be repainted, including the canvas.
     * @param e -- ItemEvent that describes user's selection.
     */
    public void itemStateChanged(ItemEvent e) {
        canvas.setPattern(patterns.getSelectedIndex(),
                        levels.getSelectedIndex());
        repaint();   // Repaint the applet
    } // itemStateChanged()
} // RecursivePatterns

<! -- ******** HTML FILE : index.html  for ex12.23 ******************
file:   index.html
author: Java Java Java
-->

<html>
<head><title>RecursivePatterns</title></head>
<body>
<hr>
<APPLET  CODE = RecursivePatterns.class
          WIDTH = 400 HEIGHT = 400 >
</APPLET>
<hr>
<h4>Source Code
<br>
```

```
<li>
        <a href="RecursivePatterns.java">
         RecursivePatterns.java
         </a>
<br>
<li><a href="Canvas.java">Canvas.java</a>
</body>
</html>
```

24. **Challenge: Towers of Hanoi.** According to legend, some Buddhist monks were given the task of moving 64 golden disks from one diamond needle to another needle, using a third needle as a backup. To begin with, the disks were stacked one on top of the other, from largest to smallest (Text Figure 12.37). The rules were that only one disk can be moved at a time, and that a larger disk can never go on top of a smaller one. The end of the world was supposed to occur when the monks finished the task!

    Write a recursive method, `move(int N, char A, char B, char C)`, that will print out directions the monks can use to solve the Towers of Hanoi problem. For example, here's what it should output for the three-disk case, `move(3,'A','B','C'`:

    ```
    Move 1 disk from A to B.
    Move 1 disk from A to C.
    Move 1 disk from B to C.
    Move 1 disk from A to B.
    Move 1 disk from C to A.
    Move 1 disk from C to B.
    Move 1 disk from A to B.
    ```

    Answer:

```
/*
 * File: Hanoi.java
 * Author: Java, Java, Java
 * Description: Provides a text-based simulation of the Towers
 *of Hanoi problem.
 */

public class Hanoi {

   /**
    * Hanoi(N) this constructor automatically calls the move()
    * method using the integer N. This prints out directions for
    * solving the Tower of hanoi problem for N disks.
    * @param N -- the number of disks
    */
    public Hanoi (int N) {
        move (N, 'A', 'B', 'C');
    } // Hanoi() constructor
```

```
    /**
     * move() simulates the process of recursively moving N
     * disks from A to B using C as a temporary storage
     * location. In the recursive case it moves N-1 disks from A
     * to C, leaving 1 disk on A. It then moves that disk from A
     * to B. Then it moves the N-1 disks it had placed on C
     * onto B. The effect is that all N disks have been moved
     * from A to B.
     * @param N -- the number of disks
     */
     public void move(int N, char A, char B, char C) {
         if (N == 1)
             System.out.println("Move a disk from " + A + " to " + B);
         else {
             move(N-1, A, C, B);   // Move top of stack to C
             System.out.println("Move a disk from " + A + " to " + B);
             move(N-1, C, B, A);   // Replace top of stack to B
         }
     }

    /**
     * main() creates an instance of this class and automatically
     * calls the move() method to write the instructions.
     */
    public static void main(String argv[]) {
        Hanoi tester = new Hanoi(3);
    } // main()

} // Hanoi
```

Chapter 13

Graphical User Interfaces

1. Explain the difference between the following pairs of terms.

   (a) A *model* and a *view*.

   Answer: The *model* refers to a component's internal state and consists of such properties as whether it is enabled and visible. The *view* refers the component's appearance. For example, when it is disabled, it should appear shaded.

   (b) A *view* and a *controller*.

   Answer: The *view* refers to the component's appearance. The *controller* monitors the component's state and the actions that involve the component. For example, a button's controller would change its state and appearance when it is clicked.

   (c) A *lightweight* and *heavyweight* component.

   Answer: AWT components are considered *heavyweight* because they depend on *peer* classes that are written in the native system rather than in Java itself. Except for the top-level window components — `JWindow`, `JDialog`, `JFrame`, and `JApplet` — all Swing components are *lightweight*, meaning they are programmed entirely within Java code.

   (d) A `JButton` and a `Button`.

   Answer: A `JButton` is a Swing version of a button whereas a `Button` is an AWT button.

   (e) A *layout manager* and a *container*

   Answer: A *layout manager* is an object that manages the relative size and positioning (layout) of components in a *container*.

   (f) A *containment hierarchy* and an *inheritance hierarchy*.

   Answer: The *containment hierarchy* describes which components are contained in which containers. An *inheritance hierarchy* describes the superclass/subclass relationship among classes — that is, which classes are derived from which other classes.

(g) A *content pane* and a `JFrame`.

Answer: A *content pane* is a built-in a `JPanel` that serves as the working area within a `JFrame`. Rather than adding Swing components directly to the `JFrame`, they are added to the content pane.

2. Fill in the blank:

(a) The GUI component that is written entirely in Java is known as a _____ component. **Answer: lightweight**

(b) The AWT is not platform independent because it uses the _____ model to implement its GUI components. **Answer: peer**

(c) The visual elements of a GUI are arranged in a _____. **Answer: containment hierarchy**

(d) A _____ is an object that takes responsibility for arranging the components in a container. **Answer: layout manager**

(e) The default layout manager for a `JPanel` is _____. **Answer: flow layout**

(f) The default layout manager for a `JApplet` is _____. **Answer: border layout**

3. Describe in general terms what you would have to do to change the standard look and feel of a Swing `JButton`.

Answer: An object's look and feel is defined by its view and its controller. To simplify changing the look and feel, for most classes the view and controller are combined into a single class. For example, for a `JButton` you would redefine the methods in the `BasicButtonUI` to implement your own look and feel.

4. Explain the differences between the model-view-controller design of a `JButton` and the design of an AWT `Button`. Why is MVC superior?

Answer: In the model-view-controller design, each of these aspects are treated separately. In a AWT `Button` the components state, view, and control methods are combined into one class. Swing components are more flexible. By redefining the controller and view you can change their look and feel. Not so for AWT components.

5. Suppose you have an applet that contains a `JButton` and a `JLabel`. Each time the button is clicked the applet rearranges the letters in the label. Using Java's event model as a basis, explain the sequence of events that happens in order for this action to take place.

Answer: When the user clicks on the button, that generates an `ActionEvent` which is passed by Java virtual machine (JVM) to the button's `ActionListener`. The `ActionListener` executes an `actionPerformed()` method, which should contain code (or a method call) to rearrange the `JLabel`'s letters.

6. Draw a containment hierarchy for the most recent GUI version of the `OneRowNim` program.

   Answer: The `OneRowNimGUI` class in Figure 4.25 has the following containment hierarchy.

```
      OneRowNimGUI
          JFrame
            |
       ---------------------
       |         |          |
 JTextArea   JTextField   JButton
 display      inField     goButton
```

7. Create a GUI design, similar to the one shown in Figure 13.25 in the text, for a program that would be used to buy tickets on-line for a rock concert.

   Answer: A GUI design for an on-line rock concert ticket program is shown in Figure 13.1.

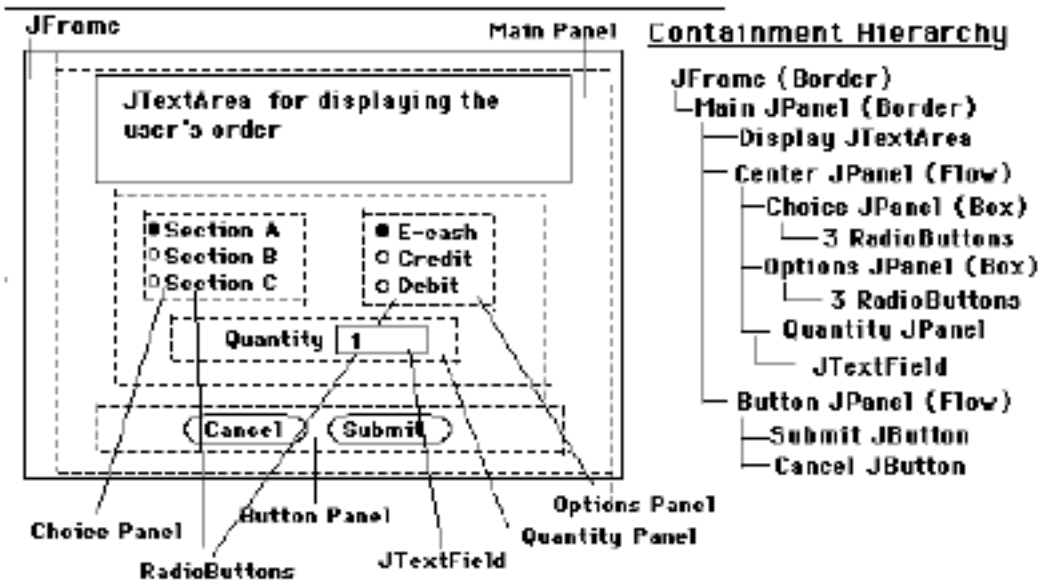


Figure 13.1: Exercise 7: A GUI design for ordering tickets online.

8. Create a GUI design, similar to the one shown in Figure 13.25 in the text, for an on-line program that would be used to play musical recordings.

   Answer: A GUI design for an on-line program to play musical recordings is shown in Figure 13.2.

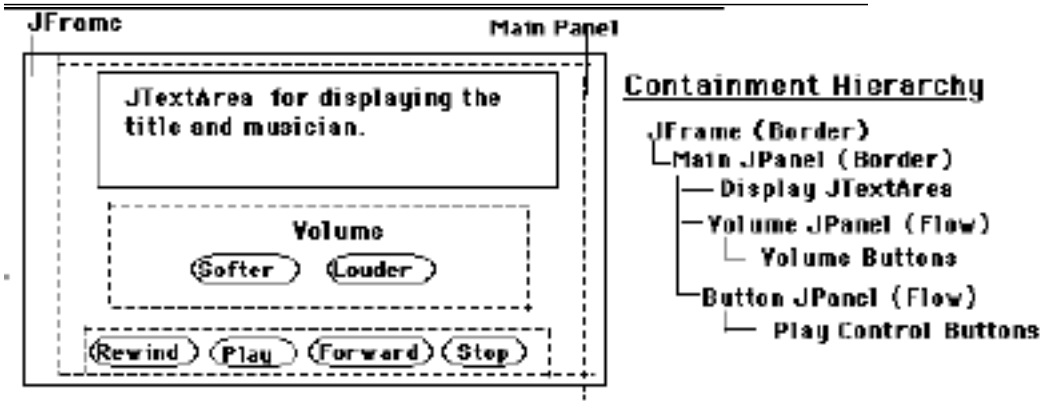


Figure 13.2: Exercise 8: A GUI design for an online audiotape player.

9. Design and implement a GUI for the `BankCD` program in Self-Study solution 5.20. This program should let the user input the interest rate, principal, and period, and should accumulate the value of the investment.

   Answer: The GUI interface for the solution below is shown in Figure 13.3. The `BankCDGUI` class uses the `BankCD` class from Chapter 5 which is not listed below.

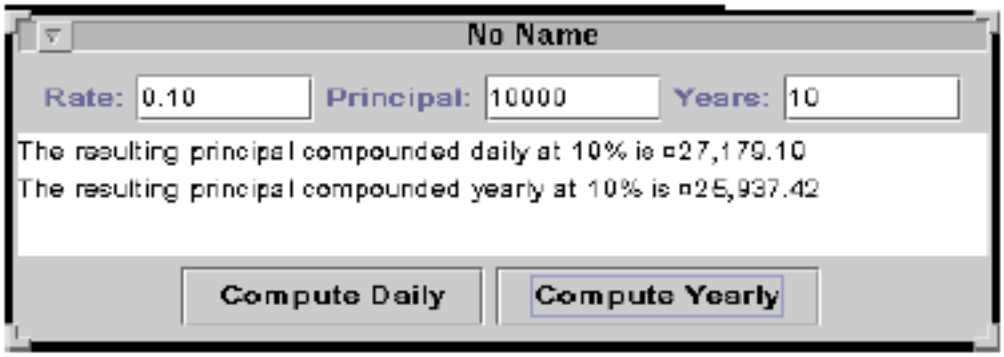


Figure 13.3: Exercise 9: A graphical user interface for the BankCD program

```
/*
 * File: BankCDGUI.java
 * Author: Java, Java, Java
 * Description: This application uses Javax.swing.*
 * classes to create a GUI interface to the BankCD
 * program.
```

```
 */

import javax.swing.*;
import java.awt.event.*;
import java.text.NumberFormat;
    // For formatting as $nn.dd an n%

public class BankCDGUI extends JFrame
                    implements ActionListener {

    private JLabel rateLabel = new JLabel("Rate:");
    private JTextField rateFld = new JTextField(8);
    private JLabel prinLabel = new JLabel("Principal:");
    private JTextField prinFld = new JTextField(8);
    private JLabel yrsLabel = new JLabel("Years:");
    private JTextField yrsFld = new JTextField(8);
    private JButton dailyButton =
        new JButton("Compute Daily");
    private JButton yearlyButton =
        new JButton("Compute Yearly");
    private JPanel controls = new JPanel();
    private JPanel inputs = new JPanel();
    private JTextArea display = new JTextArea();

    /**
     * BankCDGUI() constructor creates the layout for the
     * user interface. It uses three panels to organize the
     * GUI components and uses the default layout managers.
     */
    public BankCDGUI() {
             // Control buttons
        dailyButton.addActionListener(this);
        yearlyButton.addActionListener(this);
        controls.add(dailyButton);
        controls.add(yearlyButton);
             // Input fields
        inputs.add(rateLabel);
        inputs.add(rateFld);
        inputs.add(prinLabel);
        inputs.add(prinFld);
        inputs.add(yrsLabel);
        inputs.add(yrsFld);
            // Border layout
        getContentPane().add(inputs, "North");
        getContentPane().add(display, "Center");
        getContentPane().add(controls, "South");
    } // CDInterest

    /**
     *  actionPerformed() handles actions on the two
```

```
     *  buttons and computes  the program's results.
     *  Note the use of the NumberFormat.format() methods
     *   for making the output look pretty.
     */
    public void actionPerformed(ActionEvent e) {
            // declare variables
        double principal; // The CD's initial principal
        double rate;      // CD's interest rate
        double years;     // Number of years to maturity
        double result;    // Accumulated principal
             // Set up formats
        NumberFormat dollars =
            NumberFormat.getCurrencyInstance();
        NumberFormat percent =
            NumberFormat.getPercentInstance();
        percent.setMaximumFractionDigits(2);
              // Get data
        principal = Double.parseDouble(prinFld.getText());
        rate = Double.parseDouble(rateFld.getText());
        years = Double.parseDouble(yrsFld.getText());
             // create BankCD object
        BankCD theCD = new BankCD(principal,rate,years);

        if (e.getSource() == dailyButton) {
                  // Calculate results
            result = theCD.calcDaily();
            display.append("The principal compounded daily at " +
                percent.format(rate) + " is " +
                dollars.format(result) + "\n");
        } else {
            result = theCD.calcYearly();
            display.append("The principal compounded yearly at " +
                percent.format(rate) + " is " +
                dollars.format(result) + "\n");
        } // else
    } // actionPerformed()

    /**
     *   main() creates an instance of the BankCDGUI object.
     *   Note the use of the WindowAdapter to handle window
     *   close events.
     *   @param args[] is an array of Strings not used here.
     */
    public static void main( String args[] ) {
        BankCDGUI bcdgui = new BankCDGUI();
        bcdgui.setSize(450,300);
        bcdgui.setVisible(true);
        bcdgui.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
```

```
            }
        });
    } // main()
} // BankCDGUI
```

10. Design and implement a GUI for the `Temperature` class (Text Figure 5-5). One challenge of this design is to find a good way for the user to indicate whether a Fahrenheit or Celsius value is being input. This should also determine the order of the conversion: F to C, or C to F.

    Answer: The GUI interface for the solution below is shown in Figure 13.4. The `TemperatureGUI` class uses the `Temperature` class from Chapter 5 which is not listed below.

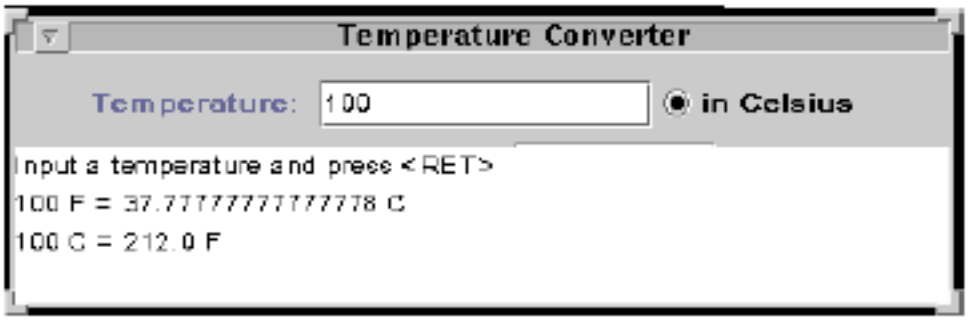


Figure 13.4: Exercise 10: A graphical user interface for the Temperature program

```
/*
 * File: TemperatureGUI.java
 * Author: Java, Java, Java
 * Description: This class provides a GUI to test the
 * Temperature class. The GUI consists of a text field,
 * into which the user can type a  temperature value, a
 * radio button, which can be toggled from celsius to
 * fahrenheit to indicate the units of the input value,
 *  and a text area where the results are displayed.
 */

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

public class TemperatureGUI extends JFrame
                        implements ActionListener  {
         // Input panel
    private JPanel inputPanel = new JPanel();
```

```
    // GUI components
private JTextField inField = new JTextField(15);
private JTextArea display = new JTextArea();
private JLabel prompt = new JLabel("Temperature: ");
private JRadioButton celsUnit =
                   new JRadioButton("in Celsius");
private JRadioButton fahrUnit =
                 new JRadioButton("in Fahrenheit");
private JButton convert = new JButton("Convert");
private ButtonGroup unitsGroup = new ButtonGroup();

/**
 *  TemperatureGUI() sets the layout and adds
 * components to the top-level JFrame.
 */
public TemperatureGUI(String title)  {
    super(title);
    setSize(250,200);
    getContentPane().setLayout(new BorderLayout());
        // Input elements
    inputPanel.add(prompt);
    inputPanel.add(inField);
        // Register text field with listener
    inField.addActionListener(this);
    inputPanel.add(celsUnit); // Radio buttons
    inputPanel.add(fahrUnit);
    unitsGroup.add(celsUnit); // Button group
    unitsGroup.add(fahrUnit);
    fahrUnit.setSelected(true);
    inputPanel.add(convert);
    convert.addActionListener(this);
    getContentPane().add(inputPanel, "North");
    getContentPane().add(display, "Center");
    display.setText("Input a temperature and press button\n");
} // TemperatureGUI()

/**
 * actionPerformed() handles input into the text
 * field. The  TextField input must be converted
 * from String to double.
 * @param e -- the ActionEvent that caused this
 *   method to be called
 */
public void actionPerformed(ActionEvent e) {
        // Get user's input
    String inputStr = inField.getText();
        // Convert it to double
    double userInput =
        Double.parseDouble(inputStr);
    double result = 0;
```

```
        if (celsUnit.isSelected()) { // Process and report
            result = Temperature.celsToFahr(userInput);
            display.append(inputStr+" C = "+result+" F \n");
        } else {
            result = Temperature.fahrToCels(userInput);
            display.append(inputStr+" F = "+result+" C \n");
        }
    } // actionPerformed()

    /**
     * main() creates an instance of this class and sets the
     * size and visibility of  its JFrame. An anonymous class is
     * used to create an instance of the WindowListener  class,
     * which handles the window close events for the application.
     * @param args - an array of strings which is not used here.
     */
    public static void main(String args[]) {
        TemperatureGUI f =
            new TemperatureGUI("Temperature Converter");
        f.setSize(700, 300);
        f.setVisible(true);
           // Quit the application
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()

} // TemperatureGUI
```

11. Design an interface for a 16-button integer calculator that supports addition, subtraction, multiplication, and division. Implement the interface so that the label of the button is displayed in the calculator's display — that is, it doesn't actually do the math.

    Answer: The GUI interface for the solution below is shown in Figure 13.5. The `CalculatorGUI` class below uses the `Calculator` class which is listed below as a solution of the next exercise.

```
/*
 *  File: CalculatorGUI.java
 *  Author: Java, Java, Java
 *  Description: This class creates an user interface for
 *  a four-function integer calculator. The keys are arranged
 *  in a 4 x 4 grid, which is at the center of the Frame's
 *  border layout. At the north of the border  layout is the
 *  calculator's accumulator/display, implemented with a text
 *  field. This version creates an instance of the Calculator
```

Figure 13.5: Exercise 11: An interface for a four function integer calculator.

```
 *  class which performs the operations associated with the
 *  interface. See the actionPerformed() method.
 */

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

public class CalculatorGUI extends JFrame
                             implements ActionListener{
    private final static int NBUTTONS = 16; // Constants

    private JPanel keyPadPanel; // Panel to hold keyPad
    private JTextField accumulator; // Calculator's display
    private JButton keyPad[]; // Internal keyPad array

    private String label[] =  // 16 keyPad Button Labels
                              {  "1","2","3","+",
                                 "4","5","6","-",
                                 "7","8","9","*",
                                 "C","0","=","/"  };

    private Calculator calcMachine =
                   new Calculator(); // The calculator

    /**
     * CalculatorGUI() constructor sets up the GUI.
```

```
     */
    public CalculatorGUI(String title) {
                // Set up the keypad grid layout
        super(title);
        keyPadPanel = new JPanel();
        keyPadPanel.setLayout(new GridLayout (4,4,1,1));
                // Create an array of buttons
        keyPad = new JButton [NBUTTONS];
        for (int k = 0; k < keyPad.length; k++){
              // For each array slot Create a button
            keyPad[k] = new JButton( label[k] );
              // And a listener for it
            keyPad[k].addActionListener(this);
              // And add it to the panel
            keyPadPanel.add(keyPad[k]);
        } // for
  // Set up the accumulator display
        accumulator = new JTextField("0",20);
        accumulator.setEditable (false);
           // Add components to the frame with Border layout
        getContentPane().setLayout(new BorderLayout(10, 10));
        getContentPane().add("North", accumulator);
        getContentPane().add("Center", keyPadPanel);
    } // CalculatorGUI()

    /**
     * actionPerformed() handles all the action on the
     * calculator's keys. In this version the key plus the
     * current value of the calculator's display are passed
     * to the Calculator object, which performs an operation
     * and returns a result.
     */
    public void actionPerformed (ActionEvent e) {
             // Get the button that was clicked
        JButton b = (JButton) e.getSource();
             //  And its label
        String key = b.getText();
        String result =
         calcMachine.handleKeyPress(key,accumulator.getText());
        accumulator.setText(result);
    } // actionPerformed()

    /**
     * main() creates an instance of the interface.
     */
    public static void main( String args[] ) {
        CalculatorGUI calc = new CalculatorGUI("Calculator");
        calc.setSize(400,400);
        calc.setVisible(true);
        calc.addWindowListener(new WindowAdapter() {
```

```
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()

} // CalculatorGUI
```

12. **Challenge:** Design and implement a `Calculator` class to go along with the interface you developed in the previous exercise. It should function the same way as a hand calculator except it only handles integers.

    Answer: The interface (`CalculatorGUI`) of the previous solution uses this solution. It creates an instance of the `Calculator class` and then pass each keystroke to this instance in the `actionPerformed()` method.

    The hardest part of this calculator is designing an algorithm. Because the calculator uses *infix notation*, where the operation occurs between the two operands, as in *operand1 operation opcode2*, you must store the opcode of the operation while the second operand is being input. The problem can be simplified somewhat if you restrict the operations to those of the form: 23 + 65 = 88 – that is, those in which the equal key is pressed after every arithmetic operator. It's a little trickier if you want to allow operations of the form: 23 + 65 - 31 * 2 / 4 ... In this case, whenever an operator key is pressed, you must perform the previous operation. So when the - key is pressed, you perform the + operation, and so on.

    Another challenge is managing the input of digits. When should a typed digit be appended to the current number in the calculator's display, and when should it start a new number. A control variable can be used to keep track of this. See the comments in the code for further details.

```
/*
 * File: Calculator.java
 * Author: Java, Java, Java
 * Description: Implements a multifunction integer calculator.
 * Here's how it works. All calculations are performed using
 * two internal registers, the accumulator, which keeps a running
 * total of ongoing calculations, and the displayRegister, which
 * is set to the value displayed in the external display just
 * before an operation is performed. Inputs to the calculator
 * occur by pressing either digit keys, which signal that an
 * integer operand is being input, and operation keys (+ - * / C =),
 * which indicate that an operation should be performed.
 *
 * Control of the machine is managed by the handleKeyPress()
 * method. Two internal state variables are used to control
 * the machine. (1) The displayState variable keeps track of when
 * a new number is being started (REPLACE) and when incoming digits
 * should be APPENDed to the present value of the display. In APPEND
```

```
 * mode, it appends each digit typed to the external display. In
 * REPLACE mode, it replaces the external display with the digit
 * typed. (2) The opCode variable stores the current operation.
 * What makes control of the calculator somewhat tricky is that it
 * uses INFIX notation, so operations take the form:
 *     Operand1 opCode Operand2
 * The OpCode has to be remembered while Operand2 is being input.
 * The following example shows the state of these control variables
 * after each key press. The vertical lines (|)  show when each
 * operation is done. For the expression 56 + 23 - 5 the calculator
 * does the addition (+) after the - key is pressed. It does the
 * subtraction (-) after the = key is pressed. The equal key resets
 * everything but the accumulator, which always stores the present
 *  total. The accumulator is only reset after a Clear or an error.
 *
 *   KEYPRESS:       5 6  +  |  2  3   -  |  5 =  |
 *   DISPLAY:      0 5 56 56 56 2  23  23 79 5    74
 *   DISPLAYREG:   0      56 0         23 0    5  0
 *   ACCUM:        0      0  56 56     56 79   79 74
 *   OPCODE       -1         1  1        2       -1
 *   DISPLAYSTATE: R A       R   A       R  A     R
 */
import java.lang.*;

public class Calculator {
    public final static int NOOP = -1;  // Operations
    public final static int EQUAL = 0;
    public final static int ADD = 1;
    public final static int SUBTRACT = 2;
    public final static int MULTIPLY = 3;
    public final static int DIVIDE = 4;
    public final static int CLEAR = 5;

    public final static int APPEND = 0; // Display states
    public final static int REPLACE = 1;

    private int accumulator; // The actual accumulator
        // Internal memory for the visible textField
    private int displayRegister;
        // The operation that's waiting for Operand2
    private int opCode;
       // State of the external display (REPLACE or APPEND)
    private int displayState;
       // Set true if divide-by-zero error
    private boolean error;

    /**
     * Calculator() constructor initializes the machine.
     */
    public Calculator () {
```

```
    initialize();
} // constructor

/**
 * initialize() resets the two registers and the opCode
 * and displayState variables. This is invoked initially
 * and then after each clear (C) and after each Error.
 */
private void initialize()  {
      // Reset the machine's registers
    accumulator = 0;
    displayRegister = 0;
       // Replace the display with the next digit typed
    displayState = REPLACE;
    opCode = NOOP; // No current operation
    error = false;
} // initialize()

/**
 * keyToIntCode() converts key characters, such as '+',
 * to an integer code.
 * @return an int representing the key that was pressed
 */
private int keyToIntCode(char keyCh) {
    switch (keyCh) {
        case 'C': return CLEAR;
        case '=': return EQUAL;
        case '+': return ADD;
        case '-': return SUBTRACT;
        case '*': return MULTIPLY;
        case '/': return DIVIDE;
    }
    return -1;      // Error Return
} // keyToIntCode()

/**
 * doCurrentOp() is invoked whenever C, =, +, -, *, or /
 * is pressed.  Except  for the Clear key, it performs
 * an operation, storing the result in the accumulator,
 * which is returned as the method's value. The Clear key
 * is treated as a special case and the machine is
 * reinitialized. For all other keys the opCode field
 * stores the operation that is waiting for its second
 * operand to be input. That operation is performed,
 * using the accumulator and displayRegister. After each
 * operation, the displayState is set to REPLACE, so the
 * next digit typed will start a new integer operand.
 * Note that EQUAL key, resets the opCode  to NOOP, thus
 * beginning another complete INFIX operation. All other
 * keys (+ - / *) set the opCode to their keyCodes, thus
```

```
     * getting the machine ready to complete that operation
     * the next time an operator key is pressed.
     * @param keyCode -- an int representing one of EQUAL, ADD,
     *    SUBTRACT, DIVIDE, MULT, CLEAR
     * @return an int representing the result of the calculation
     */
    private int doCurrentOp(int keyCode) { // do current opCode
        if (keyCode == CLEAR) {
            initialize();
            return accumulator;
        }
        switch (opCode) {
            case NOOP: accumulator = displayRegister; break;
            case ADD : accumulator =
                       accumulator + displayRegister; break;
            case SUBTRACT : accumulator =
                       accumulator - displayRegister; break;
            case MULTIPLY : accumulator =
                        accumulator * displayRegister; break;
            case DIVIDE :
                if (displayRegister == 0)
                    error = true;
                else
                    accumulator = accumulator/displayRegister;
                break;
        } // switch
        if (keyCode == EQUAL)
            opCode = NOOP;  // Reset opCode
        else
            opCode = keyCode; // Set up for next operation
        displayState = REPLACE;
        displayRegister = 0;
        return accumulator;
    } // doCurrentOp

    /**
     * handleKeyPress() handles all key presses. It
     * distinguishes two types of keys, digit keys, which
     * are used to create integer operands  by appending digits,
     * and opCode keys such as '+' and 'C'.
     * @return a String giving the value that should be displayed
     *     by the calculator.
     */
    public String handleKeyPress(String keyStr,
                                 String external_display) {
        String resultStr; // Stores the result
        char keyCh = keyStr.charAt(0); // Convert the key label to a char
        if (Character.isDigit(keyCh)) { // If this was a digit key
            if (displayState == APPEND) //  either append it
                resultStr = external_display + keyCh;
```

```
            else { //  or use it to replace the display
                displayState = APPEND;
                resultStr = keyStr;
            }
        } else { // If not a digit key, it must be an opCode
               // Get display value
            displayRegister = Integer.parseInt(external_display);
               // Perform an operation
            int result = doCurrentOp(keyToIntCode(keyCh));
            if (!error)
                resultStr = Integer.toString(result); // Return result
            else {
                resultStr = "Error";
                initialize();
            }
        }
        return resultStr;  // return the result
    } // handleKeyPress()

} // Calculator Class
```

13. Modify the `Converter` application so that it can convert in either direction: from miles to kilometers, or from kilometers to miles. Use radio buttons in your design to let the user select on or the other alternative.

    Answer: The GUI interface for the solution below is shown in Figure 13.6. The modified `Converter` class below uses a modified `MetricConverter` class which is also listed below.

```
/*
 *  File: Converter.java
 *  Author: Java, Java, Java
 *  Description:  This version of the Converter class
 *  presents an elaborate GUI for interacting with a
 *  MetricConverter class that converts miles to kilometers
 *  and kilometers to miles. It incorporates two radio
 *  buttons that allow the user to control the conversion
 *  from miles to km or vice versa. Note the changes to
 *  the constructor and actionPerformed() methods.
 */

import javax.swing.*;      // Packages used
import java.awt.*;
import java.awt.event.*;

public class Converter extends JFrame
                           implements ActionListener {

    private final static int NBUTTONS = 12;
    private JLabel prompt = new JLabel("Distance: ");
```

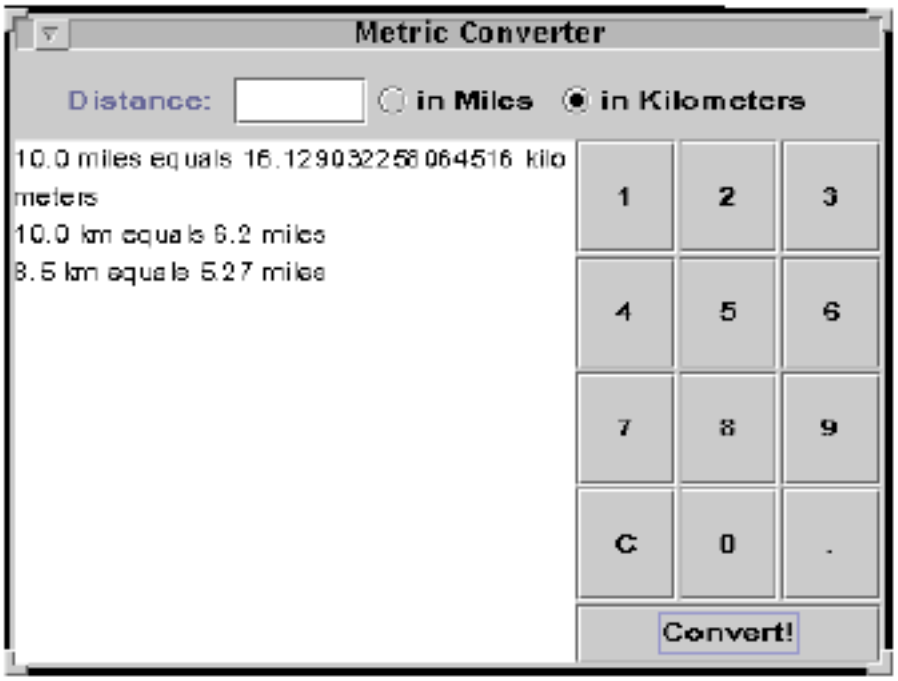


Figure 13.6: Exercise 13: A GUI interface for the `MetricConverter`.

```
private JTextField input = new JTextField(6);
private JTextArea display = new JTextArea(10,20);
private JButton convert = new JButton("Convert!");
private JPanel keypadPanel = new JPanel();
private JRadioButton miUnit =
                    new JRadioButton("in Miles");
private JRadioButton kmUnit =
                    new JRadioButton("in Kilometers");
private ButtonGroup unitsGroup = new ButtonGroup();

private JButton keyPad[];    // An array of buttons
private String label[] =     // An array of button labels
            { "1","2","3",
              "4","5","6",
              "7","8","9",
              "C","0","." };

/**
 *  Converter() constructor sets the layout and adds
 *  components to the top-level JFrame. Note that
 *  components are added to the ContentPane rather to
 *  the JFrame itself. Note how the RadioButtons are
 *  created and added to the ButtonGroup.
 */
public Converter(String title) {
    super(title);
    getContentPane().setLayout(new BorderLayout());
    initKeyPad();

    JPanel inputPanel = new JPanel(); // Input panel
    inputPanel.add(prompt);
    inputPanel.add(input);
        // Configure and add radio buttons
    inputPanel.add(miUnit);
    inputPanel.add(kmUnit);
    unitsGroup.add(miUnit);
    unitsGroup.add(kmUnit);
    miUnit.setSelected(true);
    getContentPane().add(inputPanel,"North");
         // Control panel
    JPanel controlPanel =
        new JPanel(new BorderLayout(0, 0));
    controlPanel.add(keypadPanel, "Center");
    controlPanel.add(convert, "South");
    getContentPane().add(controlPanel, "East");
        // Output display
    getContentPane().add(display,"Center");
    display.setLineWrap(true);
    display.setEditable(false);
        // add listeners
```

```
    convert.addActionListener(this);
    input.addActionListener(this);
} // Converter()

/**
 * initKeyPad() creates an array of JButtons
 * and organizes them into a graphical key pad
 * panel. Note that you must distinguish the
 * JButton array, an internal memory structure,
 * from the key pad panel, a graphical object
 * that appears in the user interface.
 */
public void initKeyPad() {
      // Grid layout
    keypadPanel.setLayout(new GridLayout(4,3,1,1));
      // Create the array itself
    keyPad = new JButton[NBUTTONS];
      // For each button
    for(int k = 0; k < keyPad.length; k++) {
          // Create a labeled button
        keyPad[k] = new JButton(label[k]);
          // and a listener
        keyPad[k].addActionListener(this);
          // and add it to the panel
        keypadPanel.add(keyPad[k]);
    } // for
} // initKeyPad()

/**
 * actionPerformed() handles all action events
 * for the program. It must distinguish whether
 * the user clicked on one of the buttons on the
 * keypad or on the "Convert" or "Input" button.
 * Note how the state of the radio buttons is
 * used to determine whether to convert mi to km
 * or vice versa.
 * @param e -- the ActionEvent which prompted
 *   this method call
 */
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == convert ||
                        e.getSource() == input) {
        if (miUnit.isSelected()) {
            double miles =
                Double.parseDouble(input.getText());
            double km = MetricConverter.milesToKm(miles);
            display.append(miles + " miles equals "
                        + km + " kilometers\n");
            input.setText("");
        } else {
```

```
                double km =
                    Double.parseDouble(input.getText());
                double miles =
                    MetricConverter.kmToMile(km);
                display.append(km+" km equals "+miles+" miles\n");
                input.setText("");
            }
        } else {    // A keypad button was pressed
            JButton b = (JButton)e.getSource();
            if (b.getText().equals("C"))
                input.setText("");
            else
                input.setText( input.getText() + b.getText() );
        }
    } // actionPerformed()

    /**
     * itemStateChanged() handles clicks on the radio buttons
     */
    public void itemStateChanged(ItemEvent e) {

    } // itemStateChanged()

    /**
     *  main() creates an instance of this (Converter) class
     *  and sets the size and visibility of its JFrame.
     *  An anonymous class is used to create an instance of the
     *  WindowListener class, which handles the window close
     *  events for the application.
     */
    public static void main(String args[]) {
        Converter f = new Converter("Metric Converter");
        f.setSize(400, 300);
        f.setVisible(true);
        f.addWindowListener(new WindowAdapter() {
              // Quit the application
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()

} // Converter

/*
 *  File: MetricConverter.java
 *  Author: Java, Java, Java
 *  Description: This class implements the static milesToKm()
 *    method, which converts miles to kilometers.
 */
```

```
public class MetricConverter {

    /**
     *  mileToKm() converts miles to kilometers
     *  @param miles -- number of miles to convert
     *  @return -- a double, the number of kilometers
     */
    public static double milesToKm(double miles) {
        return miles / 0.62;
    } // milesToKm()

    /**
     *  kmToMile() converts kilometers to miles
     *  @param km -- number of kilometers to convert
     *  @return -- a double, the number of miles
     */
    public static double kmToMile(double km) {
        return km * 0.62;
    } // kmToMile()
} // MetricConverter
```

14. Here's a design problem for you. A biologist needs an interactive program that calculates the average of some field data represented as real numbers. Any real number could be a data value, so you can't use a sentinel value, such as 9999, to indicate the end of the input. Design and implement a suitable interface for this problem.

    Answer: The problem is solved by using a GUI which includes two `JButton`s, a `JTextField`, and a `JTextArea`. Data values are typed into the text field with one of the buttons used as a signal that a value should be read and added to a running total. The other button signals that an average should be calculated and displayed in the text area. The `Average` class below creates a GUI interface and also stores a running total and computes an average when instructed to do so.

```
/*
 *  File: Average.java
 *  Author: Java, Java, Java
 *  Description: This class creates a GUI application
 *  and will compute the average of an unlimited
 *  number of data values input by the user. The
 *  data values may be any real number. The average is
 *  computed by clicking a JButton. A dialog box is
 *  displayed if the input string cannot be parsed to
 *  a double value.
 */

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

```
import java.awt.*;

public class Average extends JFrame
                        implements ActionListener {

    private JLabel prompt =
        new JLabel("Input a data value: ");
    private JTextField inField = new JTextField(10);
    private JTextArea  display = new JTextArea();
    private JButton dataButton = new JButton("input data");
    private JButton averageButton = new JButton("Find Average");
    private JPanel controls = new JPanel();
        // Variables used in the calculation
    private double runningTotal = 0.0;
    private int count = 0;

    /**
     * Average() constructor sets up the GUI.
     */
    public Average() {
        getContentPane().setLayout(new BorderLayout());
        controls.add(prompt);
        controls.add(inField);
        dataButton.addActionListener(this);
        averageButton.addActionListener(this);
        getContentPane().add(controls,"North");
        getContentPane().add(display,"Center");
        getContentPane().add(dataButton,"West");
        getContentPane().add(averageButton,"East");
    } // Average

    /**
     * actionPerformed() adds to a sum if the dataButton is
     * clicked.  Finds the average of the data if the
     * averageButton is clicked.
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == dataButton){
            try {
                runningTotal += Double.parseDouble(inField.getText());
                count++;
                inField.setText("");
            } catch(Exception exc) {
                JOptionPane.showMessageDialog(this,
                                "The value must be a real number");
            } // catch()
        } // if dataButton
        if (e.getSource() == averageButton){
            display.append("The average for " + count +
            " data values is "  + runningTotal/count + "\n");
```

```
            } // if averageButton
        } // actionPerformed()

        /**
         *  main() -- Creates an instance of the Average object to compute
         *  the average of the user's input values.
         *  @param args is an array of strings not used here.
         */
        public static void main( String args[] ) {
            Average avg = new Average();
            avg.setSize(600,300);
            avg.setVisible(true);
            avg.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        } // main()
    } // Average
```

15. **Challenge:** A dialog box is a window associated with an application that appears only when needed. Dialog boxes have many uses. An error dialog is used to report an error message. A file dialog is used to help the user search for and open a file. Creating a basic error dialog is very simple in Swing. The `JOptionPane` class has class methods that can be used to create the kind of dialog shown in Text Figure 13.36. Such a dialog box can be created with a single statement:

    ```
    JOptionPane.showMessageDialog(this,
                "Sorry, your number is out of range.");
    ```

    Convert the `Validate` program (Text Figure 6.12) to a GUI interface and use the `JOptionPane` dialog to report errors.

    Answer: The GUI design for the previous solution is also used for this solution. A dialog box is used to inform the user that an input value is not in the required range 0.0 to 100.0. A second dialog box is used to report input strings that do not parse to a double value.

    ```
    /*
    *  File: Validate.java
    *  Author: Java, Java, Java
    *  Description: This class computes the average of an
    *    unlimited number of grades input by the user. This
    *    version modifies the class to create a GUI interface
    *    and displays a dialog box if a value is input
    *    outside the range 0 to 100 or other error.
    */
    ```

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

public class Validate extends JFrame
                        implements ActionListener {


    private JLabel prompt =
        new JLabel("Input values between 0.0 and 100.0: ");
    private JTextField inField = new JTextField(10);
    private JTextArea  display = new JTextArea();
    private JButton dataButton = new JButton("input data");
    private JButton averageButton = new JButton("Find Average");
    private JPanel controls = new JPanel();
        // Variables used in the calculation
    private double runningTotal = 0.0;
    private int count = 0;

    /**
     * Validate() constructor sets up the GUI.
     */
    public Validate() {
        getContentPane().setLayout(new BorderLayout());
        controls.add(prompt);
        controls.add(inField);
        dataButton.addActionListener(this);
        averageButton.addActionListener(this);
        getContentPane().add(controls,"North");
        getContentPane().add(display,"Center");
        getContentPane().add(dataButton,"West");
        getContentPane().add(averageButton,"East");
    } // Validate()

    /**
     * actionPerformed() adds to a sum if the dataButton is
     * clicked.  Finds the average of the data if the
     * averageButton is clicked.
     */
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == dataButton){
            try {
                double value =
                      Double.parseDouble(inField.getText());
                if ((value >= 0.0) && (value <= 100.0)){
                    // Add the value to the running total
                    runningTotal += value;
                    count++;
                    inField.setText("");
                    display.setText(value + " has been input.");
```

```
                } else { // Report out of range error
                  JOptionPane.showMessageDialog(this,
                    "Error: The value must be between 0.0 and 100.0");
                } // else
            } catch(Exception exc) {
                JOptionPane.showMessageDialog(this,
                 "Error: The value must be real number.");
            } // catch()
        } // if dataButton
        if (e.getSource() == averageButton){
            display.setText("The average for " + count +
            " data values is "  + runningTotal/count + "\n");
         } // if averageButton
    } // actionPerformed()

    /**
     *  main() -- Creates an instance of the Validate object
     *  to compute the average of the user's input values.
     *  @param args is an array of strings not used here.
     */
    public static void main( String args[] ) {
        Validate val = new Validate();
        val.setSize(600,300);
        val.setVisible(true);
        val.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()

} // Validate
```

16. **Challenge:** Design and implement a version of the game *Memory*. In this game you are given a two-dimensional grid of boxes that pairs of matching images or strings. The object is to find the matching pairs. When you click on a box, its contents are revealed. You then click on another box. If its contents match the first one, their contents are left visible. If not, the boxes are closed up again. The user should be able to play multiple games without getting the same arrangement every time.

    Answer: The GUI interface for the `MemoryGame` class solution below is shown in Figure 13.7. We use a grid layout of buttons for the game board. This is placed in one control panel located at the center of a border layout. At the north border should be a second panel containing the game's controls. This solution contains a single control – a reset button. A nice refinement would be some kind of display that counts the user's guesses. The game's main control algorithm is put right into the `actionPerformed()` method.

    ```
    /*
    ```

Figure 13.7: Exercise 16: A GUI interface for the `MemoryGame` class.

```
 *  File: MemoryGame.java
 *  Author: Java, Java, Java
 *  Description: This class creates an user interface
 *  a memory game. It displays an NCELLS x NCELLS grid
 *  of buttons, each labeled  initially by "?". Under
 *  the buttons are pairs of letters 'A', 'B', and so on.
 *  The object of the game is to find all the pairs in
 *  the fewest guesses. The user clicks on one button and
 *  then another to show their hidden letters. If they
 *  match, the letters remain showing, otherwise, they
 *  are covered up again when the user clicks on
 *  another button.
 */

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

public class MemoryGame extends JFrame
                     implements ActionListener{
         // Number of hidden letters
    private final static int NCELLS = 16;
         // Should be sqrt of cells
    private final static int NROWS= 4;
        // Panel to hold game board
```

```
 private JPanel gamePanel = new JPanel();
 private JPanel controls = new JPanel();
     // New game button
 private JButton reset = new JButton("New Game");
     // Game memory cells
 private JButton cells[];
 private String alphabet =
                   "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

 private JButton firstGuess, secondGuess;
 private String firstGuessName, secondGuessName;

 /**
  * MemoryGame() constructor sets up the user interface,
  * which consists of two panels, a NROWS x NROWS
  * grid of buttons, hiding the secret letters,
  *  and a control panel containing a reset button.
  */
 public MemoryGame(String title) {
     super(title);
       // Set up the keypad grid layout
     gamePanel.setLayout(new GridLayout (NROWS,NROWS,1,1));
     controls.add(reset);
     reset.addActionListener(this);
// Create the array of button cells
     cells = new JButton [NCELLS];
        // For each array element
     for (int k = 0; k < cells.length; k++) {
         cells[k] = new JButton("?"); //  Create a button
            //  Give the button a name
         cells[k].setName(alphabet.substring(k,k+1));
            //  And a listener for it
         cells[k].addActionListener(this);
            //  And add it to the panel
         gamePanel.add(cells[k]);
     } // for
        // Add components to the frame
     getContentPane().setLayout (new BorderLayout(10, 10));
     getContentPane().add("North", controls);
     getContentPane().add("Center", gamePanel);
     initBoard();
 } // MemoryGame()

 /**
  * initBoard() is called each time the Reset button is
  *  pressed and at the beginning of the game. It scrambles
  *  the secret letters and hides them beneath the buttons.
  *  The secret string is constructed by concatenating a
  *  substring of the alphabet and then shuffling the
  *  letters. The secret string should consist of
```

```
     *  NCELLS/2 pairs of letters.
     */
    private void initBoard() {
        String secret = alphabet.substring(0, NCELLS/2) +
                            alphabet.substring(0, NCELLS/2);
        secret = shuffle(secret);
        firstGuess = null;
        firstGuessName = "?";
        secondGuess = null;
        secondGuessName = "?";
           // For each array slot
        for (int k = 0; k < cells.length; k++) {
            cells[k].setText("?");
            cells[k].setName(secret.substring(k,k+1));
        }
    } // initBoard()

    /**
     * shuffle() rearranges the letters in its parameter by
     * randomly swapping 10 pairs of letters.
     * @param s -- a String to be shuffled
     * @return the shuffled string
     */
    private String shuffle(String s) {
        StringBuffer tempStr = new StringBuffer(s);
        for (int k = 0; k < 10; k++) {
            int m = (int) (Math.random() * NCELLS);
            int n = (int) (Math.random() * NCELLS);
              // Swap two random chars
            char ch = tempStr.charAt(m);
            tempStr.setCharAt(m, tempStr.charAt(n));
            tempStr.setCharAt(n, ch);
        }
        return tempStr.toString();
    } // shuffle()

    /**
     * actionPerformed() handles all the game's action.
     * Each time the user clicks on a button on the memory
     * board, the button's secret letter is displayed. If
     * two successive clicks turn over equal letters, they
     * remain showing. Otherwise they are hidden again.
     */
     public void actionPerformed (ActionEvent e) {
          // Get the button that was clicked
        JButton b = (JButton) e.getSource();
        if (b == reset) {
            // Start a new game
            initBoard();
            return;
```

```
            } // if
              // Ignore illegal moves
            if ( !b.getText().equals("?"))
                return;
               // Reveal the button's name
            b.setText(b.getName());
               // If first guess, store it
            if (firstGuess == null) {
                firstGuess = b;
                firstGuessName =  b.getName();
            } else if (secondGuess == null) {
                  // If second guess, store it
                secondGuess = b;
                secondGuessName =  b.getName();
            } else {   // If third guess, restart
                if (!firstGuessName.equals(secondGuessName)) {
                  // If guesses wrong, Cover up the names
                    firstGuess.setText("?");
                    secondGuess.setText("?");
                } // if guesses not equal
                  // In any case, reset the guesses
                firstGuess = b;
                firstGuessName = b.getName();
                secondGuess = null;
                secondGuessName = "?";
            } // else
        } //  actionPerformed()

        /**
         * main() creates an instance of the interface.
         * @param args - array of strings, not used here
         */
        public static void main( String args[] ) {
            MemoryGame game = new MemoryGame("Memory Game");
            game.setSize(400,400);
            game.setVisible(true);
            game.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        } // main()

    } // MemoryGame
```

17. **Challenge:** Extend the `SimpleTextEditor` program by adding methods to handle the opening, closing, and saving of text files.

    Answer: A modified `SimpleTextEditor` class is listed below. A new item named `saveAsItem` has been added to the `File` menu. New methods named

`openFile()`, `saveFile()`, and `saveAsFile()` are defined and called when the `openItem`, `saveItem`, and `saveAsItem` are selected in the File menu. The code that has been added is labeled with // NEW comments.

```
/*
 *  File: SimpleTextEditor.java
 *  Author: Java, Java, Java
 *  Description:  This application program implements a
 *    menu-based interface for a text editor. It contains
 *    a file menu, an edit menu, and a recent-cuts menu,
 *    a dynamic menu whose options are built during the
 *    editing process.  The recent-cuts menu stores text
 *    that has been cut from the text being edited. Items
 *    from the cuts menu can be pasted into the document.
 *    Support for the Open .. and Save .. menu items have
 *    been added for this version of the application.
 *    New code is marked with comments: // NEW
 */

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import java.io.*; // NEW for use with files

public class SimpleTextEditor extends JFrame
                            implements ActionListener{
       // Create the menu bar
    private JMenuBar mBar = new JMenuBar();
       // Menu references
    private JMenu fileMenu, editMenu, cutsMenu;
      // Edit items
    private JMenuItem cutItem, copyItem, pasteItem,
                          selectItem, recentcutItem;
      // File items
    private JMenuItem quitItem, openItem, saveItem, saveAsItem;
      // Here's where the editing occurs
    private JTextArea display = new JTextArea();
      // Scratch pad for cut/paste
    private String scratchPad = "";
    private Vector recentCuts = new Vector();
    private File theFile = null;  // NEW
      // For use with Open .. and Save ..

    /**
     *  SimpleTextEditor() constructor sets the layout for
     *  the GUI and calls methods to initialize the menus.
     */
    public SimpleTextEditor() {
        super("Simple Text Editor"); // Set the window title
```

```
    this.getContentPane().setLayout(new BorderLayout());
    this.getContentPane().add("Center", display);
    this.getContentPane().add(new JScrollPane(display));
    display.setLineWrap(true);
    this.setJMenuBar(mBar); // Set the menu bar
    initFileMenu();         // Create the menus
    initEditMenu();
} // SimpleTextEditer()

/**
 *  initEditMenu() creates the edit menu and adds its
 *   individual menu items. Note the each menu item
 *   is registered with an ActionListener.
 */
private void initEditMenu() {
    editMenu = new JMenu("Edit"); // Create edit menu
    mBar.add(editMenu);  //  and add it to menu bar
    cutItem = new JMenuItem ("Cut");  // Cut item
    cutItem.addActionListener(this);
    editMenu.add(cutItem);
    copyItem = new JMenuItem("Copy"); // Copy item
    copyItem.addActionListener(this);
    editMenu.add(copyItem);
    pasteItem = new JMenuItem("Paste"); // Paste item
    pasteItem.addActionListener(this);
    editMenu.add(pasteItem);
    editMenu.addSeparator();
    selectItem = new JMenuItem("Select All");
    selectItem.addActionListener(this); // Select item
    editMenu.add(selectItem);
    editMenu.addSeparator();
    cutsMenu = new JMenu("Recent Cuts");
    editMenu.add(cutsMenu);  // Recent cuts submenu
} // initEditMenu()

/**
 *  initFileMenu() creates the file menu and adds its
 *   individual menu items. Note the each menu item
 *   is registered with an ActionListener.
 */
private void initFileMenu() {
      // Create the file menu
    fileMenu = new JMenu("File");
    mBar.add(fileMenu); // add it to the menu bar
    openItem = new JMenuItem("Open .."); // Open item
    openItem.addActionListener( this );
    openItem.setEnabled(true); // NEW
    fileMenu.add(openItem);
    saveItem = new JMenuItem("Save"); // Save item
    saveItem.addActionListener(this);
```

```
    saveItem.setEnabled(true); // NEW
    fileMenu.add(saveItem);
      // NEW Sava As item
    saveAsItem = new JMenuItem("Save As ..");
    saveAsItem.addActionListener(this);
    saveAsItem.setEnabled(true);
    fileMenu.add(saveAsItem);
    fileMenu.addSeparator();   // Logical separator
    quitItem = new JMenuItem("Quit"); // Quit item
    quitItem.addActionListener(this);
    fileMenu.add(quitItem);
} // initFileMenu()

/**
 *  actionPerformed() handles the user's menu selections.
 *  @param e -- the ActionEvent that led to this method call
 */
public void actionPerformed(ActionEvent e) {
       // Get the selected menu item
    JMenuItem m  = (JMenuItem)e.getSource();
    if ( m == quitItem ) {  // Quit
        dispose();
    } else if ( m == openItem ) { // Open -- NEW
        openFile();
    } else if ( m == saveItem ) { // Save -- NEW
        saveFile();
    } else if ( m == saveAsItem ) { // SaveAs -- NEW
        saveAsFile();
    } else if (m == cutItem) {
         // Copy selected text to the scratchpad
        scratchPad = display.getSelectedText();
          //delete from start of selection to end
        display.replaceRange("",
            display.getSelectionStart(),
            display.getSelectionEnd());
          // Add the cut text to the cuts menu
        addRecentCut(scratchPad);
    } else if (m == copyItem) {
          // Copy the selected text to the scratchpad
        scratchPad = display.getSelectedText();
    } else if (m == pasteItem) {
          // Paste scratchpad to document at caret position
        display.insert(scratchPad, display.getCaretPosition());
    } else if ( m == selectItem ) {
          // Select the entire document
        display.selectAll();
    } else {
            // Default case is the cutsMenu
        JMenuItem item = (JMenuItem)e.getSource();
            // Put the cut back in the scratchpad
```

```
            scratchPad = item.getActionCommand();
        }
    } // actionPerformed()

    /**
     * addRecentCut() adds its parameter to the recent cut menu,
     * a menu which grows dynamically as the user cuts text
     * from a document  Recent cuts are stored in a vector.
     * When inserting a cut into the menu, the menu is first cleared
     * with removeAll(), and then the recent cuts are inserted in
     * last-in-first-out order. Each newly inserted cut is assigned
     *  an ActionListener.
     *  @param cut -- the text to be added as the menu item
     */
    private void addRecentCut(String cut) {
        recentCuts.insertElementAt(cut,0);
        cutsMenu.removeAll();
        for (int k = 0; k < recentCuts.size(); k++) {
            JMenuItem item =
                new JMenuItem((String)recentCuts.elementAt(k));
            cutsMenu.add( item );
            item.addActionListener(this);
        } // for
    } // addRecentCut()


    /** NEW ** NEW ** NEW **
     * openFile() opens a JFileChooser showOpenDialog() dialog.
     * If a text file is chosen, the text from the file is read
     * into the display JTextArea.
     */
    private void openFile() {
JFileChooser chooser = new JFileChooser();
int result = chooser.showOpenDialog(this);
if (result == JFileChooser.APPROVE_OPTION) {
    theFile = chooser.getSelectedFile();
    try{
                BufferedReader inStream =
                  new BufferedReader(new FileReader(theFile));
          // Read data from file
                String line = inStream.readLine();
                display.setText("");
                while (line != null){
                    display.append(line + "\n");
                    line = inStream.readLine();
                } // while
                   // close the stream
                inStream.close();
            } catch(FileNotFoundException e) {
        System.out.println("ERROR:"+e.getMessage()+"\n");
```

```
                e.printStackTrace();
            } catch(IOException e) {
        System.out.println("ERROR:"+e.getMessage()+"\n");
                e.printStackTrace();
            } // catch()
} // if
     } // openFile()

    /** NEW ** NEW ** NEW **
     * saveFile()  the text from display is written
     * into the current file.
     */
    private void saveFile() {
if (theFile != null) {
    try{
                FileWriter outStream =
                             new FileWriter(theFile);
          // Write data to file
                outStream.write(display.getText());
                  // close the stream
                outStream.close();
            } catch(IOException e) {
        System.out.println("ERROR:"+e.getMessage()+"\n");
                e.printStackTrace();
            } // catch()
} else { // else theFile == null
            saveAsFile();
        } // else
    } // saveFile()

    /** NEW ** NEW ** NEW **
     * saveAsFile() opens a JFileChooser showSaveDialog() dialog.
     * If a text file is chosen, the text from display is written
     * into the chosen file.
     */
    private void saveAsFile() {
JFileChooser chooser = new JFileChooser();
int result = chooser.showSaveDialog(this);
if (result == JFileChooser.APPROVE_OPTION) {
    theFile = chooser.getSelectedFile();
    try{
                FileWriter outStream =
                             new FileWriter(theFile);
          // Write data to file
                outStream.write(display.getText());
                  // close the stream
                outStream.close();
            } catch(IOException e) {
        System.out.println("ERROR:"+e.getMessage()+"\n");
                e.printStackTrace();
```

```
            } // catch()
    } // if
        } // saveAsFile()

        /**
         *  main() creates an instance of the SimpleTextEditor
         */
        public static void main(String args[]) {
            SimpleTextEditor f = new SimpleTextEditor();
            f.setSize(300, 200);
            f.setVisible(true);
            f.addWindowListener(new WindowAdapter() { // Quit application
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        } // main()
    } // SimpleTextEditor
```

Chapter 14

Threads and Concurrent Programming

1. Explain the difference between the following pairs of terms.

   (a) *Blocked* and *ready*.

   Answer: A thread is *blocked* and prevented from running when it is waiting for some kind of I/O operation to complete. A thread in in the *ready* state is waiting for the CPU to become available.

   (b) *Priority* and *round-robin* scheduling.

   Answer: *Priority scheduling* schedules threads according to their relative priority, with higher priority threads getting the CPU first. In *round-robin scheduling* threads of the same priority take turns using the CPU with each thread getting a fixed *quantum* of time.

   (c) *Producer* and *consumer*.

   Answer: *Producer* and *consumer* threads work cooperatively with the producer thread producing (creating) some resource that is consumed (used) by the consumer thread.

   (d) *Monitor* and *lock*.

   Answer: A *monitor* is a mechanism that guarantees mutually exclusive access to a synchronized method among a group of cooperating threads. Whenever a thread calls a synchronized method, the object containing that method is *locked*, preventing other threads from accessing any of the object's synchronized methods. When the first thread finishes the synchronized method, the lock is released.

   (e) *Concurrent* and *time slicing*.

   Answer: *Concurrent* execution implies that two or more threads are sharing the same CPU over a certain period of time. One way to implement concurrent execution is to give each thread a small slice of the CPU's time, a process known as *time slicing*.

(f) *Mutual exclusion* and *critical section*.

Answer: *Mutual exclusion* means that among a group of threads, one and only one thread should have access to a certain resource (e.g., a memory variable or method) at a time. A *critical section* is a section of code that should be executed in a mutually exclusive manner by a group of threads.

(g) *Busy* and *nonbusy* waiting.

Answer: *Busy* waiting occurs when the CPU sets in a trivial loop for a fixed amount of time as a way of delaying a thread's progress. In *nonbusy* waiting, a thread gives up the CPU to another thread while waiting for some event or condition to occur.

2. Fill in the blank:

(a) _____ happens when a CPU's time is divided among several different threads. **Answer: Time slicing**

(b) A method that should not be interrupted during its execution is known as a _____. **Answer: synchronized method**

(c) The scheduling algorithm in which each thread gets an equal portion of the CPU's time is known as _____. **Answer: round-robin scheduling**

(d) The scheduling algorithm in which some threads can preempt other threads is known as _____. **Answer: priority scheduling**

(e) A _____ is a mechanism that enforces mutually exclusive access to a synchronized method. **Answer: monitor**

(f) A thread that performs an I/O operation may be forced into the _____ state until the operation is completed. **Answer: blocked**

3. Describe the concept of *time slicing* as it applies to CPU scheduling.

Answer: In *time slicing* each thread is given a fixed *quantum* of CPU time before being preempted. The CPU is then given to another thread for a fixed amount of time. And so on.

4. What's the difference in the way concurrent threads would be implemented on a computer with several processors and on computer with a single processor.

Answer: On a computer with several processors, each thread could, in principle, be given its own processor, assuming there are enough processors to go around. In that case the threads could execute at the same time, that is, concurrently. On a computer with a single processor, the threads have to share the processor. So they cannot operate concurrently. No two threads could be running at exactly the same instant of time.

5. Why are threads put into the *blocked* state when they perform an I/O operation?

Answer: It would be inefficient for a thread to occupy the CPU while waiting for an I/O operation to complete. Because I/O operations usually involve some kind of mechanical process, they are much slower than CPU operations, which are entirely electronic.

6. What's the difference between a thread in the sleep state and a thread in the ready state?

   Answer: In the sleep state a thread has voluntarily given up the CPU for a fixed quantum of time. It cannot be ready to run again until its sleep time has expired. A thread in the ready state is just waiting to get its turn on the CPU.

7. *Deadlock* is a situation that occurs when one thread is holding a resource that another thread is waiting for, while the other thread is holding a resource that the first thread is waiting for. Describe how deadlock can occur at a four-way intersection, with cars entering the intersection from each branch. How can it be avoided?

   Answer: Imagine that one car from each of the four directions approaches the intersection at the same time. The result is that each car should defer to the car on the car's right; the east car is blocking the south car from proceeding, the south car is blocking the west car, and so on. This is *deadlock*. Each car is holding a resource (a piece of the road) while waiting for another car to give up a resource (another piece of the road). No car can proceed until the deadlock is broken. Stop lights are one way to avoid deadlock in this situation. A second way to avoid deadlock is to have stop signs for the roads from the east and west and yield signs for the roads from the north and south.

8. *Starvation* can occur is one thread is repeatedly preempted by other threads. Describe how starvation can occur at a four-way intersection and how it can be avoided.

   Answer: Suppose that the north car must always yield to oncoming traffic from both the west and east. This would be the case if north has a stop sign but east and west do not. If traffic is very heavy, the north car could be prevented from proceeding. This would cause *starvation* in the sense that the north car would never get the desired resource (the road). To avoid starvation, cars in the oncoming traffic should yield to the side street occasionally, either because of a stop light or because of courtesy.

9. Use the `Runnable` interface to define a thread that repeatedly generates random numbers in the interval 2 through 12.

   Answer: The following `RandomNumbers` class is tested in it's `main()` method.

```
/*
 * File: RandomNumbers.java
 * Author: Java, Java, Java
 * Description: This class implements the Runnable
 * interface, which consists of the run() method.
 * The run() method prints an infinite sequence
 * of random integers between 2 and 12.
 */

public class RandomNumbers implements Runnable {
```

```
    /**
     *  run() is the thread's primary method. It generates a
     *  random number between 2 and 12
     */
    public void run() {
        while (true)     // Infinite loop
            System.out.println(2 + (int)(Math.random() * 11));
    } // run()

    /**
     * main() -- Creates a Thread instance with a constructor
     * that has a RandomNumbers instance as an argument
     * and starts the thread.
     * @param args is an array of strings not used here.
     */

    public static void main(String args[]) {
        Thread numbers = new Thread(new RandomNumbers());
        numbers.start(); // Create and start each thread
    } // main()

} // RandomNumbers
```

10. Create a version of the `Bakery` program that uses two clerks to serve customers.

    Answer: The only change required to implement a two-clerk version of the `Bakery` program is to create a second `Clerk` thread in the main program, and then start both threads. The synchronization of the two clerks will happen automatically, as a result of the design of the `TakeANumber` class:

```
public class Bakery {

  public static void main(String args[]) {
    System.out.println("Starting clerk and customer threads");
    TakeANumber numberGadget = new TakeANumber();
    Clerk clerk = new Clerk(numberGadget);
    Clerk clerk2 = new Clerk(numberGadget);
    clerk.start();
    clerk2.start();
    for (int k = 0; k < 5; k++) {
      Customer customer = new Customer(numberGadget);
      customer.start();
    } // for
  } // main()

} // Bakery
```

    However, if just this change is made it will be impossible to tell which clerk is doing the serving. To remedy this we want to give each clerk a unique ID

number. We can use a `static` variable to serve as the source of unique IDs and assign each new clerk object an ID in the constructor method. Then when the clerk calls the `nextCustomer()` method, it passes its ID number.

```
/*
 * File: Clerk.java
 * Author: Java, Java, Java
 * Description: This class defines the clerk
 * thread of the bakery simulation. The clerk
 * has a reference to  the TakeANumber gadget.
 * The clerk "serves" the next customer by
 * repeatedly calling takeANumber.nextCustomer().
 * To simulate the randomness involved in serving
 * times, the clerk sleeps for a random interval
 * on each iteration of its run loop.
 */

public class Clerk extends Thread {
    private static int id = 0;
    private int myId;

    private TakeANumber takeANumber;

    /**
     * Clerk() constructor gives the clerk a
     * reference to the TakeANumber gadget.
     */
    public Clerk(TakeANumber gadget) {
      takeANumber = gadget;
      myId = id++;
    }

    /**
     * run() is the main algorithm for the clerk
     * thread. Note that it runs in an infinite
     * loop.  Note that in this version the clerk
     * no longer checks if there are customers
     * waiting before calling nextCustomer(). This
     * check has been moved into the TakeANumber
     * object itself.
     */
    public void run() {
      while (true) {
        try {
          sleep((int)(Math.random() * 1000));
          takeANumber.nextCustomer(myId);
        } catch (InterruptedException e) {
          System.out.println("Exception: "+e.getMessage());
        } // catch()
      } // while
```

```
    } // run()
} // Clerk
```

Finally, the `TakeANumber.nextCustomer()` method must be modified, to include the `clerkId` as a parameter:

```
  public synchronized int nextCustomer(int clerkId) {
    try {
      while (next <= serving)  {
        System.out.println("  Clerk "+clerkId+" waiting ");
        wait();
      } // while
    } catch(InterruptedException e) {
      System.out.println("Exception " + e.getMessage() );
    } finally {
      ++serving;
      System.out.println("  Clerk " + clerkId +
                         " serving ticket " + serving );
      return serving;
    } finally
  } // nextCustomer()
```

Given these changes, the program should produce something like the following output:

```
Starting clerk and customer threads
Starting clerk and customer threads
Customer 10001 takes ticket 1
Customer 10002 takes ticket 2
  Clerk 1 serving ticket 1
  Clerk 0 serving ticket 2
Customer 10005 takes ticket 3
  Clerk 1 serving ticket 3
  Clerk 1 waiting
Customer 10003 takes ticket 4
  Clerk 1 serving ticket 4
  Clerk 0 waiting
Customer 10004 takes ticket 5
  Clerk 0 serving ticket 5
  Clerk 0 waiting
  Clerk 1 waiting
```

11. Modify the `Numbers` program so that the user can interactively create `NumberThread`s and assign them a priority. Modify the `NumberThread`s so that they print their numbers indefinitely (rather than for a fixed number of iterations). Then experiment with the system by observing the effect of introducing threads with the same, lower, or higher priority. How do the threads behave when they all have the same priority? What happens when you introduce a higher-priority thread

into the mix? What happens when you introduce a lower-priority thread into the mix?

Answer: The only change required in the `NumberPrinter` class is to change the for loop in `run()` to an infinite loop. In the `Numbers` class, a `BufferedReader` is used to perform input and the user is prompted for the priority of each of 5 threads. Each thread's priority is set in `main()` before starting the thread.

```
/*
 * File: NumberPrinter.java
 * Author: Java, Java, Java
 * Description: This class implements the Runnable
 *  interface, which consists of the run() method.
 *  Implementing the Runnable interface is a way to
 *  turn an existing class into a separate thread.
 */

public class NumberPrinter implements Runnable {
    int num;

    /**
     *  NumberPrinter() constructor assigns its
     *  parameter as the thread's number.
     *  @param n -- the number assigned to this object
     */
    public NumberPrinter(int n) {
        num = n;
    }

    /**
     *  run() is the thread's primary method. It prints its
     *  number indefinitely. This version has each thread sleep
     *  for a random interval. This will cause the threads
     *  to run in an arbitrary order.
     */
    public void run() {
        for (; ; ) { // Infinite Loop
            try {
                Thread.sleep((int)(Math.random() * 1000));
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
            System.out.print(num);
        } // for
    } // run()
} // NumberPrinter

/*
 * File: Numbers.java
 * Author: Java, Java, Java
```

```
 * Description: This class provides an interface to the
 * NumberPrinter class. It allows the user to create
 * NumberPrinter threads and and assign each one a priority.
 * It starts each one, each of which repeatedly prints its
 * number. This allows you to see the order and duration of
 * each thread's execution.
 *
 * In this version, because NumberPrinter's are objects that
 * implement the Runnable interface, we create them with
 * new Thread(new NumberPrinter(k)). Note also that an array
 * is used to store the individual threads.
 */

import java.io.*;

public class Numbers {

    public static void main(String args[])
                                  throws IOException {
          // An array of 5 threads
        Thread number[] = new Thread[5];
        BufferedReader input = new BufferedReader
                    (new InputStreamReader(System.in));
        System.out.println("The min and max priorities are " +
          Thread.MIN_PRIORITY + " .... " + Thread.MAX_PRIORITY);
        for (int k = 0; k < 5; k++) {
            System.out.print("Input the priority of thread " +
                                                 (k+1) + " :");
            int priority = Integer.parseInt(input.readLine());
              // Create each thread
            number[k] = new Thread(new NumberPrinter(k + 1));
            number[k].setPriority(priority);
        } // for
          // Start each thread
        for (int k = 0; k < 5; k++)
            number[k].start();
        System.out.println();
    } // main()
} // Numbers
```

12. Create a bouncing ball simulation in which a single ball (thread) bounces up and down in a vertical line. The ball should bounce off the bottom and top of the enclosing frame.

    Answer: The `Ball` class is a subclass of `Thread`. It's bouncing is effected by repeatedly drawing, moving and erasing an oval on the screen. In order to insure that the ball remains within the applet's visible window, the `Ball` is given a reference to the applet, from which it can get the applet's dimensions. The GUI interface for the `BounceApplet` class solution below is shown in Figure 14.1.

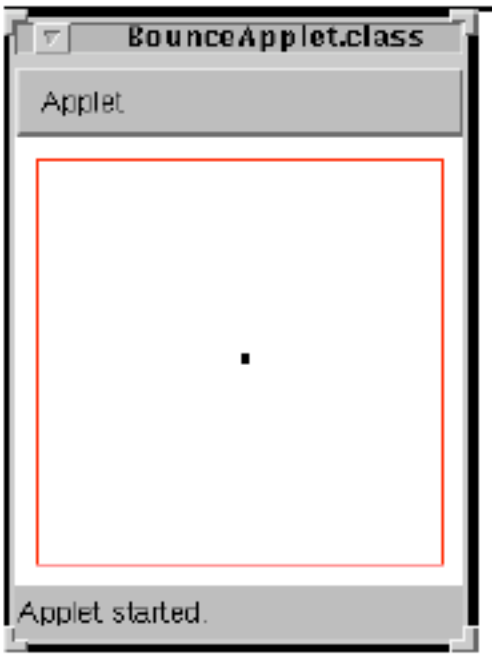


Figure 14.1: Exercise 12: A GUI interface for the bouncing ball applet.

```
/*
 * File: Ball.java
 * Author: Java, Java, Java
 * Description: This class defines a ball that bounces
 * up and  down within an applet. The "bouncing" is
 * effected by repeatedly drawing, and erasing and moving
 * the ball in the run() method.
 */

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

public class Ball extends Thread implements Runnable {
         // Diameter of the ball
    public static final int SIZE = 5;
        // Number of pixels to move the ball
    private static final int DY = 5;
        // A 10 pixel border around drawing area
    private static final int BORDER = 10;
        // Reference to the applet
    private JApplet applet;
        // Boundaries
    private int topWall, bottomWall;
        // Current location of the ball
    private Point location;
         // Ball's x- and y-direction (1 or -1)
    private int directionX = 1, directionY = 1;

    /**
     * Ball() constructor sets a pointer to the
     * applet and initializes the ball's location
     * at the center of the bouncing region
     */
    public Ball(JApplet app) {
        applet = app;
          // Define bouncing region
        Dimension gameArea = applet.getSize();
        topWall = BORDER + 1;
        bottomWall = gameArea.height - BORDER - SIZE - 1;
        location =    // Set initial location
          new Point(gameArea.width/2, gameArea.height/2);
        applet.getGraphics().setColor(Color.black);
    } // Ball()

    /**
     * draw() draws the ball
     */
    public void draw() {
        Graphics g = applet.getGraphics();
```

```
    g.setColor(Color.black);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // draw()

/**
 * erase() erases the ball
 */
public void erase() {
    Graphics g = applet.getGraphics();
    g.setColor(Color.white);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // erase()

/**
 * move() changes the ball's vertical location
 * (y-coordinate) by DY pixels. It then checks
 * if the ball has reached a boundary and if so
 * reverses direction
 */
 public void move() {
      // Calculate a new location
     location.y = location.y + DY * directionY;
      // If ricochet
     if (location.y > bottomWall) {
           //  reverse direction
         directionY = -directionY;
         location.y =  bottomWall;
     } // if
     if (location.y < topWall) {
           // Reverse direction
         directionY = -directionY;
         location.y = topWall;
     } // if
} // move()

/**
 * run() repeatedly draws, erases and moves
 * the ball
 */
public void run() {
    while (true) {
        draw();  // Draw
        try {
            sleep(100);
        }
        catch (InterruptedException e) {}
        erase(); // Erase
        move();  // Move
   } // while
} // run()
```

```
} // Ball

/*
 * File: BounceApplet.java
 * Author: Java, Java, Java
 * BounceApplet displays a bouncing ball within a
 * red rectangle.
*/

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

public class BounceApplet extends JApplet {

    private Thread ball;

    public void paint (Graphics g ) {
        g.setColor(Color.white);
        g.fillRect(10,10,180,180);
        g.setColor(Color.red);
           // Draw the bouncing region
        g.drawRect(10,10,180,180);
          // Create and start the ball
    } // paint()

    public void start(){ // The JApplet start() method
        repaint();
           // Create and start the ball
        ball = new Thread(new Ball(this));
        ball.start();
    } // start()

} // BounceApplet
```

13. Modify the simulation in the previous exercise so that more than one ball can be introduced. Allow the user to introduce new balls into the simulation by pressing the space bar or clicking the mouse.

    Answer: This version must implement the `KeyListener` interface so that each time the spacebar is pressed, the applet can create and start a new applet thread. Only the `KeyListener.keyTyped()` method must be implemented. The applet must also `requestFocus()` in order to be sent key events.

    In the `Ball` class, the only change we require is that we want to start each ball at a different, random location. We do this by assigning a random value to the ball's x-coordinate. That way each ball will bounce in a different vertical plane. The GUI interface for the new `BounceApplet` class solution below is shown in Figure 14.2.

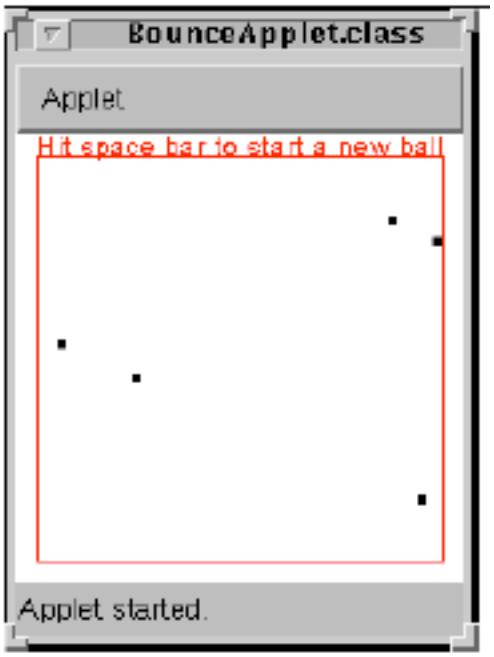


Figure 14.2: Exercise 13: In this version of the bouncing ball applet the user can add new balls to the mix by pressing the space bar.

```
/*
 * File: Ball.java
 * Author: Java, Java, Java
 * Description: This class defines a ball that bounces
 * up and down within an applet. The "bouncing" is
 * effected by repeatedly drawing, and erasing and moving the
 * ball in the run() method.
 *
 * In this version of Ball, instead of starting every Ball
 * in the center of the bounce area, each ball is given
 * a random x-coordinate. So each bounces in its own vertical
 * space.
 */

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

public class Ball extends Thread implements Runnable {
      // Diameter of the ball
    public static final int SIZE = 5;
      // Number of pixels to move the ball
    private static final int DY = 5;
      // A 10 pixel border around drawing area
    private static final int BORDER = 10;

    private JApplet applet; // Reference to the applet
    private int topWall, bottomWall; // Boundaries

    private Point location; // Current location of the ball
      // Ball's x- and y-direction (1 or -1)
    private int directionX = 1, directionY = 1;

    /**
     * Ball() constructor sets a pointer to the applet and
     * initializes the ball's location
     * at the center of the bouncing region
     */
    public Ball(JApplet app) {
        applet = app;
          // Define bouncing region
        Dimension gameArea = applet.getSize();
        topWall = BORDER + 1;
        bottomWall = gameArea.height - BORDER - SIZE - 1;
          // Pick a random xLoc
        int xLoc = BORDER + 1 + (int)(Math.random() *
                 (gameArea.width - 2 * BORDER - 2 - SIZE));
          // Set initial location
        location = new Point(xLoc, gameArea.height/2);
    } // Ball()
```

```
/**
 * draw() draws the ball
 */
public void draw() {
    Graphics g = applet.getGraphics();
    g.setColor(Color.black);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // draw()

/**
 * erase() erases the ball
 */
public void erase() {
    Graphics g = applet.getGraphics();
    g.setColor(Color.white);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // erase()

/**
 * move() changes the ball's vertical location
 * (y-coordinate) by DY pixels. It then checks if the ball
 * has reached a boundary and if so reverses direction
 */
 public void move() {
       // Calculate a new location
     location.y = location.y + DY * directionY;
     if (location.y > bottomWall) { // If ricochet
         directionY = -directionY; // reverse direction
         location.y =  bottomWall;
     } // if
     if (location.y < topWall) {
         directionY = -directionY; // Reverse direction
         location.y = topWall;
     } // if
} // move()

/**
 * run() repeatedly draws, erases and moves the ball
 */
public void run() {
    while (true) {
        draw();      // Draw
        try {
            sleep(100);
        }
        catch (InterruptedException e) {}
        erase();   // Erase
        move();  // Move
   } // while
```

```
    } // run()

} // Ball
/*
 * File: BounceApplet.java
 * Author: Java, Java, Java
 * Description:  This version allows the user to introduce
 * new balls by clicking the spacebar. It implements the
 * KeyListener interface.
 */

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

public class BounceApplet extends JApplet
                                implements KeyListener {
    private Thread  ball;

    public void init() {
        setSize(200,200);
        setBackground(Color.white);
        requestFocus(); // Required to receive key events
        addKeyListener(this);
    } // init()

    public void paint (Graphics g ) {
        g.setColor(Color.white);
        g.fillRect(10,10,180,180);
        g.setColor(Color.red); // Draw the bouncing region
        g.drawRect(10,10,180,180);
        g.drawString("Hit space bar to start a new ball",10,10);
         // Create and start the ball
    } // paint()

    public void start(){ // The JApplet start() method
        repaint();
           // Create and start the ball
        ball = new Thread(new Ball(this));
        ball.start();
    } // start()

    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar() == ' ') { // If spacebar pressed
            ball = new Thread(new Ball(this));
            ball.start();
        }
    } // keyType()
      // Unused part of KeyListener interface
    public void keyReleased( KeyEvent e) {}
```

```
    public void keyPressed( KeyEvent e) {}  // Unused
} // BounceApplet
```

14. Modify your solution to the previous problem by having the balls bounce off the wall at a random angle.

    Answer: To introduce randomness we calculate a random value for `deltaY`, the amount by which the y-coordinate changes, each time the ball comes in contact with the left or right wall. The causes the ball to move at an angle, instead of horizontally. See the `Ball.move()` method. The GUI interface for the new `BounceApplet` class solution below is shown in Figure 14.3.

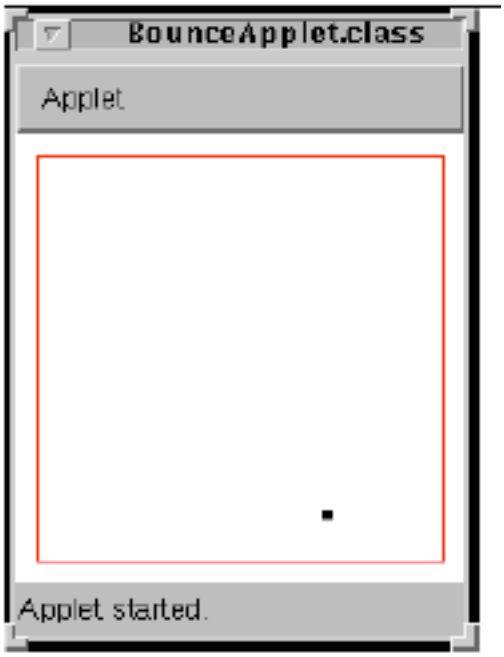


Figure 14.3: Exercise 14:In this version of the bouncing ball applet the ball bounces at random angles off of the left and right walls.

```
/*
 * File: Ball.java
 * Author: Java, Java, Java
 * Description: This class defines a ball that bounces
 * up and down within an applet. The "bouncing" is
 * effected by repeatedly drawing, and erasing and moving the
 * ball in the run() method.
 *
```

```
 * In this version of Ball, instead of starting every Ball
 * in the center of the bounce area, each ball is given
 * a random x-coordinate. So each bounces in its own vertical
 * space. Also, the balls bounce at random angles off of the
 * four walls. The randomness is introduced by changing the
 * y-coordinate by a random amount between -4 and +4 each
 * time the ball bounces off either the right or left wall.
 * This changes the angle
 *  of the ball's motion.
 */

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

public class Ball extends Thread implements Runnable {
       // Diameter of the ball
    public static final int SIZE = 5;
       // Number of pixels to move the ball
    private static final int DX = 5;
       // Number of pixels to move the ball
    private static final int DY = 5;
       // A 10 pixel border around drawing area
    private static final int BORDER = 10;
       // Used by randomizer: Vertical changes goes from
    private static final int MINDY = -4;
       //   -4 to +4, giving 9 discrete values
    private static final int MAXDY = 9;

    private JApplet applet; // Reference to the applet
    private int topWall, bottomWall; // Boundaries
    private int leftWall, rightWall; // Boundaries

    private Point location; // Current location of ball
     // Ball's x- and y-direction (1 or -1)
    private int directionX = 1, directionY = 1;
    private int deltaY; // Change in the Y coordinate

    /**
     * Ball() constructor sets a pointer to the applet and
     * initializes the ball's location
     *  at the center of the bouncing region
     */
    public Ball(JApplet app) {
        applet = app;
          // Define bouncing region
        Dimension gameArea = applet.getSize();
          // And location of walls
        rightWall = gameArea.width - BORDER - SIZE - 1;
      leftWall = BORDER + 1;
```

```
    topWall = BORDER + 1;
    bottomWall = gameArea.height - BORDER - SIZE - 1;
     // Pick a random xLoc
    int xLoc = BORDER + 1 + (int)(Math.random() *
             (gameArea.width - 2 * BORDER - 2 - SIZE));
      // Set initial location
    location = new Point(xLoc, gameArea.height/2);
    applet.getGraphics().setColor(Color.black);
} // Ball()

/**
 * draw() draws the ball
 */
public void draw() {
    Graphics g = applet.getGraphics();
    g.setColor(Color.black);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // draw()

/**
 * erase() erases the ball
 */
public void erase() {
    Graphics g = applet.getGraphics();
    g.setColor(Color.white);
    g.fillOval(location.x, location.y, SIZE, SIZE);
} // erase()

/**
 * move() changes the ball's vertical location
 * (y-coordinate) by DY pixels. It then checks if the ball
 * has reached a boundary and if so reverses direction
 */
 public void move() {
        // Calculate a new location
     location.x = location.x + DX * directionX;
     location.y = location.y + deltaY * directionY;

     if (location.y > bottomWall) { // If ricochet
         directionY = -directionY; // reverse direction
         location.y =  bottomWall;
     }
     if (location.y < topWall) {
         directionY = -directionY; // Reverse direction
         location.y = topWall;
     }
     if (location.x > rightWall )  {
         directionX = -directionX;
            // CHANGE VERTICAL DIRECTION
         deltaY = MINDY + (int) (Math.random() * MAXDY);
```

```
            location.x =  rightWall;
        }

        if ( location.x < leftWall ) {
            directionX = -directionX;
              // CHANGE VERTICAL DIRECTION
            deltaY = MINDY + (int) (Math.random() * MAXDY);
            location.x = leftWall;
        }
    } // move()

    /**
     * run() repeatedly draws, erases and moves the ball
     */
    public void run() {
        while (true) {
            draw(); // Draw
            try {
                sleep(100);
            }
            catch (InterruptedException e) {}
            erase(); // Erase
            move(); // Move
       } // while
    } // run()
} // Ball

/*
 * File: BounceApplet.java
 * Author: Java, Java, Java
 * Description:  This version allows the user to
 * introduce new balls by clicking the spacebar. It
 * implements the KeyListener interface.
 */

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

public class BounceApplet extends JApplet
                              implements KeyListener {
    private Thread  ball;

    /**
     * init() gets the focus for key events and registers
     * the applet as a KeyListener.
     */
    public void init() {
        setSize(200,200);
        setBackground(Color.white);
```

```
        requestFocus(); // Required to receive key events
        addKeyListener(this);
    } // init()

    /**
     * paint() paints a red rectangle for the bouncing
     * region and creates the first bouncing ball.
     */
    public void paint (Graphics g ) {
        g.setColor(Color.white);
        g.fillRect(10,10,180,180);
        g.setColor(Color.red); // Draw the bouncing region
        g.drawRect(10,10,180,180);
        g.drawString("Hit space bar to start a new ball",10,10);
    } // paint()

    /**
     * start() The JApplet start() method.
     * Calls repaint() and creates and starts a ball.
     */
    public void start(){
        repaint();
           // Create and start the ball
        ball = new Thread(new Ball(this));
        ball.start();
    } // start()

    /**
     * keyTyped() is invoked every time a KeyEvent occurs.
     * The getKeyChar() method gets the value of the key
     * that was pressed. Whenever the spacebar is pressed, a
     * new bouncing ball is introduced into the applet.
     */
    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar() == ' ') { // If spacebar pressed
            ball = new Thread(new Ball(this));
            ball.start();
        }
    } // keyType()

     // Unused part of KeyListener interface
    public void keyReleased( KeyEvent e) {}
    public void keyPressed( KeyEvent e) {}  // Unused

} // BounceApplet
```

15. **Challenge:** One type of producer/consumer problem is the *reader/writer* problem. Create a subclass of `JTextField` that can be shared by threads, one of which writes a random number to the text field, and the other of which reads the value in the text field. Coordinate the two threads so that the overall effect of

program will be that it will print the values from 0 to 100 in the proper order. In other words, the reader thread shouldn't read a value from the text field until there's a value to be read. The writer thread shouldn't write a value to the text field until the reader has read the previous value.

Answer: The shared resource is a subclass of `JTextField`, which it extends by defining two new methods, `getValue()`, which returns the text field's value, and `setValue()`, which assigns a value to the text field. Both of these methods are synchronized, meaning that once a thread calls one of the methods it cannot be preempted by another thread. This ensures that the reader and writer threads will have mutually exclusive access to these methods (whenever they are running and not sleeping.)

To further coordinate access to the shared text field, it contains a `writeable` control variable, which is set to true initially. It is set to false only after the writer thread has written a value to the field. It is set back to true only after the reader thread has read a value from the field. In this way writing and reading of the shared resource will occur in a alternating fashion, as the following figures shows. The GUI interface for the solution below is shown in Figure 14.4.

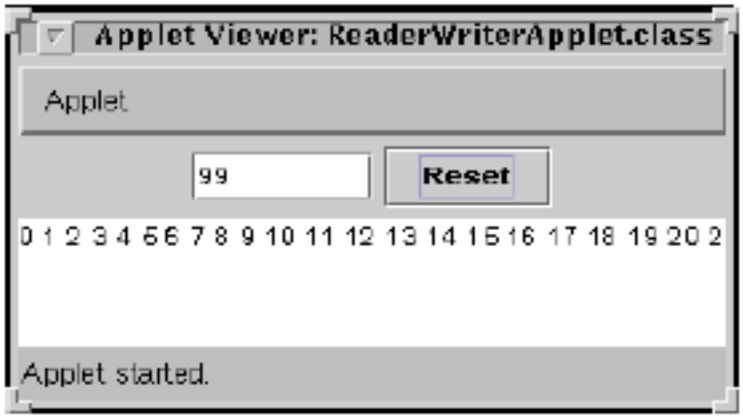


Figure 14.4: Exercise 15:Cooperating reader and writer threads produce a sequence from 0 to 99.

```
/*
 * File: SharedField.java
 * Author: Java, Java, Java
 * Description: This JTextField subclass synchronized the
 *  cooperation between a Reader and Writer thread. Its
 *  setValue() and getValue() methods are synchronized so
 *  that the reader thread can only read when a new value is
 *  available, and the writer can only write when the buffer
 *  is shared field is empty.
```

```
 */
import javax.swing.*;

public class SharedField extends JTextField {

    private boolean writeable = true; // The field is initially empty

   /**
    * SharedField() creates a JTextFIeld with a given width
    * @param width -- the text field's width
    */
    public SharedField(int width) {
        super(width);
    }

   /**
    * setValue() assigns its parameter to the text field
    *  provided the text field is writeable, which means it
    *  does not contain a value that has not yet been read.
    *  The calling thread waits until the field is writeable.
    * @param s -- a String to be assigned to the text field
    */
    public synchronized void setValue(String s) {
        while (!writeable) {
            try {
                wait();
            }
            catch (InterruptedException e) {
            }
        } // while
        setText(s);
        writeable = false;
        notify();
    } // setValue()

   /**
    * getValue() returns the text field's value, provided it
    *  has a value to return -- i.e., it is not writeable,
    *  meaning it has not yet been given a value. The calling thread
    *  waits until a string has been written to the field.
    * @return a String representing the text field's value
    */
    public synchronized String getValue() {
        while (writeable) {
            try {
                wait();
            }
            catch (InterruptedException e) {
            }
         }
```

```
            writeable = true;
            notify();
            return getText();
        } // getValue()

    } // SharedField

    /*
     * File: Writer.java
     * Author: Java, Java, Java
     * Description: This class implements a synchronized writer
     *  which shares a resource with a Reader thread. This thread
     *  repeatedly writes data to the resource.
     */
    import javax.swing.*;

    public class Writer extends Thread {

       private SharedField buffer;

       /**
        * Writer() constructor is given a pointer to the shared buffer.
        * @param b -- a reference to a shared text field
        */
       public Writer (SharedField b) {
           buffer = b;
       } // Writer()

       /**
        * run() repeatedly writes a value to the buffer. See
        *  the SharedField.getValue() method for details on how
        *  this thread is coordinated with the Reader thread.
        */
       public void run() {
           for (int k = 0; k < 100; k++) {
               buffer.setValue(k + "");
               try {
                   sleep((int) Math.random() * 2000);
               }
               catch (InterruptedException e) {
               }
           } // for
       } // run

    } // Writer

    /*
     * File: Reader.java
     * Author: Java, Java, Java
     * Description: This class implements a synchronized reader
```

```
 *  which shares a resource with a Writer thread. This thread
 *  repeatedly reads data from the resource and writes its
 *  value to the JTextArea display.
 */
import javax.swing.*;

public class Reader extends Thread {

   private SharedField buffer;
   private JTextArea display;

   /**
    * Reader() constructor is given pointers to the shared buffer
    *  and to  the User Interface's JTextarea.
    * @param b -- a reference to a shared text field
    * @param ta -- a reference to the UI's text area
    */
   public Reader (SharedField b, JTextArea ta) {
       buffer = b;
       display = ta;
   } // Reader()

   /**
    * run() repeatedly reads a value from the buffer. See
    *  the SharedField.getValue() method for details on how
    *  this thread is coordinated with the Writer thread.
    */
   public void run() {
       String value;
       for (;;) {
           try {
               sleep((int) Math.random() * 2000);
           }
           catch (InterruptedException e) {
           }

           value = buffer.getValue();
           int intValue = Integer.parseInt(value);
           display.append(value + " ");
            // Wrap around every 20 numbers
           if ((intValue + 1) % 20 == 0)
              display.append("\n");
       }
   } // run
} // Reader

/*
 * File: ReaderWriter.java
 * Author: Java, Java, Java
 * Description: Sets up  reader/writer cooperation between
```

```
* two threads using a shared resource, in this case a
* JTextField. The writer thread repeatedly writes values
* to the textfield, and the reader thread repeatedly reads
* values from it. There actions are coordinated.
*/

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

public class ReaderWriter extends JFrame
                              implements ActionListener {
  public static final int WIDTH=425, HEIGHT=300;
    // Restarts things
  private JButton reset = new JButton("Reset");
    // The shared resource
  private SharedField buffer = new SharedField(8);
  private JTextArea display = new JTextArea();
  private JPanel controls = new JPanel();

  private Reader reader;
  private Writer writer;

  /**
   * ReaderWriter() constructor sets up program's interface.
   */
  public  ReaderWriter(String title) {
      super(title);
      setSize( WIDTH, HEIGHT );
      reset.addActionListener(this);
      controls.add(buffer);
      controls.add(reset);
      getContentPane().add(controls, "North");
      getContentPane().add(display, "Center");

      reader = new Reader(buffer, display);
      reader.start();
      writer = new Writer(buffer);
      writer.start();
  } //init

  /**
   * actionPerformed() method creates new Reader and
   * Writer threads whenever the reset button is clicked
   * and resets the display.
   */
  public void actionPerformed(ActionEvent e) {
      display.setText("");
      writer = new Writer(buffer);
      writer.start();
```

```
        reader = new Reader(buffer, display);
        reader.start();
    } // actionPerformed()

    /**
     * main() creates an instance of the interface.
     */
    public static void main( String args[] ) {
        ReaderWriter rw = new ReaderWriter("Reader/Writer");
        rw.setSize(200,200);
        rw.setVisible(true);
        rw.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()
}//ReaderWriter
```

16. **Challenge:** Create a streaming banner thread that moves a simple message across a panel. The message should repeatedly enter at the left edge of the panel and exit from the right edge. Design the banner as a subclass of `JPanel` and have it implement the `Runnable` interface. That way it can be added to any user interface. One of its constructors should take a `String` argument that let's the user set the banner's message.

    Answer: One consideration in designing this solution is you want to be able to add the banner to any container, such as a `JApplet` or `JFrame`. Since you can't add a `Thread` to a container – a thread is not a `Component` – you don't want to define the banner as a `Thread` subclass or even as implementing the `runnable` interface. Instead define the banner as a subclass of `JPanel`. Then define a `Timer` thread subclass, which can be started by the banner object. The `Timer` thread repeatedly calls the banner's `repaint()` method, causing the banner to be redrawn at a new location. By rapidly redrawing the banner, it appears to move from right to left across the panel. The GUI interface for the solution below is shown in Figure 14.5.

```
/*
 * File: Banner.java
 * Author: Java, Java, Java
 * Description: This subclass of JPanel implements a
 * streaming banner in the panel. The Banner() constructor
 * takes a single string parameter giving the message that
 * will repeatedly stream through the panel. The Banner can
 * be added to any Container. To start streaming its start()
 * method should be called. The start() method creates a
 * separate Timer thread which repeatedly calls the Banner's
 * repaint() method to paint the message at a new location.
 */
```

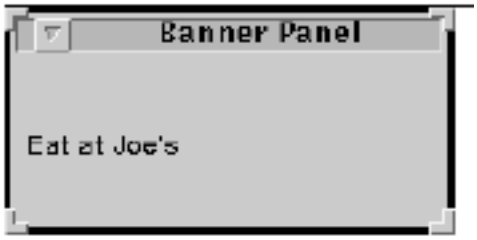


Figure 14.5: Exercise 16: The "Eat at Joe's" streaming banner advertisement.

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

public class Banner extends JPanel {

    private String message; // The panel's message
      // The message's coordinates
    private int xRef = 0, vRef = 40;

    /**
     * Banner() constructor sets the panel's message. The
     * message's initial location is beyond the right edge
     * of the panel.
     */
    public Banner(String s) {
        message = s;
          // Initial message location
        xRef = getSize().width + 10;
    }

    /**
     * start() method creates and starts a timer thread. The
     *  timer repeatedly calls the panel's repaint() method.
     */
    public void start() {
        Timer timer = new Timer(this);
        timer.start();
    } // start()

    /**
     * paintComponent() is invoked whenever the Banner is
     * repainted. It erases the entire panel and then moves
     * the message slightly to the left and repaints it.
     */
```

```
    public void paintComponent(Graphics g) {
        g.setColor(getBackground()); // Erase the panel
        g.fillRect(0,0,getSize().width, getSize().height);
        if (xRef  > -100 ) // Move the message
            xRef -= 5;
        else
            xRef = getSize().width + 10;

        g.setColor(Color.black);
        g.drawString(message, xRef, vRef); // Draw message
    } // paintComponent()

    /**
     *  main() creates an instance of Banner and adds it to
     *  a top-level JFrame. Note the use of the anonymous
     *  WindowAdapter to serve as a listener for the
     *  window close event.
     */
    public static void main(String args[]) {
        JFrame f = new JFrame("Banner Panel");
        Banner banner = new Banner("Eat at Joe's");

        f.getContentPane().add(banner);
        f.setSize(200, 100);
        f.setVisible(true);
        banner.start();
            // Quit the application
        f.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()
} // Banner

/*
 * File: Timer.java
 * Author: Java, Java, Java
 * Description: This timer thread is created by a JPanel
 *  and it repeatedly calls the JPanel's repaint() method,
 *  sleeping for an instant between each call.
 */

import javax.swing.*;

public class Timer extends Thread {
    private JPanel panel;

    /**
     * Timer() constructor sets a reference to the JPanel.
```

```
     */
    public Timer(JPanel p) {
       panel = p;
    }

    /**
     * run() repeatedly calls repaint() forcing the panel to
     *  repaint its message.
     */
    public void run() {
        try {
            while (true) {
                panel.repaint();
                sleep(100);
            }
        }
        catch (InterruptedException e) {}
    } // run
} // Timer
```

17. **Challenge:** Create a slide show applet, which repeatedly cycles through an array of images. The displaying of the images should be a separate thread. The applet thread should handle the user interface. Give the user some controls that let it pause, stop, start, speed up, and slow down the images.

    Answer: We break up this problem into two classes: an applet, `SlideShowApplet`, that serves as the user interface, and a thread subclass, `SlideMachine`, that manages the displaying of the slides themselves. Note how the slide machine class uses an internal state variable to control the running of the slides. It also contains several methods uses to control the machine. The GUI interface for the solution below is shown in Figure **??**.

```
/*
 * File: SlideMachine.java
 * Author: Java, Java, Java
 * Description: The class simulates an online slide machine.
 *  It runs as a separate thread.  The machine's internal state
 *  is either ON, OFF, or PAUSED, although there is no real
 *  difference between OFF and PAUSED, since to restart either
 *  an OFF or PAUSED machine requires that a new thread be
 *  created and started.
 */

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

public class SlideMachine extends Thread {
    private static final int NIMGS = 4;
    private static final int OFF = 0;     // Machine states
```

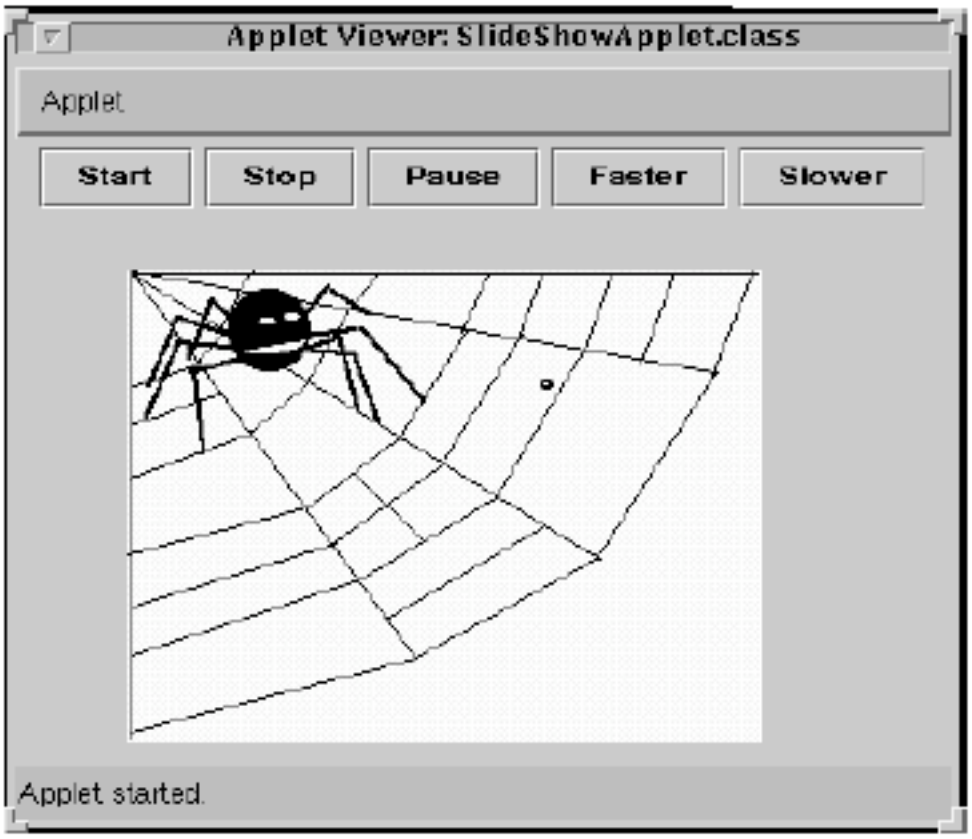


Figure 14.6: Exercise 17: The `SlideShowApplet`.

```
private static final int ON = 1;
private static final int PAUSED = 2;

private JApplet app;                  // Link to applet
private Image[] slide = new Image[NIMGS];
private Image currentImage = null;
private int nextImg = 0;
private int delay = 1000;   // The delay used by sleep()
private int state = OFF;

/**
 * SlideMachine() loads the images into an array.
 */
public SlideMachine(JApplet a) {
    app = a;
    for (int k=0; k < NIMGS; k++) {
        slide[k] = a.getImage(a.getCodeBase(),
                              "slide" + k + ".gif");
    }
    state = ON;
} // SlideMachine()

/**
 * nextSlide() sets currentImage to the next slide
 *  in the array, wrapping around to 0 if necessary.
 *  It then draws the image on the applet.
 */
public void nextSlide( ) {
    currentImage = slide[nextImg];
    nextImg = (nextImg + 1) % NIMGS;
    Graphics g = app.getGraphics();
    if (currentImage != null)
         g.drawImage(currentImage,50,60,app);
} // nextSlide()

/**
 * setDelay() sets the delay between slides to d.
 *  Negative delays are not allowed.
 * @param d -- an int giving the size of the delay.
 */
public void setDelay(int d) {
    if (d > 0)
        delay = d;
    else
        delay = 1;
} // setDelay()

/**
 * getDelay() returns the current delay value
 * @return an int giving the value of the delay
```

```
     */
    public int getDelay() {
        return delay;
    } // getDelay()

    /**
     * stopSlides() puts the machine in OFF state
     *  causing it to exit the run loop.
     */
    public void stopSlides() {
        state = OFF;
    }

    /**
     * pause() puts the machine in PAUSED state
     *  causing it to exit the run loop.
     */
    public void pause() {
        state = PAUSED;
    }

    /**
     * run() repeatedly calls the nextSlide()
     *  method and then sleeps for a certain period.
     */
    public void run() {
        state = ON;
        try {
            while (state == ON) {
                nextSlide();
                Thread.sleep(delay);
            } // while
        }
        catch (InterruptedException e) {}
    } // run()
} // SlideMachine

/*
 * File: SlideShowApplet.java
 * Author: Java, Java, Java
 * Description: The file implements a user interface to
 *  the SlideMachine class, a class that implements an
 *  online slide machine.
 */

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

 public class SlideShowApplet extends JApplet
```

```
                        implements ActionListener {
public static final int WIDTH=425, HEIGHT=300;
  // User interface buttons
private JPanel controls = new JPanel();
private JButton start = new JButton("Start");
private JButton stop = new JButton("Stop");
private JButton pause = new JButton("Pause");
private JButton faster = new JButton("Faster");
private JButton slower = new JButton("Slower");

private SlideMachine slidemachine; // The slide machine

/**
 *  init() initializes the user interface and starts
 *   the slide machine thread.
 */
public void init() {
    setSize( WIDTH, HEIGHT );
    controls.add(start);
    controls.add(stop);
    controls.add(pause);
    controls.add(faster);
    controls.add(slower);
    start.addActionListener(this);
    stop.addActionListener(this);
    pause.addActionListener(this);
    faster.addActionListener(this);
    slower.addActionListener(this);

    slidemachine = new SlideMachine(this);
    slidemachine.start();
    getContentPane().add("North", controls);
} //init()

/**
 *  actionPerformed() handles actions on the several buttons
 *   that control the slide show. Note that the start button
 *   causes a new thread to be created.
 */
 public void actionPerformed(ActionEvent e) {
     JButton b = (JButton) e.getSource();
     if (b == slower)
         slidemachine.setDelay(slidemachine.getDelay() + 300);
     else if (b == faster)
         slidemachine.setDelay(slidemachine.getDelay() - 300);
     else if (b == stop)
         slidemachine.stopSlides();
     else if (b == pause)
         slidemachine.pause();
     else if (b == start) {
```

```
                slidemachine = new SlideMachine(this);
                slidemachine.start();
            }
        } // actionPerformed()

    }//SlideShowApplet
```

18. **Challenge:** Create a horse race simulation, using separate threads for each of the horses. The horses should race horizontally across the screen, with each horse having a different vertical coordinate. If you don't have good horse images to use, just make each horse a colored polygon or some other shape.

    Answer: The `Horse` class is modeled after the `Ball` class used in the bouncing ball applet. The horse's image randomly moves from left to right across the screen. The top-level frame, `RaceTrack`, serves as the user interface for this application. It also manages the race by providing two *callback* methods: `raceOver()`, a boolean method that is checked by each horse before moving forward, and `stopRace()`, the method called by the first horse to reach the finish line. These methods are synchronized to insure that they are executed by one horse at a time. The GUI interface for the solution below is shown in Figure 14.7.

Figure 14.7: Exercise 18: The horse race simulation.

```
/*
 * File: Horse.java
 * Author: Java, Java, Java
 * Description: This class simulates a race horse. The horse
 * is represented by a colored oval at a certain location
 * on the JPanel. On each iteration of run() the horse moves
```

```
 * ahead by a random amount. When it reaches the finish line
 * it calls the frame's stopRace() method. This causes the
 * other horses to stop. Thus the top-level frame acts as
 * an intermediary between the horses and controls the stopping
 * and starting of the race.
 */

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

public class Horse extends Thread implements Runnable {
      // Diameter of the horse
    public static final int SIZE = 5;
      // Maximum length of move
    private static final int MAXDX = 10;

    private int finishLine;  // Boundary
    private boolean finished = false; // At finish line?

    private Dimension size;
      // Current location of the horse
    private Point location = new Point(20, 20);
    private RaceTrack frame;
    private JPanel panel;  // Drawing panel
    private Color color = Color.black;


    public Horse() {} // Default constructor

    /**
     * Horse() constructor sets pointers to the frame and drawing
     * panel and initializes the horse's location and color.
     */
    public Horse(RaceTrack f, JPanel p, Color c, Point loc) {
        frame = f;
        panel = p;
        size = p.getSize();
        color = c;
        location = loc;        // Set initial location
    } // Horse()

    /**
     * draw() either draws or erases the horse. When
     * XORmode is used it has the effect of drawing and
     * erasing the horse on each successive call to draw().
     */
    public void draw(Color color) {
        Graphics g = panel.getGraphics();
        g.setXORMode(color); // Note use of XOR mode here
          //  to either draw or erase
```

```
            g.fillOval(location.x, location.y, SIZE, SIZE);
        } // draw()

        /**
         * move() moves the horse forward by a random
         * amount between 0 and MAXDX.
         */
         public void move() {
             int dx = (int)(Math.random() * MAXDX);
               // Calculate a new location
             location.x = location.x + dx;
             finished = location.x >= size.width - SIZE;
             if (finished) frame.stopRace();
        } // move()

        /**
         * run() repeatedly draws, erases and moves the horse
         */
        public void run() {
            finished = false;
            while (!finished && !frame.raceOver()) {
                draw(color); // Draw
                try {
                    sleep(200);
                }
                catch (InterruptedException e) {}
                  // Erase = XOR Draw at same location
                draw(color);
                move(); // Move
           } // while
           if (finished) {
               location.x = size.width - SIZE;
               draw(Color.black); // Single out the winner
           }
           else
               draw(color);
        } // run()
    } // Horse

    /*
     * File: RaceTrack.java
     * Author: Java, Java, Java
     * Description: This class serves as the interface
     * and controller for a simulated horse race. The
     * user is provided buttons to start and reset the
     * race. When started, five horse threads are created
     * and started. The race is controlled by a pair
     * of synchronized callback methods, raceOver() and
     * stopRace(). The first is checked repeatedly by
     * each horse before it can move ahead. The second
```

```
 * is called by the horse that first reaches
 * the finish line.
*/
import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;

public class RaceTrack extends JFrame
                       implements ActionListener{
    private JButton start = new JButton("Start");
    private JButton reset = new JButton("Reset");
    private JPanel controls = new JPanel();
    private JPanel raceCourse = new JPanel();
    private Border b =
               BorderFactory.createEtchedBorder();
      // Stores references to the horses
    private Horse stable[] = new Horse[5];
      // Used to control the race
    private boolean raceIsOver = false;

    /**
     * RaceTrack() constructor sets up a simple
     * interface for the horse  race. Two buttons are
     * provided and a border is drawn around the
     * racing area.
     */
    public RaceTrack() {
        start.addActionListener(this);
        reset.addActionListener(this);
        controls.add(start);
        controls.add(reset);
        getContentPane().add(controls,"North");
        getContentPane().add(raceCourse,"Center");
        raceCourse.setBorder(
             BorderFactory.createTitledBorder(b,"Race Course",
             TitledBorder.TOP, TitledBorder.CENTER));
    } // RaceTrack

    /**
     * actionPerformed() handles clicks on the two buttons.
     * When the start  button is clicked, five horses are
     * created and started. On reset, the raceIsOver control
     * variable is reinitialized. Note how the horse's colors
     * are set. (30,30,30) is a shade of gray and each horse
     * is assigned a different shade of gray.
     */
    public void actionPerformed(ActionEvent e) {
        int col = 30; // Initial color = 30,30,30 (gray)
        if (e.getSource() == start)
```

```
            for (int k = 0; k < stable.length; k++) {
                stable[k] = new Horse(this, raceCourse,
                  new Color(col,col,col), new Point(10, 40 + 20 * k));
                stable[k].start();
                col += 40; // Make the color a bit more gray
            }
            else {
                raceIsOver = false;
                repaint();
            }
        }

        /**
         * stopRace() is a callback method that is invoked as soon
         *  as a horse reaches the finish line. It sets the control
         *  variable, raceIsOver, to true. We synchronize this method
         *  so that only one horse can execute it at a time.
         */
        public synchronized void stopRace() {
            raceIsOver = true;
        }

        /**
         * raceOver() is a callback method that is invoked by each
         *  horse just before it makes its next move. A horse is allowed
         *  to proceed only if the race is not over. We synchronize this
         *  method so that only one horse can execute it at a time.
         */
        public synchronized boolean raceOver() {
            return raceIsOver;
        }

        /**
         * main() creates an instance of this class.
         */
        public static void main(String args[]) {
            RaceTrack rt = new RaceTrack();
            rt.setSize(500,300);
            rt.setVisible(true);
        } // main()
    } // RaceTrack
```

19. **Challenge:** Create a multithreaded digital clock application. One thread should keep time in an endless loop. The other thread should be responsible for updating the screen each second.

    Answer: The `Clock` class extends the `Thread` class and uses `Calendar` objects to monitor the current time. The `ClockField` extends `JTextField` class and implements the `Runnable` interface. It uses a `Clock` thread to monitor the time and uses the thread executing its own `run()` method to decide when

to update the current time that it displays in its `JTextField`. A `ClockField` object is tested in the `ClockFrame` class which also contains a toggle button to demonstrate the behavior of different GUI components controlled by different threads.

```
/*
 * File: Clock.java
 * Author: Java, Java, Java
 * Description: This class simulates a clock. It extends
 * the Thread class and possesses methods for getting
 * the current seconds and the current time as a string.
 */

import javax.swing.*;
import java.awt.*;
import java.util.*; // To access Calendar

public class Clock extends Thread {

    private Calendar cal;
    private int hr;
    private int min;
    private int sec;
    private String amOrPm;
    private boolean keepRunning;

    /**
     * Clock() constructor creates a Calendar object
     * and retrieves the time.
     */
    public Clock() {
        cal = Calendar.getInstance();
        hr = cal.get(Calendar.HOUR);
        min = cal.get(Calendar.MINUTE);
        sec = cal.get(Calendar.SECOND);
        if (cal.get(Calendar.AM_PM) == Calendar.AM)
            amOrPm = "AM";
        else if (cal.get(Calendar.AM_PM) == Calendar.PM)
            amOrPm = "PM";
        else amOrPm = "??";
        keepRunning = true;
    } // Clock()

    /**
     * getTime() returns the time as a string
     * @return returns a string with hour, minute, second,
     * and either AM or PM.
     */
    public String getTime() {
       String str  = hr + ":";
```

```
   if (min == 0)
       str = str + "00" + ":";
   else if (min < 10)
       str = str + "0" + min + ":";
   else str = str + min + ":";
   if (sec == 0)
       str = str + "00";
   else if (sec < 10)
       str = str + "0" + sec;
   else str = str + sec;
   str = str + " " + amOrPm;
   return  str;
} // getTime()

/**
 * getSec() returns the current second
 * @return returns current value of sec.
 */
public int getSec() {
   return sec;
} // getSec()


/**
 * stopClock() sets keepRunning to false which
 * will cause the loop in the run() method to
 * terminate.
 */
public void stopClock() {
    keepRunning = false;
} // getSec()


/**
 * run() repeatedly checks for a new time.
 */
public void run() {
    while (keepRunning) {
           // get new time and date
        cal = Calendar.getInstance();
           // update if seconds have changed
        if (cal.get(Calendar.SECOND) != sec){
            hr = cal.get(Calendar.HOUR);
            min = cal.get(Calendar.MINUTE);
            sec = cal.get(Calendar.SECOND);
            if (cal.get(Calendar.AM_PM) == Calendar.AM)
                amOrPm = "AM";
            else if (cal.get(Calendar.AM_PM) == Calendar.PM)
                amOrPm = "PM";
            else amOrPm = "??";
```

```
            } // if
            try {
                sleep(200);
            } catch (InterruptedException e) {}
        } // while
    } // run()

    /**
     * main() creates an instance of this class to
     * test that the class works.
     */
    public static void main(String args[]) {
        Clock clock = new Clock();
        clock.start();
        System.out.println("The time is " + clock.getTime());
        int k = 0;  // counts the seconds passed
        int currSec = clock.getSec(); // current second
        while (k < 30){
            if (currSec != clock.getSec()) {
                System.out.println("The time is " + clock.getTime());
                currSec = clock.getSec();
                k++;
            } // if
        } // while
        clock.stopClock();
    } // main()

} // Clock

/*
 * File: ClockField.java
 * Author: Java, Java, Java
 * Description: This subclass of JTextField creates a Clock
 * object and uses it to display and update the time in its
 * textfield. The Runnable interface is implemented
 */

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

public class ClockField extends JTextField
                                    implements Runnable  {
     private Clock clock;
     private boolean keepUpdating;
     private int lastSec; // second on last update

    /**
     * ClockField() constructor calls the super()
     * constructor then calls init().
```

```
     */
    public ClockField() {
        super();
        init();
    } // ClockField() default constructor

    /**
     * ClockField(k) constructor calls the super(k)
     * constructor then calls init(k). This sets the size
     * (number of chars) in the JTextField.
     */
    public ClockField(int k) {
        super();
        init();
    } // ClockField() default constructor

    /**
     * init() creates a Clock object and displays
     * the time in the JTextField.
     */
    private void init() {
        clock = new Clock();
        clock.start();
        setText(clock.getTime());
        keepUpdating = true;
        lastSec = clock.getSec();
    } // init()

    /**
     * startUpdating() starts updating the textfield with
     * the current time.
     */
   public void startUpdating() {
        keepUpdating = true;
        lastSec = clock.getSec();
    } // startUpdating()

    /**
     * stopUpdating() stops updating the textfield with
     * the current time.
     */
   public void stopUpdating() {
        keepUpdating = false;
    } // startUpdating()


   /**
     * run() updates the time displayed in the JTextField
     * if keepUpdating == true..
     */
```

```
    public void run() {
         while (true) {
            if (keepUpdating){
                if (lastSec != clock.getSec()) {
                    setText(clock.getTime());
                    lastSec = clock.getSec();
                } // if
            } // if
            try {
                Thread.sleep(200);
            } catch (InterruptedException e) {}
        } // while
    } // run();
} // ClockField

/*
 * File: ClockFrame.java
 * Author: Java, Java, Java
 * Description: Creates a GUI application which contains
 * a ClockField. The ClockField uses two threads, one to
 * keep track of the time and one to update it JTextField;
 */

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

 public class ClockFrame extends JFrame
                                implements ActionListener {
   private JLabel timeLabel = new JLabel("The time is ");
   private ClockField cField = new ClockField(12);
   private JButton updateButton = new JButton("Stop Updating");
   private boolean updating = true;

   /**
    * ClockFrame() constructor sets up program's interface.
    * and starts the ClockField's clock.
    */
   public  ClockFrame(String title) {
       super(title);
       Thread cThread = new Thread(cField);
       cThread.start();
       updateButton.addActionListener(this);
       getContentPane().setLayout(new FlowLayout());
       getContentPane().add(timeLabel);
       getContentPane().add(cField);
       getContentPane().add(updateButton);
   } //ClockFrame()

   /**
```

```
     * actionPerformed() method changes the update
     * flag in cField when buttons are clicked
     */
    public void actionPerformed(ActionEvent e) {
       if (updating) {
         updating = false;
         cField.stopUpdating();
         updateButton.setText("Start Updating");
       } else {
         updating = true;
         cField.startUpdating();
         updateButton.setText("Stop Updating");
       } // else
    } // actionPerformed()

     /**
      * main() creates an instance of the interface.
      */
     public static void main( String args[] ) {
         ClockFrame cf = new ClockFrame("ClockFrame");
         cf.setSize(400,100);
         cf.setVisible(true);
         cf.addWindowListener(new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
                 System.exit(0);
             }
         });
    } // main()
}//ClockFrame
```

Chapter 15

Sockets and Networking

1. Explain the difference between each of the following pairs of terms:

   (a) *Stream* and *socket*.

   Answer: A *stream* carries data in one direction (as water flows in one direction). A *socket* carries data in two directions.

   (b) *Internet* and *internet*.

   Answer: The *Internet* is the largest interconnected set of networks on the planet, it is a specific instance of an *internet* (lower case i), which is any set of two or more networks connected by some medium, such as a router.

   (c) *Domain name* and *port*.

   Answer: A *domain name* specifies a specific computer or group or computers which have been logically grouped together. A *port* number specifies logical location (or service) on a particular machine.

   (d) *Client* and *server*.

   Answer: A *client* is a program, such as a browser, that typically makes a request for some service to be performed. A *server* waits for such requests and actually performs the service.

   (e) *Ethernet* and *Internet*.

   Answer: *Ethernet* is a low-level network protocol. *Internet* is an interconnected set of networks.

   (f) *URL* and *domain name*.

   Answer: A *URL* includes all of the information needed to locate and access a particular resource on the Internet. A *domain name* is a key component of a URL, helping to specify the machine on which the resource is located.

2. What is a *protocol*. Give one or two examples of protocols that are used on the Internet.

   Answer: A *protocol* is a set of rules that governs the communication of information. HTTP, SMTP, and FTP are all examples of protocols.

3. What service is managed by the HTTP protocol?

   Answer: The *HTTP (HyperText Transfer Protocol)* governs communication between web browsers (i.e, Internet Explorer and Navigator) and web servers.

4. Give examples of client applications that use the HTTP protocol.

   Answer: Web Clients, such as Microsoft Internet Explorer, Safari, and Netscape Navigator, and Web Servers, such as Microsoft IIS, Apache, and Netscape Enterprise Server, use the HTTP protocol.

5. Why is it important that applets be limited in terms of their network and file system access. Describe the various networking restrictions that apply to Java applets.

   Answer: Since applets are downloaded from a web server onto a client automatically, without the explicit consent of the user, it is important that an applet's ability to affect the machine it is running on be limited. For this reason, an applet cannot communicate (ie, via sockets) with any other machine than that which it was downloaded from. In addition, an applet cannot read or write files on the machine where it is running.

6. What does the *Internet Protocol* do? Describe how it would be used to join together an Ethernet and a token ring network.

   Answer: The *Internet Protocol (IP)* is used to allow computers on networks running heterogeneous low-level protocols such as Ethernet Protocol or Token Ring to communicate with each other. The IP effectively acts as a "translator" allowing the different networks to exchange data.

7. Describe one or two circumstances under which a `ConnectException` would be thrown.

   Answer: A `ConnectException` may be thrown when a host cannot be reached, or when there is no server listening on the port at which a process is attempting to connect.

8. Modify the `SlideShowApplet` so that it plays an audio file along with each slide.

   Design: The only modification needed is to include calls to input the sound files in the `init()` method, and then to `play()` them in the `paint()` method, along with showing the images. Like the images, the sounds are stored in an array.

```
/*
 * File: SlideShowApplet.java
 * Author: Java, Java, Java
 * Description: This applet illustrates how to download
 *  images from the Internet using the Applet.getImage()
 *  method. A set of slides is downloaded and stored in
 *  an array of Image. The applet then displays each slide
```

```
 *  in an infinite loop, using a Timer to determine when
 *  to switch to the next slide.
 */

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

public class SlideShowApplet extends JApplet  {
    public static final int WIDTH=300, HEIGHT=200;
    private static final int NIMGS = 3;
    private Image[] slide = new Image[NIMGS];

    private URL soundURL[] =
                   new URL[NIMGS];    // Added for sound
    private Image currentImage = null;
    private int currentImgNumber = 0; // Added for sound
    private int nextImg = 0;

    /**
     *  paint() is invoked automatically to draw
     *   the applet whenever necessary. It displays
     *   the current image.
     */
    public void paint(Graphics g) {
        g.setColor(getBackground());
        g.fillRect(0, 0, WIDTH, HEIGHT);
        if (currentImage != null) {
            g.drawImage(currentImage, 10, 10, this);
    //     play(soundURL[currentImgNumber]);
    play(getCodeBase(), currentImgNumber + ".au");
        }
    } //paint()

    /**
     *  nextSlide() picks the next slide in the array
     *  by advancing the nextImg variable mode NIMGS
     */
    public void nextSlide() {
        currentImage = slide[nextImg];
        currentImgNumber = nextImg;
        nextImg = (nextImg + 1) % NIMGS;
        repaint();
    } // nextSlide()

    /**
     *  init() sets up the applet's interface and
     *  downloads the images from GIF files. It also
     *  creates and starts a separate Timer thread.
     */
```

```
    public void init() {
      URL url = null;
      System.out.println("Starting");
      try {
        for (int k=0; k < NIMGS; k++) {
            // Uncomment the following lines after placing
            // images in same folder as applet code.
  url = new
    URL("http://starbase.trincoll.edu/~jjjava/slideshow/slide"
                                    + k + ".gif") ;
            // Uncomment the following line after placing
            // images in the same folder as applet code.
            //slide[k] = getImage( url );
          slide[k] = getImage(getCodeBase(), "slide" + k + ".gif" );
          System.out.println("Loaded image " + slide[k].toString());
    // soundURL[k] = getAudioClip(getCodeBase(), k + ".au");
        } // for
      } catch (MalformedURLException e) {
         System.out.println("ERROR: Malformed URL: "+url.toString() );
      } // catch()

        Thread timer = new Thread(new Timer(this));
        timer.start();
        setSize( WIDTH, HEIGHT );
    } // init()
} // SlideShowApplet

/*
 * File: Timer.java
 * Author: Java, Java, Java
 * Description: This class implements the Runnable
 *  interface, so it runs as a separate thread. It
 *  repeatedly calls the applet's nextSlide() method
 *  at the end of a fixed time interval. Between calls
 *  to nextSlide() it sleeps.
 */

public class Timer implements Runnable {
    private SlideShowApplet applet;

    /**
     *  Timer() constructor is given a reference to its
     *   associated applet.
     */
    public Timer(SlideShowApplet app) {
        applet = app;
    }

    /**
     *  run() enters an infinite loop in which it calls
```

```
     *   the applet.nextSlide() method and then sleeps for 5
     *   seconds.
     */
    public void run() {
        try {
            while (true) {
                applet.nextSlide();
                Thread.sleep(5000);
            }
        } catch (InterruptedException e) {
            System.out.println(e.getMessage());
        }
    } // run()
} // Timer
```

9. Design and implement a Java application that downloads a random substitution cryptogram and provides an interface that helps the user try to solve the cryptogram. The interface should enable the user to substitute an arbitrary letter for the letters in the cryptogram. The cryptogram files should be stored in the same directory as the application itself.

   Answer: The interface should show the user the cryptogram and the result of replacing letters in the cryptogram's alphabet. The cryptogram is displayed in one text area and the solution in another. A pair of `JComboBox`es are used to make the substitutions. The user selects a letter from the cryptogram and a letter from the replacement alphabet, and then clicks a button to apply the substitution to the cryptogram. A number `BorderLayout` panels are used to arrange the components. The GUI interface for the solution below is shown in Figure 15.1.

```
/*
 * File: CryptoViewer.java
 * Author: Java, Java, Java
 * Description: This program provides a GUI interface to
 *  assist the user in solving a cryptogram that is downloaded
 *  from the Internet. The cryptogram is downloaded into a
 *  TextArea. The user can then use a pair of JComboBoxes to
 *  create substitutions (e.g., 'B' for 'W'), which are then
 *  applied to the cryptogram. The result is shown in a second
 *  TextArea.
 */

import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import javax.swing.*;
import java.util.Random;

public class CryptoViewer extends JFrame implements ActionListener {
```

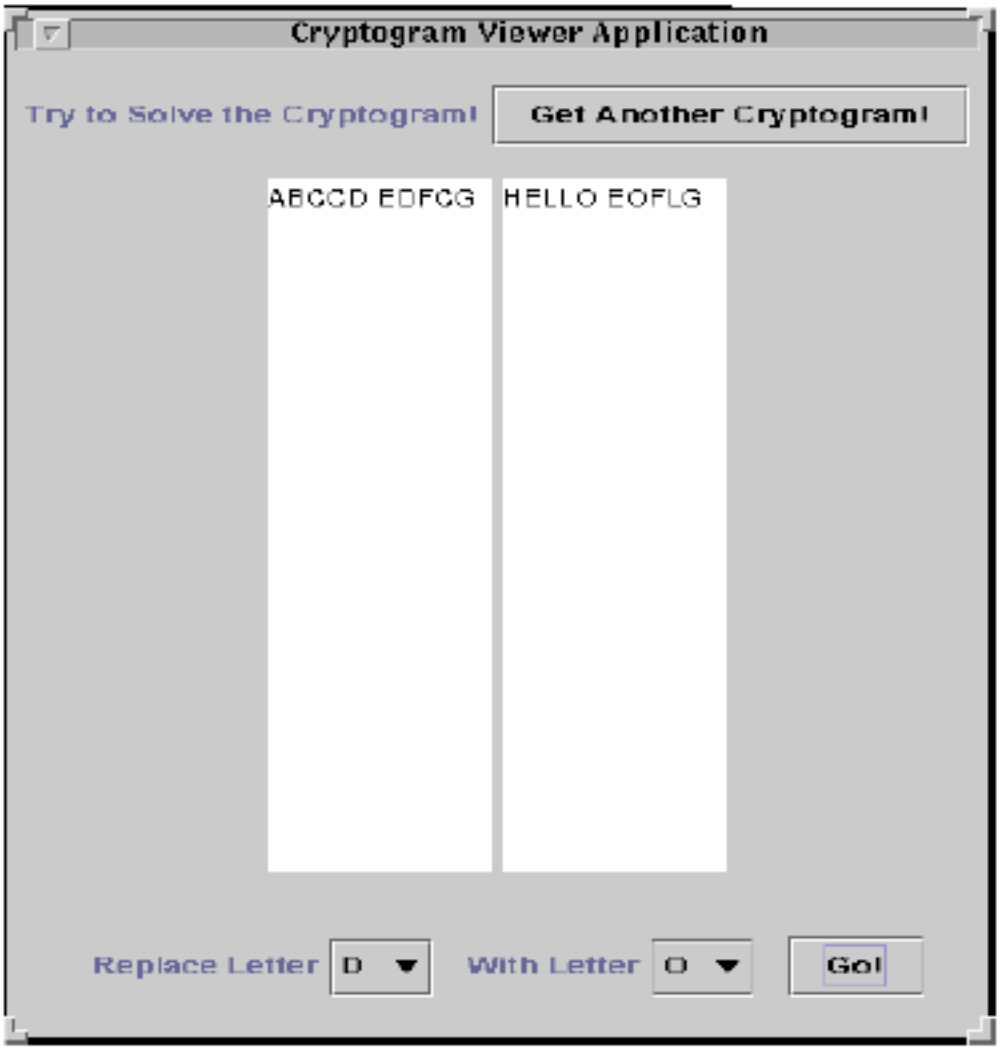


Figure 15.1: Exercise 9: A Java application for displaying and solving simple cryptograms.

```
public static final int WIDTH=500, HEIGHT=400, MESSAGES=1;
private final String baseURL = "http://www2.trincoll.edu/~jmazur/";
private String ReplacementAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
private String solution = new String();
private JPanel p1 = new JPanel();
private JPanel p2 = new JPanel();
private JPanel p3 = new JPanel();
private JPanel p4 = new JPanel();
private JPanel p5 = new JPanel();
private JPanel p6 = new JPanel();
private JPanel p7 = new JPanel();
private JLabel label0 = new JLabel("Try To Solve The Cryptogram");
private JLabel label1 = new JLabel("Replace Letter");
private JLabel label2 = new JLabel("With Letter");
private JComboBox oldLetter= new JComboBox();
private JComboBox newLetter= new JComboBox();
private JButton go = new JButton("Go!");
private JTextArea display = new JTextArea(20,10);
private JTextArea display2 = new JTextArea(20,10);
private JButton getCrypto = new JButton("Get Another Cryptogram!");

/**
 * CryptoViwer() sets up the user interface and
 * downloads and displays the cryptogram.
 */
public CryptoViewer () {
      // Set the window title
    super("Cryptogram Viewer Application");
    getCrypto.addActionListener(this);
    go.addActionListener(this);
    initLetterChoices();
    p6.add("North", label0);
    p6.add("South", getCrypto);
    p7.add("North",p6);
    p5.add("North", display);
    p5.add("South", display2);
    p7.add("South",p5);
    this.getContentPane().add("North",p7);
    p1.add("East",label1);
    p1.add("West",oldLetter);
    p2.add("East",label2);
    p2.add("West",newLetter);
    p3.add("East",p1);
    p3.add("West",p2);
    p4.add("East",p3);
    p4.add("West",go);
    this.getContentPane().add("South",p4);
    display.setLineWrap(true);
    showCurrentCrypto();   // Display the cryptogram
```

```
} // CryptoViewer()

/**
 * initLetterChoices() initializes the JCryptoBoxes.
 */
private void initLetterChoices() {
    for(int i=0; i < ALPHABET.length(); i++)
    {
        oldLetter.addItem(ALPHABET.substring(i,i+1));
        newLetter.addItem(ALPHABET.substring(i,i+1));
    }
}   // initLetterChoices()

/**
 * readTextIntoDisplay() downloads the cryptogram given
 * its URL and displays it in a TextField.
 * @param url -- the URL of the cryptogram
 */
private void readTextIntoDisplay(URL url) throws IOException {
    BufferedReader data = new
        BufferedReader(new InputStreamReader(url.openStream()));
    display.setText("");   // Reset the text area
    String line = data.readLine();
    while (line != null)  { // Read each line
        display.append(line + "\n"); // And add it to the display
        line = data.readLine();
    }
    data.close();
} // readTextIntoDisplay()

/**
 * getSolution() downloads the solved cryptogram given its URL
 * @param url -- the URL of the cryptogram
 */
private void getSolution(URL url) throws IOException {
    BufferedReader data =
      new BufferedReader(new InputStreamReader(url.openStream()));
    String line = data.readLine();
    while (line != null)  {      // Read each line
        solution += line + "\n"; // And add it to the display
        line = data.readLine();
    }
    data.close();
} // getSolution()

/**
 * showCurrentCrypto() selects a random cryptogram.
 */
private void showCurrentCrypto() { // throws IOException {
    label0.setText("Try to Solve the Cryptogram!");
```

```
    ReplacementAlphabet = ALPHABET;
    URL url = null;
    URL url1 = null;
    Random rand = new Random(); // Get random message file
    int choice = (rand.nextInt() % MESSAGES) + 1;
    try { // Create url
      url = new URL(baseURL + "encrypted"+choice+".txt") ;
      readTextIntoDisplay(url);
       // Download and display text file
      url1 = new URL(baseURL + "decrypted"+choice+".txt") ;
      getSolution(url1);
      repaint();
    } catch (MalformedURLException e) {
        System.out.println( "ERROR: " + e.getMessage()) ;
    } catch (IOException e) {
        System.out.println("ERROR: " + e.getMessage()) ;
    }
} // showCurrentCrypto()

/**
 * updateDisplay2() updates the deciphered crypto
 * based on letter substitutions the user has
 * selected. The letter substitutions are stored in
 * pairs of strings.
 */
private void updateDisplay2()
{
    char oldChar, newChar;
    oldChar =
      ((String)oldLetter.getSelectedItem()).charAt(0);
    newChar =
      ((String)newLetter.getSelectedItem()).charAt(0);
    int index = ALPHABET.indexOf(oldChar);
    String temp = "";
    for(int j=0; j<ALPHABET.length(); j++)
        if(j == index)
            temp += newChar;
        else
            temp += ReplacementAlphabet.charAt(j);
    ReplacementAlphabet = temp;
    String Crypto = display.getText();
    String newCrypto = "";
    for(int i=0; i < Crypto.length(); i++)
    {   // get current char in Crypto
        char c = Crypto.charAt(i);
        try
        {     // get location in alphabet
          int alphaIndex = ALPHABET.indexOf(c);
          newCrypto +=
            ReplacementAlphabet.charAt(alphaIndex);
```

```
                }
                catch(Exception ex)
                {
                    newCrypto += c;
                }
            }
            display2.setText(newCrypto);
            if(newCrypto.equals(solution))
              label0.setText("You've Solved The Cryptogram!");
        }

        /**
         * actionPerformed() handles all user actions.
         */
        public void actionPerformed(ActionEvent evt) {
            if(evt.getSource() == getCrypto)
                showCurrentCrypto();
            else if(evt.getSource() == go)
                updateDisplay2();
        }

        /**
         * main() creates an instance and displays it.
         * An anonymous WindowAdapter is used to handle
         * window close events.
         */
        public static void main(String args[]) {
            CryptoViewer viewer = new CryptoViewer();
            viewer.setSize(viewer.WIDTH,viewer.HEIGHT);
            viewer.setVisible(true);
               // Quit the application
            viewer.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
        } // main()
    } // CryptoViewer
```

10. Design and implement a Java applet that displays a random message (or a random joke) each time the user clicks a `GetMessage` button. The messages should be stored in a set of files in the same directory as the applet itself. Each time the button is clicked, the applet should download one of the message files.

    Answer: This implementation is designed as a Java application. It assumes that the messages are stored on some publicly available web site, whose address is set as the value program's `baseURL` variable. It is assumed that the messages are named "message0.text," "message1.text," and so on.

    The interface consists of a single control button and a display text area. Each

time the button is clicked, a new message is displayed. The program constructs the URL of the message file and then reads it into memory and displays it. The GUI interface for the solution below is shown in Figure 15.2.

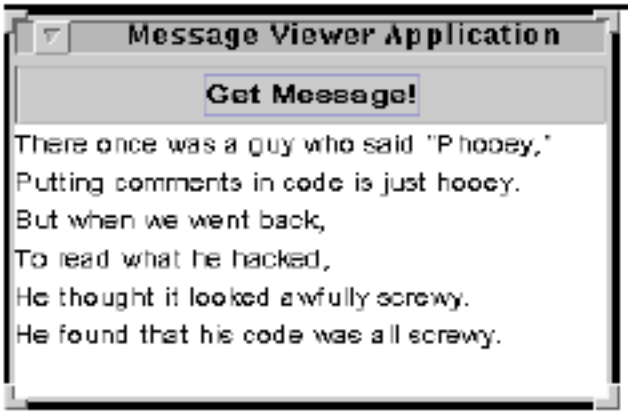


Figure 15.2: Exercise 10: A Java application for viewing messages stored on the Internet.

```
/*
 * File: MessageViewer.java
 * Author: Java, Java, Java
 * Description: This application displays
 * messages that are downloaded over the
 * Internet. To install on your system,
 * load a set of files named "message0.text",
 * "message1.txt" and so on, into some publicly
 * available web site. Then change the baseURL
 * and recompile.
 */

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

import java.util.Random;

public class MessageViewer extends JFrame
                   implements ActionListener {
    public static final int
               WIDTH=500,HEIGHT=300,MESSAGES=3;
    private final String baseURL =
```

```
  "http://troy.trincoll.edu/~jjjava/messages/";
private JTextArea display =
                        new JTextArea(20,20);
private JButton getMessage =
                  new JButton("Get Message!");

/**
 * MessageViewer() constructor sets up the
 * user interface and displays the first message.
 */
public MessageViewer () {
       // Set the window title
    super("Message Viewer Application");
    getMessage.addActionListener(this);
    this.getContentPane().add("North",getMessage);
    this.getContentPane().add("Center",display);
    display.setLineWrap(true);
        // Display the current home
    showCurrentMessage();
} // MessageViewer()

/**
 * readTextIntoDisplay() reads text from the
 * supplied URL into a TextArea.
 * @param url -- the URL giving the location
 * of a text file.
 */
private void readTextIntoDisplay(URL url)
                             throws IOException {
    BufferedReader data
        = new BufferedReader(new
            InputStreamReader(url.openStream()));
    display.setText("");  // Reset the text area
    String line = data.readLine();
    while (line != null)  {    // Read each line
          // And add it to the display
        display.append(line + "\n");
        line = data.readLine();
    }
    data.close();
} // readTextIntoDisplay()

/**
 * showCurrentMessage() cycles through
 * 0, 1, ... MESSAGES, displaying each
 * messageon successive calls.
 */
private void showCurrentMessage() {
      // throws IOException {
    URL url = null;
```

```
            // Get random message file
        Random rand = new Random();
        int choice =
            (Math.abs(rand.nextInt()) % MESSAGES);
        try { // Create url
            url = new URL(baseURL + "message" +
                                choice + ".txt");
              // Download and display text file
            readTextIntoDisplay(url);
            repaint();
        } catch (MalformedURLException e) {
            System.out.println( "ERROR: "
                               + e.getMessage()) ;
        } catch (IOException e) {
            System.out.println("ERROR: "
                               + e.getMessage()) ;
        }
    } // showCurrentMessage()

    /**
     * actionPerformed() shows the next message
     * each time the button is clicked.
     */
    public void actionPerformed(ActionEvent evt) {
        showCurrentMessage();
    }

    /**
     * main() creates an instance of this class and uses it
     *  to display some messages across the Internet.
     */
    public static void main(String args[]) {
        MessageViewer viewer = new MessageViewer();
        viewer.setSize(viewer.WIDTH,viewer.HEIGHT);
        viewer.setVisible(true);
          // Quit the application
        viewer.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    } // main()
} // MessageViewer
```

11. Write a client/server application of the message or joke service described in the previous exercise. Your implementation should extend the `Server` and `Client` classes.

    Answer: The solution to this exercise should follow the design of the object-oriented client-server application described in this chapter. The `MessageServer`

class should extend `Server`, which should extend `ClientServer`. Neither of the latter two classes need to be modified. The `MessageClient` class should extend the `Client` class, which extends `ClientServer`. The `MessageServer` class should implement the abstract `provideService()` method, which is defined in `Server`. Similarly, the `MessageClient` class should implement the `requestService()` method, which is defined abstractly in `Client`. In other words, given the infrastructure provided by the generic `Client` and `Server` classes, this new service can be implemented by defining `requestService()` and `provideService()`. These methods describe the particular functionality of this service.

The program should produce something like the following output on the client side:

```
java MessageClient troy
CLIENT: connected to troy.trincoll.edu:10001
MESSAGE FROM SERVER: Hello, how may I help you?
Type EXIT to Quit, or anything else for another message
INPUT: hello
SERVER: The prof found it incredibly odd
INPUT: help
SERVER: That the student was out on quad
INPUT: more
SERVER: Catching up on his resting,
INPUT: more
SERVER: When he should have been testing,
INPUT: more
SERVER: His code which was awfully sad.
INPUT: EXIT
CLIENT: connection closed
```

```
/*
 * File: MessageClient.java
 * Author: Java, Java, Java
 * Description:  This Client subclass provides an
 * interface to a message service, which consists of
 * displaying lines of the files named 'message0.txt',
 * 'message1.txt' and 'message2.txt' to clients.
 *
 * Usage: recompile this file and its superclasses
 * after changing the URL for the server in main().
 * Then run it with: java MessageClient
 */

import java.net.*;
import java.io.*;

public class MessageClient extends Client {
```

```
     public MessageClient(String url, int port) {
         super(url, port);
     }

    /**
     * requestService() defines the protocol that
     * must be observed to interact with MessageServer.
     * It is defined abstractly in the Client superclass.
     * @param socket -- the Socket used to connect to
     *   the server.
     */
    protected void requestService(Socket socket)
                                  throws IOException {
          // Check for "Hello"
        String servStr = readFromSocket(socket);
          // Report the server's response
        System.out.println("MESSAGE FROM SERVER: "
                                          + servStr);
          // Prompt the user
        System.out.println("Type EXIT to Quit," +
             " or anything else for another message");
        if (servStr.substring(0,5).equals("Hello")) {
            String userStr = "";
            do {  // Get input from user
                userStr = readFromKeyboard();
                    // Send it to server
                writeToSocket(socket, userStr + "\n");
                   // Read the server's response
                servStr = readFromSocket(socket);
                   // Report the server's response
                System.out.println("SERVER: " + servStr);
                   // Until user says 'exit'
            } while (!userStr.toUpperCase().equals("EXIT"));
        }
    } // requestService()

    /**
     *  main() creates a client instance and starts it.
     * You may need to change the URL for the server.
     */
    public static void main(String args[]) {
        MessageClient client =
            new MessageClient("troy.trincoll.edu",10001);
       client.start();
    } // main()
} // MessageClient


/*
 * File: MessageServer.java
```

```
 * Author: Java, Java, Java
 * Description:  This Server subclass provides
 * a message service, which consists of displaying
 * lines of the files named 'message0.txt',
 *  'message1.txt' and 'message2.txt' to clients.
 *
 * Usage: recompile this file and its superclasses
 * after changing the path name of the baseURL.
 * Place the message files in the directory
 * specified for the baseURL. Then run it with:
 * java MessageServer
 */

import java.net.*;
import java.io.*;
import java.util.Random;

public class MessageServer extends Server {
        // Total number of messages
    private static final int MESSAGES = 3;

    private static final String baseURL =
      "http://troy.trincoll.edu/~jjjava/exsolutions"
       + "/ch15/code/messageserver/";

    public MessageServer( int portNum, int nBacklog)  {
        super(portNum, nBacklog);
    }

    /**
     * provideService() implements the message
     * service. It is defined abstractly in the
     * Server superclass. It defines the protocol
     * that must be observed by clients that wish
     * to use this service.
     * @param socket -- the Socket by which the
     * client is connected
     */
    protected void provideService (Socket socket) {
        String str="";
        try {
            writeToSocket(socket,
                   "Hello, how may I help you?\n");
            do {
                str = readFromSocket(socket);
                if (str.toUpperCase().equals("EXIT"))
                  writeToSocket(socket, "Goodbye\n");
                else
                  writeToSocket(socket, getNextMessage());
            }  while (!str.toUpperCase().equals("EXIT"));
```

```
    } catch (IOException e) {
        e.printStackTrace();
    }
} // provideService()

/**
 * getFileText() reads one line of the message
 * from the designated file.
 * @param url -- the file's URL
 * @return a String containing one line of the message
 */
private String getFileText(URL url) throws IOException {
    String msg = "";
    BufferedReader data
        = new BufferedReader(new
              InputStreamReader(url.openStream()));
    String line = data.readLine();
    while (line != null)  {    // Read each line
         // And add it to the display
        msg += line + "\n";
        line = data.readLine();
    }
    data.close();
    return msg;
} // getFileText()

/**
 * getNextMessage() gets the next line of
 * the message from a file whose  URL is
 * constructed here.
 * @return a String containing one line
 *   of the message
 */
private String getNextMessage() {
    URL url = null;
    String Message = "";
      // Get random message file
    int choice = (int)(Math.random() * MESSAGES);
    try {  // Create url
        url = new URL(baseURL + "message" +
                                choice + ".txt") ;
           // Note: To work with the
           // Client-Server abstraction
        Message = getFileText(url);
           // in the text, files must be
    } catch (Exception e) {
           // one line long each.
        return ("INTERNAL SERVER ERROR: "
                                + e.getMessage());
    }
```

```
        return Message;
    }

    /**
     *  main() creates a server and starts it.
     */
    public static void main(String args[]) {
        MessageServer server =
                          new MessageServer(10001,3);
        server.start();
    } // main()
} // MessageServer
```

12. Write an implementation of the scramble service. Given a word, the scramble service will return a string containing all possible permutations of the letters in the word. For example, given "man," the scramble service will return "amn, anm, man, mna, nam, nma." Use the `Server` and `Client` classes in your design. (See the Self-Study Exercises for a description of the design.)

    Design: The solution should follow the same design as in the previous exercise. The functionality for this service should be placed in the `requestService()` and `provideService()` methods. A challenging part of this exercise is designing an algorithm to compute all the permutations of the letters of a string. See the documentation in the code for details.

    The program should produce something like the following output on the client side:

```
CLIENT: connected to troy:10001
MESSAGE FROM SERVER: Hello, how may I help you?
Type EXIT to Quit, or type a word to send to the Scramble Server
INPUT: ajax
SERVER: ajax ajxa aajx aaxj axja axaj jaax jaxa jaax jaxa jxaa jxaa aajx
        aaxj ajax ajxa axaj axja xaja xaaj xjaa xjaa xaaj xaja
INPUT: exit
SERVER: Goodbye
CLIENT: connection closed
```

```
/*
 * File: ScrambleServer.java
 * Author: Java, Java, Java
 * Description:  This Server subclass provides a word
 * scrambling service, which consists of calculating all
 * the permutations of a word that is supplied by a client.
 *
 * Usage: java ScrambleServer
 *
 * Algorithm: The permutations are calculated recursively.
 * For a given string, say "act", the algorithm takes all
```

```
 * possible singleton prefixes -- "a", "c", "t" -- and then
 * combines them with all permutations of the suffixes --
 * "ct", "at", "ac". The algorithm recurses on the suffixes.
 * When the suffix is a single letter, the  recursion stops.
 * Here's a trace of the recursive calls for permuting "act".
 * here's how the third parameter, list, carries along the
 * list of permutations. Note also how the prefix is used
 * to produce all possible permutations starting with one
 * of the characters of the second parameter.
 *
 *    getPermutations("", "act", " ")
 *       getPermutations("a", "ct", " ")
 *          getPermutations("ac", "t", " ") --> "act"
 *          getPermutations("at", "c", "act") --> "act atc"
 *       getPermutations("c", "at", "act atc")
 *          getPermutations("ca", "t", "act atc")
 *                                       --> "act atc cat"
 *          getPermutations("ct", "a", "act atc cat")
 *                                    --> "act atc cat cta"
 *       getPermutations("t", "ac", "act atc cat cta")
 *          getPermutations("ta", "c", "act atc cat cta")
 *                                  --> "act atc cat cta tac"
 *          getPermutations("tc", "a", "act atc cat cta")
 *                             --> "act atc cat cta tac tca"
 */

import java.net.*;
import java.io.*;
import java.util.Random;

public class ScrambleServer extends Server {

    public ScrambleServer( int portNum, int nBacklog)  {
        super(portNum, nBacklog);
    }

    /**
     * provideService() implements the scramble service.
     * It is defined abstractly in the Server superclass.
     * It defines the protocol that must be observed by
     * clients that wish to use this service.
     *  @param socket -- the Socket by which the client
     *       is connected
     */
    protected void provideService (Socket socket) {
        String str="";
        try {
            writeToSocket(socket,
                       "Hello, how may I help you?\n");
            do {
```

```
            str = readFromSocket(socket);
            if (str.toUpperCase().equals("EXIT"))
                writeToSocket(socket, "Goodbye\n");
            else
                writeToSocket(socket,
                     getPermutations("",str," "));
        }  while (!str.toUpperCase().equals("EXIT"));
    } catch (IOException e) {
        e.printStackTrace();
    }
} // provideService()

/**
 * getPermutations() recursively computes all the
 * permutations of a string s, returning the result
 * as a string containing a list of substrings.
 * @param prefix -- a substring to be combined with
 *     s to give a permutation
 * @param s -- the string or substring being permuted
 * @param list  --  a String containing the list of
 *     permutations
 */
private String getPermutations(String prefix, String s,
                                        String list) {
    if (s.length() <= 1) {
        return list + " " + prefix + s;    // Base case
    } else {                    // For each letter in s
        for (int i=0; i < s.length(); i++) {
              // Add it to prefix, giving new prefix
            String newPrefix = prefix + s.charAt(i);
            String newString = "";
            if (i > 0 && i < s.length()) {
                        // Set string to rest of chars
                newString = s.substring(0,i) +
                                    s.substring(i+1);
            } else if (i <= 0)
                newString = s.substring(1);
            if (newString == "")        // Base case:
                return list;
            else
                list = getPermutations(newPrefix,
                                     newString, list);
        } // for
        return list;
    } // else
} // getPermutations()

/**
 * main() creates an instance of the server and starts it.
 */
```

```
    public static void main(String args[]) {
        ScrambleServer server = new ScrambleServer(10001,3);
        server.start();
    } // main()
} // ScrambleServer

/*
 * File: ScrambleClient.java
 * Author: Java, Java, Java
 * Description:  This Client subclass provides an
 * interface to the scramble service, which permutes
 * the letters of a string passed to the service by
 * the client. The user is prompted to input strings
 * to be permuted. The server's URL is given on the
 * command line.
 *
 * Usage: java ScambleClient serverURL
 */
import java.net.*;
import java.io.*;

public class ScrambleClient extends Client {


     public ScrambleClient(String url, int port) {
         super(url, port);
     }

    /**
     * requestService() defines the protocol that
     * must be observed to interact with ScrambleServer.
     * It is defined abstractly in the Client superclass.
     * @param socket -- the Socket used to connect to
     *   the server.
     */
    protected void requestService(Socket socket)
                                throws IOException {
             // Check for "Hello"
        String servStr = readFromSocket(socket);
             // Report the server's response
        System.out.println("MESSAGE FROM SERVER: " +
                                            servStr);
             // Prompt the user
        System.out.print("Type EXIT to Quit, or type a word");
        System.out.print(" to send to the Scramble Server");
        if (servStr.substring(0,5).equals("Hello")) {
            String userStr = "";
            do {
                    // Get input from user
                userStr = readFromKeyboard();
```

```
                    // Send it to server
                writeToSocket(socket, userStr + "\n");
                    // Read the server's response
                servStr = readFromSocket(socket);
                    // Report the server's response
                System.out.println("SERVER: " + servStr);
                    // Until user says 'EXIT'
            } while (!userStr.toUpperCase().equals("EXIT"));
        }
    } // requestService()

    /**
     * main() creates an instance of the client and starts it. Note
     * that the server's URL is given as a command line argument.
     */
    public static void main(String args[]) {
        ScrambleClient client = new ScrambleClient(args[0],10001);
        client.start();
    } // main()
} // ScrambleClient
```

13. **Challenge:** Modify the Nim server game in this chapter so that the client and server can negotiate the rules of the game, including how many sticks, how many pick-ups per turn, and who goes first.

    Answer: The solution should follow the same design as in the Nim server in this chapter. The only difference here is that the protocol between client and server must be extended in order to negotiate the rules of the game. This is accomplished by adding code to the `NimServer` class. The `NimClient` class is identical to the class in the text but is included below for completeness.

    The `OneRowNim` class needs to be modified and `NimPlayer` needs to be recompiled so that the server can construct a `OneRowNim` game to the specifications of the client user. The modified version of the `OneRowNim` class is included in the solution below.

    The program should produce something like the following output on the client side:

```
CLIENT: connected to troy:10001
MESSAGE FROM SERVER: Hi Nim player. Whoever picks up the
  last stick loses. Let's decide the rules. How many sticks
  do you want to play with?
INPUT: 15
SERVER: How many sticks can we pick up per turn?
INPUT: 3
SERVER: Who should go first? (YOU or ME)
INPUT: me
SERVER: The rules are 15 sticks, 3 max sticks per move. You
  first. Okay?
```

```
INPUT: okay
SERVER: Your move first. How many sticks do you take?
INPUT: 2
SERVER: That leaves 13 sticks. I pick up 1. 12 sticks remain.
  How many sticks do you take?
INPUT: 1
SERVER: That leaves 11 sticks. I pick up 3. 8 sticks remain.
  How many sticks do you take?
INPUT: 2
SERVER: That leaves 6 sticks. I pick up 2. 4 sticks remain.
  How many sticks do you take?
INPUT: 1
SERVER: That leaves 3 sticks. I pick up 3. There's no more
  sticks. I win!
CLIENT: connection closed

/*
 * File: NimClient.java
 * Author: Java, Java, Java
 * Description:  This Client subclass provides an interface to the
 * Nim service, which plays a game of Nim with the user. The server's
 * URL is "LocalHost".
 *
 * NOTE: In order to compile and run this program access is needed
 *  to the following classes:
 *    Server.class, Client.class, ClientServer.class
 *    KeyboardReader.class
 *    OneRowNim.class, NimPlayer.class
 *    The twoplayer class hierarchy
 */
import java.net.*;
import java.io.*;

public class NimClient extends Client {
    private KeyboardReader kb = new KeyboardReader();

    public NimClient(String url, int port) {
        super(url, port);
    }

    /**
     * requestService() defines the protocol that must be observed
     * to interact with NimServer. It is defined abstractly
     * in the Client superclass.
     * @param socket -- the Socket used to connect to the server.
     */
    protected void requestService(Socket socket) throws IOException {
        String servStr = readFromSocket(socket);
           // Report server's response
        kb.prompt("NIM SERVER: " + servStr + "\n");
```

```
        if ( servStr.substring(0,6).equals("Hi Nim") ) {
            String userStr = "";
            do {
                  // Get input from user
                userStr = kb.getKeyboardInput();
                  // Send it to server
                writeToSocket(socket, userStr + "\n");
                  // Read the server's response
                servStr = readFromSocket(socket);
                  // Report the server's response
                kb.prompt("NIM SERVER: " + servStr + "\n");
                 // Until game over
            } while(servStr.indexOf("Game over!") == -1);
        } // if
    } // requestService()

    /**
     * main() creates an instance of the client and starts it.
     * Note that the server's URL is "local host".
     */
    public static void main(String args[]) {
        NimClient client = new NimClient("localhost",10001);
        client.start();
    } // main()
} // NimClient

/*
 * File: NimServer.java
 * Author: Java, Java, Java
 * Description:  This Server subclass plays Nim with its clients.
 * In this version  the client is invited to set the rules of the
 * game, including how many sticks, the number of sticks per turn,
 * and who goes first.
 *
 * Nim Rules: In this version the game starts with STICKS sticks
 * and whoever picks up the last stick loses. On each move, a player
 * must pick up between 1 and a maximum of sticks negotiated before
 * the game begins.
 */

import java.net.*;
import java.io.*;
import java.util.Random;

public class NimServer extends Server {
       // OneRowNim object records rule choices and game moves.
       // nim has been made an instance variable so that both
       // negotiateRules() and provideService() can access it.
    private OneRowNim nim;
```

```
/**
 * NimServer() constructor creates a server object given
 *  it port number and a number representing the number of
 *  clients it can backlog.
 * @param portNum -- an int giving the port number
 * @param nBacklog -- the number of clients that can backlog
 */
public NimServer(int port, int backlog) {
    super(port, backlog);
} // nimServer() constructor

/**
 * provideService() implements the Nim service. It is defined
 * abstractly in the Server superclass. It defines the protocol
 * that must be observed by clients that wish to use this service.
 * @param socket -- the Socket by which the client is connected
 */
protected void provideService (Socket socket) {
  try {
    negotiateRules(socket); //NEW - Negotiate OneRowNim rules.
    play(socket);   //  MODIFIED - Uses the nim instance variable.
  } catch (IOException e) {
    e.printStackTrace();
  } // catch()
} // provideServer()

 /*** NEW method to set One Row Nim rules.
 * negotiateRules(socket) defines protocol for setting rules.
 * @param socket  -- the Socket by which the client is connected
 */
private void negotiateRules(Socket socket) throws IOException {

  int  sticks = 21; // These will be set by client
  int maxSticks = 3;
  boolean serverFirst = false;
  String temp;
    do {
      writeToSocket(socket,
        "Hi Nim player. Let's play NIM. Whoever picks up " +
        "the last stick loses. Let's decide the rules. " +
                "How many sticks do you want to play with?");
      sticks = Integer.parseInt(readFromSocket(socket));
      writeToSocket(socket, "How many sticks can we pick up per turn?");
      maxSticks = Integer.parseInt(readFromSocket(socket));
      writeToSocket(socket, "Who should go first? (YOU or ME)");
      if (readFromSocket(socket).toUpperCase().equals("YOU")) {
        serverFirst = true;
        writeToSocket(socket, "The rules are " + sticks + " sticks, " +
          maxSticks + " max sticks per move. Me first. Okay?");
```

```
      } else {
        serverFirst = false;
        writeToSocket(socket, "The rules are " + sticks + " sticks, " +
          maxSticks + " max sticks per move. You first. Okay?");
      } // else
      temp = readFromSocket(socket).toUpperCase();
    } while (!temp.equals("OKAY") && !temp.equals("YES"));

    // Create a OneRowNim object with these rules.
    nim = new OneRowNim(sticks, maxSticks, serverFirst);
} // negotiateRules()

/** MODIFIED play() METHOD
 * This plays the game of nim after the rules have been negotiated.
 * It is now possible that the server moves first so it is necessary
 * to check whose move it is.
 */
private void play(Socket socket) throws IOException {
    NimPlayer computerPlayer = new NimPlayer(nim);
    nim.addComputerPlayer(computerPlayer);
    // server = player two, computerPlayer plays for the server

    String str="", response="";
    int userTakes = 0, computerTakes = 0;

    // The client has just indicated OKAY I am ready to play.
    // This method must manage play until game is over.

    if (nim.getPlayer() == TwoPlayerGame.PLAYER_TWO){
           // the server moves first - NEW CODE
        response = nim.reportGameState() + " ";
        computerTakes =
            Integer.parseInt(computerPlayer.makeAMove(""));
        response = response + " My turn. I take " +
                              computerTakes + " sticks. ";
        nim.takeSticks(computerTakes);
        nim.changePlayer();
        response = response + nim.reportGameState() + " ";
        if (!nim.gameOver())
            response = response + nim.getGamePrompt();
    } else { // The client user plays first
        response = nim.reportGameState() + " ";
        response = response + nim.getGamePrompt();
    } // else
    writeToSocket(socket, response); // Ask user to make a move

    while (!nim.gameOver()){ //
        str = readFromSocket(socket);
        boolean legalMove = false;
        do {
```

```
                userTakes = Integer.parseInt(str);
                if (nim.takeSticks(userTakes)) {
                    legalMove = true;
                    nim.changePlayer();
                    response = nim.reportGameState() + " ";
                    if (!nim.gameOver()) {
                        computerTakes =
                        Integer.parseInt(computerPlayer.makeAMove(""));
                        response = response + " My turn. I take " +
                                          computerTakes + " sticks. ";
                        nim.takeSticks(computerTakes);
                        nim.changePlayer();
                        response = response + nim.reportGameState() + " ";
                        if (!nim.gameOver())
                             response = response + nim.getGamePrompt();
                    }
                    writeToSocket(socket, response);
                } else {
                   writeToSocket(socket,"That's an illegal move. Try again.\n");
                   str = readFromSocket(socket);
                } // else
            } while (!legalMove);
        } // while

        // Game is over and nim.reportGameState() has included the
        // substring "Game over!" signal to the client.
    } // play()

    /**
     * Overriding writeToSocket to remove \n from str
     */
    protected void writeToSocket(Socket soc, String str)
                                    throws IOException {
    StringBuffer sb = new StringBuffer();
        for (int k = 0; k < str.length(); k++)
            if (str.charAt(k) != '\n')
        sb.append(str.charAt(k));
        super.writeToSocket(soc, sb.toString() + "\n");
    } // writeToSocket()

    /**
     * main() creates an instance of the server and starts it.
     */
    public static void main(String args[]) {
        NimServer server = new NimServer(10001,5);
        server.start();
    } // main()
} // NimServer

/*
```

```
 * File: OneRowNim.java
 * Author: Java, Java, Java
 * Description:  This version of OneRowNim is modified to support a
 * constructor with three parameters specifying the number of sticks,
 * the maximum number of sticks per turn, and who goes first.
 * This class extends the TwoPlayerGame class and implements
 * the CLUIPlayableGame interface, which enables two players to play
 * using a command-line interface.  It makes extensive use of Java's
 * inheritance and polymorphism mechanisms.
 *
 */

public class OneRowNim extends TwoPlayerGame implements CLUIPlayableGame {
    public int MAX_PICKUP = 3; // THIS IS NO LONGER CONSTANT
    public static final int MAX_STICKS = 50; // THIS VALUE IS INCREASED
    private int nSticks = MAX_STICKS;  // A constructor may change this

    /**
     * OneRowNim() default constructor
     */
    public OneRowNim() { }      // Constructors

    /**
     * OneRowNim() constructor sets the initial number of sticks.
     * @param sticks, an int representing the number of sticks.
     */
    public OneRowNim(int sticks) {
        nSticks = sticks;
    }  // OneRowNim()

    /**
     * OneRowNim() constructor sets the initial number of sticks and
     *  the initial player.
     * @param sticks, an int representing the number of sticks.
     * @param starter, an int representing which player goes first
     */
    public OneRowNim(int sticks, int starter) {
        nSticks = sticks;
        setPlayer(starter);
    } // OneRowNim()

    /** *** THE NEW CONSTRUCTOR ***
     * OneRowNim() constructor sets the initial number of sticks,
     * the max  number of sticks per turn and which player starts.
     * @param sticks, an int representing the number of sticks
     *   when the game starts.
     * @param maxperTurn, an int representing which player goes first
     */
    public OneRowNim(int sticks, int maxPerTurn, boolean twoStarts) {
        nSticks = sticks;
```

```
    MAX_PICKUP = maxPerTurn;
    if (twoStarts)
            setPlayer(PLAYER_TWO);
    else
            setPlayer(PLAYER_ONE);
} // OneRowNim()


/**
 * takeSticks() removes a valid number of sticks and returns true
 *  or returns false if the parameters is not valid.
 * @param num, an int representing the number of sticks to take.
 * @return true is returned if the move is valid, false otherwise.
 */
public boolean takeSticks(int num) {
    if (num < 1 || num > MAX_PICKUP || num > nSticks)
        return false;               // Error
    else                            // Valid move
    {   nSticks = nSticks - num;
        return true;
    } //else
} // takeSticks()

/**
 * getSticks() returns the current number of sticks.
 */
public int getSticks() {
    return nSticks;
} // getSticks()

/**
 * getRules() is overridden from the TowPlayerGame class. It
 *  returns a String giving the games rules.
 * @return a String describing the game
 */
public String getRules() {
    return "\n*** The Rules of One Row Nim ***\n" +
      "(1) A number of sticks between 7 and " + MAX_STICKS +
      " is chosen.\n" +
      "(2) Two players alternate making moves.\n" +
      "(3) A move consists of subtracting between 1 and\n\t" +
      MAX_PICKUP + " sticks from the current number of sticks.\n" +
      "(4) A player who cannot leave a positive\n\t" +
      " number of sticks for the other player loses.\n";
} // getRules()

/**
 * gameOver() is defined as abstract in TwoPlayerGame
 *  and implemented here. A OneRowNim game is over when
 *  there are 0 sticks left.
```

```
 * @return true if the game is over and false otherwise
 */
public boolean gameOver() {
    return (nSticks <= 0);
}  // gameOver()

/**
 * getWinner() is defined as abstract in TwoPlayerGame
 *  and implemented here. It defines who wins OneRowNim.
 * @return a String describing the winner
 */
public String getWinner()  {       /** From TwoPlayerGame */
    if (gameOver()) //{
        return "" + getPlayer() + " Nice game.";
    return "The game is not over yet.";   // Game is not over
} // getWinner()

/**
 * getGamePrompt() is implemented as part of the CLUIPlayableGame
 * interface. It defines the prompt presented to the user
 * before each move.
 * @return a String giving the prompt.
 */
public String getGamePrompt() {
    return "\nYou can pick up between 1 and " +
                          Math.min(MAX_PICKUP,nSticks) + " : ";
} // getGamePrompt()

/**
 * reportGameState() is implemented as part of the CLUIPlayableGame
 * interface. It describes the current state of the game.
 * @return a String describing the game's current state.
 */
public String reportGameState() {
    if (!gameOver())
        return ("\nSticks left: " + getSticks() +
                        " Who's turn: Player " + getPlayer());
    else
        return ("\nSticks left: " + getSticks() +
          " Game over! Winner is Player "  + getWinner() +"\n");
}   // reportGameState()

/**
 * play() is implemented as part of the CLUIPlayableGame interface.
 * It contains the control algorithm for playing two-player OneRowNim.
 * @param ui gives a reference to the UserInterface where I/O is
 *   performed.
 */
public void play(UserInterface ui) {
    int sticks = 0;
```

```
        ui.report(getRules());
        if (computer1 != null)
            ui.report("\nPlayer 1 is a " + computer1.toString());
        if (computer2 != null)
            ui.report("\nPlayer 2 is a " + computer2.toString());
        while(!gameOver()) {
            IPlayer computer = null;     // Assume no computers
            ui.report(reportGameState());
            switch(getPlayer()) {
                case PLAYER_ONE:             // Player 1's turn
                  computer = computer1;
                  break;
                case PLAYER_TWO:             // Player 2's turn
                  computer = computer2;
                  break;
            } // cases
            if (computer != null) {      // If computer's turn
                sticks = Integer.parseInt(computer.makeAMove(""));
                ui.report(computer.toString() + " takes " + sticks +
                                                        " sticks.\n");
            } else {                  // otherwise, user's turn
                ui.prompt(getGamePrompt());
                      // Get user's move
                sticks = Integer.parseInt(ui.getUserInput());
            }
            if (takeSticks(sticks))   // If a legal move
                changePlayer();
        } // while
        ui.report(reportGameState()); // The game is now over
    } // play()

    /**
     * main() sets up an instance of a OneRowNim game to be played
     * USING THE NEW CONSTRUCTOR.  A game of OneRowNim is played
     * using a command-line interface between a user and a NimPlayer.
     */
    public static void main(String args[]) {
        KeyboardReader kb = new KeyboardReader();
        // USE THE NEW CONSTRUCTOR
        OneRowNim game = new OneRowNim(33, 5, true);
        game.addComputerPlayer(new NimPlayer(game));
        game.play(kb);
    } // main()

} // OneRowNim class
```

Chapter 16

Data Structures: Lists, Stacks, and Queues

1. Explain the difference between each of the following pairs of terms:

   (a) *Stack* and *queue*.
   Answer: A *stack* is a Last-In First-Out data structure (LIFO), while a *queue* is a First-In First-Out data structure (FIFO).

   (b) *Static structure* and *dynamic structure*.
   Answer: The size of a *static* structure is fixed, while the size of a *dynamic* structure may change during the course of execution.

   (c) *Data structure* and *abstract data type*.
   Answer: A *data structure* is a construct used to organize data, while an *abstract data type (ADT)* describes both the data that needs to be stored and the methods necessary to manipulate the data.

   (d) *Push* and *pop*.
   Answer: The *push* stack operation adds an element to the top of the stack, while the *pop* operation removes the top element from the stack.

   (e) *Enqueue* and *dequeue*.
   Answer: The *enqueue* queue operation adds an element to the end of the queue, while *dequeue* removes the element from the front of the queue.

   (f) *Linked list* and *node*.
   Answer: A *linked list* is made up of multiple nodes. Conversely, a *node* encapsulates one piece of data in the linked list.

2. Fill in the blanks:

   (a) An *abstract data type* consists of two main parts: _____ and _____. **Answer:** data, methods

(b) An object that contains a variable that refers to an object of the same class is a _____. **Answer:** self-referential object

(c) One application for a _____ is to manage the method call and returns in a computer program. **Answer:** stack

(d) One application for a _____ is to balance the parentheses in an arithmetic expression. **Answer:** stack

(e) A _____ operation is one that starts at the beginning of a list and processes each element. **Answer:** traversal

(f) A vector is an example of a _____ data structure. **Answer:** dynamic

(g) An array is an example of a _____ data structure. **Answer:** static

(h) By default the initial value of a reference variable is _____. **Answer:** null

3. Add a `removeAt()` method to the `List` class to return the object at a certain index location in the list. This method should take an `int` parameter, specifying the object's position in the list, and it should return an `Object`.

Answer:

```
/**
 * removeAt() destructively  removes the node at a
 * given position. The algorithm requires that the list
 * be traversed until the desired position is reached.
 * @param position -- an int giving the node's position
 * @return a reference to the Object removed
 */
public Object removeAt (int position) {
    if (isEmpty())      // Special case: empty list
        return null;
    Node current = head; // Special case: singleton list
    if (current.getNext() == null) {
        head = null;
        return current;
    }

    Node previous = null; // Otherwise traverse the list
    int counter = 0;
    while ((current.getNext() != null) &&
                                (counter < position)) {
        previous = current;
        current = current.getNext();
        counter++;
    } // while
      // Remove by setting previous pointers
    previous.setNext(current.getNext());
    return current;
} // removeAt()
```

4. Add an `insertAt()` method to the `List` class that will insert an object at a position in the list. This method should take two parameters, the `Object` to be inserted, and an `int` to designate where to insert it. It should return a `boolean` to indicate whether the insertion was successful.

   Answer:

```
/**
 * insertAt() inserts a new node at a certain
 * position within the list
 * @param newNode -- the Object to be inserted
 * @param position -- the location to insert at
 * @return -- a boolean indicating whether the
 *   insertion was successful
 */
public boolean insertAt(Object newNode,
                                int position) {
    if (isEmpty()) {    // For empty list
           //   just insert at head
        Node newHead = new Node(newNode);
        head = newHead;
        return true;
    }

    Node current = head;
    if (position == 0) { // Insert at the front
        Node newHead = new Node(newNode);
        newHead.setNext(current);
        return true;
    }

    Node previous = null;  // Traverse the list
    int counter = 0;
    while ((current.getNext() != null) &&
                        (counter < position)) {
        previous = current;
        current = current.getNext();
        counter++;
    } // while
      // If desired position is reached
    if (counter == position) {
          // insert the new node.
        Node temp = new Node(newNode);
        previous.setNext(temp);
        temp.setNext(current);
        return true;
    } // if
    return false;
} // insertAt()
```

5. Add a `removeAll()` method to the `List` class. This `void` method should

remove all the members of the list.

Answer:

```
/**
 * removeAll() removes all elements of the list by
 * simply setting its head to null. Java's automatic
 * garbage collector takes care of reclaiming the
 * memory occupied by the list's objects.
 */
public void removeAll() {
    head = null;
} // removeAll()
```

6. Write an `int` method named `size()` that returns the number of elements in a `List`

   Answer:

```
/**
 * size() returns the number of elements in the list
 * @return an int giving the number of elements
 */
public int size() {
    int counter = 0;
    Node current = head;
    while (current != null) {
        current = current.getNext();
        counter++;
    }
    return counter;
} // size()
```

7. Write an `boolean` method named `contains(Object o)` that returns `true` if its `Object` parameter is contained in the list.

   Answer:

```
/**
 * contains() returns true if the list contains
 * the object referenced by its parameter.
 * @param obj -- a reference to the object being
 *   searched for
 * @return a boolean indicating whether the list
 *   contains the object
 */
public boolean contains(Object obj) {
    Node current = head;
    while (current != null) {
        if (current.getData() == obj) {
            return true;
```

```
            } // if
            current = current.getNext();
        } // while
        return false;
    } // contains()
```

8. The *head* of a list is the first element in the list. The *tail* of a list consists of all the elements except the head. Write a method named `tail()` that returns a reference to the tail of the list. Its return value should be `Node`.

   Answer:

```
/**
 * tail() returns a reference to the tail of a list.
 * Except for the empty list, a list's tail is the list
 * starting at the next node after the head.
 * @return a reference to the first Node in the tail
 */
public Node tail() {
    if (head == null)
        return null;
    return head.getNext();
} // tail()
```

9. Write a program that uses the `List` ADT to store a list of 100 random floating-point numbers. Write methods to calculate the average of the numbers.

   Answer: A good object-oriented design for this solution is to create a subclass of `List` called `NumberList`, which is designed to store numbers. Its main feature is a method, `getAverage()`, to compute the average of the numbers stored in the list. (Of course, it could easily be extended to contain other useful number-processing methods.)

   The `getAverage()` method will traverse the list, beginning at its head, adding each value to a running total. In order to effect this design, the `head` variable must be made `protected` (instead of private) so it can be referenced in the subclass. So we make the following revision to the `List` class:

```
protected Node head; // Reference to the first Node
```

   To test the `getAverage()` method, we also define a method to store random numbers into the list.

```
/* File: NumberList.java
 * Author: Java, Java, Java
 * Description: This class uses an instance of the List ADT
 *  to create a list of floating point numbers. It includes
 *  a method to average the list.
 *
 * Modification to List class: For this program the head
```

```
 *  must be declared protected.
 */
public class NumberList extends List {

    private java.util.Random generator = new java.util.Random();

    /**
     * fillList() creates 10 random doubles and inserts them.
     * into the list. It uses an instance of the java.util.Random
     * class to generate random doubles.
     */
    public void fillList() {
        for(int x = 0; x < 10; x++) {
            double n = generator.nextDouble() * 100;
            insertAtRear(new Double(n));
        }
        System.out.println("The random list of numbers is ");
        print();
    } // fillList()

    /**
     * getAverage() traverses the list and computes the
     * average of its elements
     * @return a double giving the average of the numbers
     *   in the list
     */
    public double getAverage() {
        double total = 0.0;
        int count = 0;
        Node current = head;  // Get the head of the list
        while(current != null) {    // Traverse the list
            Double f = (Double)current.getData();
            current = current.getNext();
                // Adding each value to the total
            total = total + f.doubleValue();
            count++;
        } // while
        return (total / count); // Return the average
    } // getAverage()

    /**
     * main() creates an instance of this class and tests
     * its methods.
     */
    public static void main(String args[]) {
      NumberList app = new NumberList();
      app.fillList();
      System.out.println("The average value of the list is "
                              + app.getAverage() + ".\n");
    } // main()
```

```
} // NumberList
```

The program will produce something like the following output:

```
The random list of numbers is
14.293357756612412
31.692229478699108
45.68808020888433
40.73013928289202
66.76290105394833
94.64654445105502
57.4289343759532
78.24981938587118
..................
22.591468918532563
12.658355878815863
The average value of the list is 46.4741830791264.
```

10. Write a program that uses the `List ADT` to store a list of `Student` records, using a variation of the `Student` class defined in Chapter 11. Write a method to calculate the mean grade point average for all students in the list.

    Answer: As in the previous exercise, the `StudentList` class extends the `List` class. It contains a method, `calcAvgGPA()` to calculate the average grade-point average. The `head` variable must be declared `protected` in order to extend the `List` class in this way.

```
/*
 * File: StudentList.java
 * Author: Java, Java, Java
 * Description: This subclass of List implements a list
 *  of student records. Each Student record consists of
 *  a name, year and gpa field. (The Student class defined
 *  in Chapter 11 is used. The only change to that class is
 *  to add a getGPA() access method.
 *
 * Note that this class depends on the version of List and Node
 *  that were defined in the previous exercise. The List class
 *  must contain a public instance variable named head.
 */

public class StudentList extends List {

    /**
     * calcAvgGPA() calculates the average GPA of all students
     *  in the list. Note that it is necessary to convert the
     *  Node's data to (Student) before getGPA() can be called.
     * @return a double giving the average GPA
     */
```

```
    public double calcAvgGPA() {
        int counter = 0;
        double sum = 0;
        Node current = head;
        while (current != null) {
            sum += ((Student)current.getData()).getGPA();
            counter++;
            current = current.getNext();
        }
        return sum / counter;
    }

    /**
     * main() creates a linked list of students and then tests
     *   the calcGPA() method.
     */
    public static void main( String argv[] ) {
                    // Create list and insert heterogeneous nodes
        StudentList list = new StudentList();
        list.insertAtFront(new Student("adam", 2000, 12.9));
        list.insertAtFront(new Student("baker", 2000, 10.6));
        list.insertAtFront(new Student("curtis", 2000, 8.2));
        list.insertAtFront(new Student("dietz", 2000, 12.5));
        list.insertAtFront(new Student("eliot", 2000, 6.5));

                    // Print the list
        System.out.println("Student List");
        list.print();
        System.out.println(" The average GPA for these students is "
                                              +list.calcAvgGPA());
    } // main()
} // StudentList
```

The program will produce something like the following output:

```
Student List
eliot   2000    6.5
dietz   2000    12.5
curtis  2000    8.2
baker   2000    10.6
adam    2000    12.9
 The average GPA for these students is 10.139999999999999
```

11. Write a program that creates a copy of a `List`. It is necessary to copy each node of the list. This will require that you create new nodes that are copies of the nodes in the original list. To simplify this task, define a copy constructor for your node class and then use that to make copies of each node of the list.

    Answer: In the `Node` class we want to add a constructor method that makes a new node from a copy of an existing node:

```
/**
 * Node() copy constructor makes a new node as a copy of
 *  an existing node.
 * @param obj -- a reference to an Object
 */
public Node(Node node) {  // Constructor
    data = node.data;
    next = node.next;
}
```

Then in the `List` class, we want to add a method that returns a reference to a `List` which is made by copying each of the nodes of an existing list. By placing this method in the `List` class, you are giving objects of that class the ability to *clone* themselves:

```
/**
 * getCopy() creates a copy of each node of the list
 * and links them together into a duplicate of the original
 * list. It then returns a pointer to the head of the new list.
 * @return a reference to the head of new list
 */
public List getCopy() {
    List newList = new List();
    Node current = head;
    while(current != null) { // For each node in the list
          // make a copy of it
        newList.insertAtRear(new Node(current));
        current = current.getNext();
    }
    return newList;  // Return a reference to the new list
} // getCopy()
```

Given these changes to `Node` and `List`, we can now obtain a copy of any list using the following command:

```
List copyOfOldList = oldList.getCopy();
```

The following program tests our copy making ability by creating a list, making a copy of it, destroying the original list, and then printing the copy. This shows that the copy survived the destruction of the original list.

```
/* File: TestCopy.java
 * Author: Java, Java, Java
 * Description: This class uses an instance of the List ADT
 *  to create a list of floating point numbers. It then makes
 *  a copy of the list and prints the copy.
 *
 * This program requires that the Node class be modified to
```

```
 *  include a copy constructor of the form Node(Node). And that
 *  the List class is modified to include a getCopy() method.
 */
public class TestCopy {

    private java.util.Random generator = new java.util.Random();
    private List list = new List();  // The internal list

    /**
     * fillList() creates 10 random doubles and inserts them.
     *  into the list. It uses an instance of the java.util.Random
     *  class to generate random doubles.
     */
    public void fillList() {
        for(int x = 0; x < 10; x++) {
            double n = generator.nextDouble() * 100;
            list.insertAtRear(new Double(n));
        }
    } // fillList()

    /**
     * printCopy() creates a copy of the original list. Then
     * it removes all the nodes from the original list. Finally
     * it prints the copy of the original list, showing that the
     * nodes in the original list were indeed copied, not merely
     * doubly referenced.
     */
    public void printCopy() {
        List copy = list.getCopy();
        System.out.println("Printing the original list");
        list.print();
        while (!list.isEmpty()) {
            list.removeFirst();      // Remove each node
        }
        System.out.println("Printing the original list after" +
                                    " removing all its nodes");
        list.print();
        System.out.println("Now printing the copy of the" +
                                              " original list");
        copy.print();
    } // printCopy()

    /**
     * main() creates an instance of this class and tests its
     * methods.
     */
    public static void main(String args[]) {
        TestCopy app = new TestCopy();
        app.fillList();
        app.printCopy();
```

```
    } // main()
} // TestCopy
```

12. Write a program that uses a `Stack` ADT to determine if a string is a palindrome — spelled the same way backwards and forwards.

    Answer: A stack is ideally suited for this task because it is ideally suited for reversing a string. `isPalindrome()` algorithm simply pushes each letter of the string onto the stack, then it repeatedly pops the stack, comparing the characters with the letters in the original string. Having stored the characters in a stack means that it is comparing the last letter with the first, the second-to-last with the second, and so on.

```
/*
 * File: PalindromeTester.java
 * Author: Java, Java, Java
 * Description: This class tests the isPalindrome() method,
 * which uses a Stack to test whether a string is a palindrome.
 * A palindrome is a string that reads the same backwards and
 * forwards, such as "radar". The palindrome tester algorithm
 * relies on the stack's last-in-first-out (LIFO) behavior
 * to reverse the string and compare to the original.
 *
 * Usage: java PalindromeTester
 */
public class PalindromeTester {

    /**
     * isPalindrome() returns true if its parameter is a palindrome
     * Note how data popped from the stack must be coerced from
     * Object to Node and then from Node to Character.
     * @param w -- a String to be tested
     * @return true if w is a palindrome, false otherwise
     */
    public boolean isPalindrome(String w) {
        Stack s = new Stack();              // Create a stack

        for (int k = 0; k < w.length(); k++) // Stack the letters
            s.push(new Character(w.charAt(k)));
              // Unstacking reverses the string
        for (int k = 0; k < w.length(); k++) {
            Node node = (Node)s.pop();      // Get top node
               // Get its data
            Character currentChar = (Character)node.getData();
            if (currentChar.charValue() != w.charAt(k))
                return false;
        }
        return true;
    } // isPalindrome()
```

```
    /**
     * main() creates an instance of this class and tests the
     * isPalindrome() method on three strings.
     */
    public static void main(String args[]) {
        String word = "radar";   // word to test
        PalindromeTester pt = new PalindromeTester();
        System.out.println("isPalindrome(" + word + ") returned " +
                                            pt.isPalindrome(word));
        word = "reader";
        System.out.println("isPalindrome(" + word + ") returned " +
                                            pt.isPalindrome(word));
        word = "able was I ere I saw elba";
        System.out.println("isPalindrome(" + word + ") returned " +
                                            pt.isPalindrome(word));
    } // main()
} // PalindromeTester
```

13. Design and write a program that uses a `Stack` to determine whether a parenthesized expression is well formed. Such an expression is well formed only if there is a closing parenthesis for each opening parenthesis.

    Answer: A stack is ideally suited for this task because parentheses follow a last-in-first-out protocol. Processing the string left to right, each time a left parenthesis is encountered, it is pushed onto the stack. Each time a right parenthesis is encountered the stack is popped. In a balanced expression, the stack should be empty when you run out of letters in the string and not before.

```
/*
 * File: ParenthesisChecker.java
 * Author: Java, Java, Java
 * Description: This program uses a Stack ADT to determine if
 * expressions entered on the command line contained balanced
 * parentheses. The algorithm makes use of the Stack's LIFO
 * behavior.
 *
 * Usage: java ParenthesisChecker
 */
public class ParenthesisChecker {

    /**
     * isBalanced() returns true if its parameter contains
     * balanced parentheses. The string is processed from
     * left to right, char by char. Each time a left parens
     * is found, it is pushed on the stack. Each time a right
     * parens is found, the stack is popped. If an attempt to
     * pop is made on an empty stack, that's an error caused by
     * too many right parens. If the stack is not empty at the
```

```
     * end, the string contains too many left parens. Note how
     * data popped from the stack must be coerced from Object
     * to Node and then from Node to Character.
     * @param s -- the String to be tested
     * @return true if s has balanced parentheses
     */
    public boolean isBalanced(String s) {
        Stack stack = new Stack();
        for(int k = 0; k < s.length(); k++) {
               // Push each left parens
            if (s.charAt(k) == '(')
                stack.push(new Object());
            else if (s.charAt(k) == ')') {
                if (stack.isEmpty())
                    return false; // Too many right parens
                else
                    stack.pop();
            } // if
        } // for
        return (stack.isEmpty()); // Stack should be empty
    } // isBalanced()

    /**
     * main() creates an instance of this class and tests its
     * isBalanced() method. The expression to be tested is
     * with three sample strings of parenthese.
     */
    public static void main (String args[]) {
      ParenthesisChecker app = new ParenthesisChecker();
      String expr = "(()())";
      System.out.println("isBalanced ('" + expr + "') returned " +
                                           app.isBalanced(expr));
      expr = "(()()))";
      System.out.println("isBalanced ('" + expr + "') returned " +
                                           app.isBalanced(expr));
      expr = "((()())";
      System.out.println("isBalanced ('" + expr + "') returned " +
                                           app.isBalanced(expr));
    } // main()
} // ParenthesisChecker
```

14. Design and write a program that uses `Stacks` to determine whether an expression involving both parentheses and square brackets is well formed.

    Answer: This exercise requires that we modify the `isBalanced()` method from the solution to the previous exercise. In this case not only must there be one right parenthesis (or bracket) for each left, right brackets must match up with left brackets, and right parentheses must match up with left parentheses. So in this case, whenever we encounter a left parenthesis (or bracket) we store a right parenthesis (or bracket) on the stack. Then when we encounter a right parenthe-

sis we pop the stack. The top character should match the right parenthesis (or bracket).

```
/*
 * File: ParenthesisChecker2.java
 * Author: Java, Java, Java
 * Description: This program uses a Stack ADT to determine if
 * expressions entered on the command line contained balanced
 * parentheses and brackets.  The algorithm makes use of the
 * Stack's LIFO behavior.
 *
 * Usage: java ParenthesisChecker2
 */
public class ParenthesisChecker2 {

    /**
     * isBalanced() returns true if its parameter contains
     * balanced parentheses. The string is processed from
     * left to right, char by char. Each time a left
     * parens or bracket is found, a right parens or
     * bracket is pushed on the stack. Each time a right
     * parens or bracket is found, the stack is popped.
     * The character popped must match either parens or
     * bracket. If an attempt to pop is made on an empty
     * stack, that's an error caused by too many right
     * parens. If the stack is not empty at the end, the
     * string contains too many left parens. Note how data
     * popped from the stack must be coerced from Object
     * to Node and then from Node to Character.
     * @param s -- the String to be tested
     * @return true if s has balanced parentheses
     */
    public boolean isBalanced(String s) {
        Stack stack = new Stack();
        for (int k = 0; k < s.length(); k++) {
            if (s.charAt(k) == '('      // Push parens
                stack.push(new Character(')'));
            else if (s.charAt(k) == '['  // or bracket
                stack.push(new Character(']'));
            else if (s.charAt(k) == ')') {
                if (stack.isEmpty())
                    return false;
                else {  // Character popped must match
                    Node node = (Node)stack.pop();
                    Character currentChar =
                              (Character)node.getData();
                    if (currentChar.charValue() != ')')
                        return false;
                }
            }
```

```
            else if (s.charAt(k) == ']') {
                if (stack.isEmpty())
                    return false;
                else { // Character popped must match
                    Node node = (Node)stack.pop();
                    Character currentChar =
                             (Character)node.getData();
                    if(currentChar.charValue() != ']')
                    return false;
                } // else
            } // if
        } // isBalanced()
          // Stack should be empty at end
        return (stack.isEmpty());
    } // isBalanced()

    /**
     * main() creates an instance of this class and
     * tests its isBalanced() method. The expression
     * to be tested with three strings of parenthese
     * and brackets.
     */
    public static void main (String args[]) {
        ParenthesisChecker2 app =
                          new ParenthesisChecker2();
      String expr = "([]())";
      System.out.println("isBalanced ('" + expr +
                "') returned " + app.isBalanced(expr));
      expr = "(()()])";
      System.out.println("isBalanced ('" + expr +
                "') returned " + app.isBalanced(expr));
      expr = "(([)(])";
      System.out.println("isBalanced ('" + expr +
                "') returned " + app.isBalanced(expr));
    } // main()
} // ParenthesisChecker2
```

15. Write a program that links two lists together, appending the second list to the tail of the first list.

    Answer: The solution to this exercise is simple: Just insert the head of the second list as the last element in the first list.

```
/*
 * File: ListLinker.java
 * Author: Java, Java, Java
 * Description: This class contains the linkLists()
 * method which links together two lists by appending
 * the second list to the first.
 */
```

```
public class ListLinker {

    /**
     * linkLists() appends list2 to the rear of list1.
     * list2 is destroyed (emptied) in the process.
     * @param list1 -- a list of nodes
     * @param list2 -- a list of nodes
     * @return a reference to the combined list
     */
    public List linkLists(List list1, List list2) {
        while (! list2.isEmpty())
            list1.insertAtRear(list2.removeFirst());
        return list1;
    } // linkLists()

    /**
     * main() creates an instance of this class and
     * tests the linkLists() method.
     */
    public static void main (String args[]) {
        ListLinker app = new ListLinker();
        List a = new List();
        List b = new List();
        a.insertAtRear(new String("Hello"));
        a.insertAtRear(new String("World"));
        b.insertAtRear(new String("Goodbye"));
        b.insertAtRear(new String("Dolly"));
        System.out.println ("A list");
        a.print();
        System.out.println ("B list");
        b.print();
        List c = app.linkLists(a,b);
        System.out.println ("C list");
        c.print();
    } // main()
} // ListLinker
```

16. Design a `Stack` class that uses a `Vector` instead of a linked list to store its elements. This is the way Java's `Stack` class is defined.

    Answer: We extend the `Vector` class, using its `insertElementAt(obj, 0)` and `removeElementAt(0)` methods to implement push and pop. The `main()` method is used to test that the implementation indeed functions like a stack.

```
/*
 * File: Stack.java
 * Author: Java, Java, Java
 * Description: This implementation of a Stack ADT
 * extends the Java Vector class.
```

```
 */

import java.util.*;

public class Stack extends Vector {

    /**
     * Stack() creates an empty Vector by invoking
     * the superclass constructor.
     */
    public Stack() {
        super();
    } // Stack()

    /**
     * push() inserts an object on the top of the
     * stack (location 0)
     * @param Object -- the object inserted
     */
    public void push (Object obj) {
        insertElementAt(obj, 0);
    } // push()

    /**
     * pop() removes and returns the object at the
     * top of the stack.
     * @return the Object at location 0
     */
    public Object pop() {
        Object obj = peek();
        removeElementAt(0);
        return obj;
    } // pop()

    /**
     * peek() returns the top object without
     * removing it
     * @return the Object at location 0
     */
    public Object peek() {
        return elementAt(0);
    } // peek()

    /**
     * main() creates a Stack and tests its
     * behavior by inserting and then removing
     * a string of characters. The result is
     * that the string will be reversed, since
     * the last character inserted will be the
     * first removed. Note how the Character
```

```
     * wrapper class is used to create Character
     * objects.
     */
    public static void main( String argv[] ) {
        Stack stack = new Stack();
        String string =
                "Hello this is a test string";

        System.out.println("String: " + string);
        for (int k = 0; k < string.length(); k++)
          stack.push(new Character( string.charAt(k)));

        Object ob = null;
        String reversed = "";
        while (!stack.isEmpty()) {
            ob  = stack.pop();
            reversed = reversed + ob.toString();
        }
        System.out.println("Reversed String: "
                                      + reversed);
    } // main()

} // Stack
```

17. Design a `Queue` class that uses a `Vector` instead of a linked list to store its elements.

    Answer: The design here is similar to that used in the previous exercise, except that hear we use `insertElementAt()` to insert elements at the rear of the list. The `Vector.size()` method can be used to get the index of the last element in the list.

```
/*
 * File: Queue.java
 * Author: Java, Java, Java
 * Description: This Queue ADT is designed as an
 * extension of the Vector class.
 */

import java.util.*;

public class Queue extends Vector {

    /**
     * Queue() creates an instance by involving the
     * superclass constructor.
     */
    public Queue() {
        super();
    } // Queue()
```

```
    /**
     * enqueue() inserts its parameter at the rear
     * of the vector.
     * @param obj -- the Object to be inserted.
     */
    public void enqueue (Object obj) {
      // This automatically increases the vector's size
      insertElementAt(obj, size());
    } // enqueue()

    /**
     * dequeue() removes and returns the first element
     * of the vector
     * @return the Object at the head of the Queue
     */
    public Object dequeue() {
        Object obj = firstElement();
        removeElement(firstElement());
        return obj;
    } // dequeue()

    /**
     * main() creates a Queue and tests its behavior
     * by inserting and then removing a string of
     * characters. Note how the Character wrapper
     * class is used to create Character objects.
     */
    public static void main(String argv[]) {
        Queue queue = new Queue();
        String string = "Hello this is a test string";
        System.out.println("Inserting: " + string);
        for (int k = 0; k < string.length(); k++)
          queue.enqueue( new Character(string.charAt(k)));

        Object ob = null;
        System.out.print("Dequeuing:");
        while (!queue.isEmpty()) {
            ob  = queue.dequeue();
            System.out.print( ob.toString() );
        } // while
        System.out.println();
    } // main()

} // Queue
```

18. Write a program that uses the `List<E>` and `LinkedList<E>` classes to store a list of `Student` records, using a variation of the `Student` class defined in Chapter 11. Write a method to calculate the mean grade point average for all students in the list.

Answer: This `StudentList` class is modeled after the `testList()` method in Figure 16.31 in the text.

```
/*
 * File: StudentList.java
 * Author: Java, Java, Java
 * Description: This version of StudentList implements a list
 * of student records using generic types with the List<E> interface
 * and LinkedList<E> class. Each Student record consists of a name,
 * year and gpa field. (The Student class defined in Chapter 11
 * is used. The only change to that class is to add a getGPA()
 * access method.
 *
 * Note that a Java 5.0 compiler is needed to compile source
 * code that uses generic types.
 */

import java.util.*;  // For  List<E> and Linked<E>

public class StudentList {

    /**
     * calcAvgGPA(stList) calculates the average GPA of all
     * students in the list.
     * @return a double giving the average GPA
     */
    public static double calcAvgGPA(List<Student> stList) {
        int counter = 0;
        double sum = 0;
        for (Student st : stList) {
            sum += st.getGPA();
            counter++;
        } // for
        return sum / counter;
    } // calcAvgGPA()

    /**
     * main() creates a linked list of students and then tests
     *   the calcGPA() method.
     */
    public static void main( String argv[] ) {
                    // Create list and insert heterogeneous nodes
        List<Student> stList = new LinkedList<Student>();
        stList.add(new Student("adam", 2000, 12.9));
        stList.add(new Student("baker", 2000, 10.6));
        stList.add(new Student("curtis", 2000, 8.2));
        stList.add(new Student("dietz", 2000, 12.5));
        stList.add(new Student("eliot", 2000, 6.5));

                    // Print the list
```

```
        System.out.println("Printing generic type Student List ");
        for (Student st : stList)
            System.out.println(st);
        System.out.println(" The average GPA for these students is "
                           + StudentList.calcAvgGPA(stList));
    } // main()
} // StudentList
```

19. Write an implementation of the `insert()` method of the `PhoneTree` class described at the end of this chapter.

    Answer: This method uses the `insert()` method of the `PhoneTreeNode` class that is defined in the next exercise.

```
/**
 * insert(String nam, String pho) inserts a PhoneTreeNode with
 * data values nam and pho into this PhoneTree object.
 * @param nam is string representing a name for a phone
 *   directory entry.
 * @param pho is string representing a phone number.
 */
public void insert(String nam, String pho) {
      // If root == null create a new PhoneTreeNode
    if (root == null)
        root = new PhoneTreeNode(nam, pho);
    else // Have the root do the inserting.
        root.insert(nam, pho);
} // insert()
```

20. rite an implementation of the `insert()` method of the `PhoneTreeNode` class described at the end of this chapter.

    Answer: This method is modeled after the `contains` method of the `PhoneTreeNode` class at the end of this chapter in the text. We assume that the binary tree cannot contain more than one node with the same name. The exercise does not specify what should be done if nam is equal to the name variable of this object. We will change the phone number in this case. Other implementations are possible.

```
/**
 * insert(String nam, String pho) inserts a PhoneTreeNode
 * with data values (nam,pho) into this PhoneTreeNode object.
 * @param nam is string representing a name for a phone
 *   directory entry.
 * @param pho is string representing a phone number.
 */
public void insert(String nam, String pho) {
      // If name == nam,  change the phone number
    if (name.equals(nam))
        phone = pho;
    else if (name.compareTo(nam) < 0){
       // name < nam so insert (nam,pho) into right node
```

```
            if (right == null)
                right = new PhoneTreeNode(nam, pho);
            else
                right.insert(nam, pho);
        } else // name > nam so insert (nam,pho) into left node
            if (left == null)
                left = new PhoneTreeNode(nam, pho);
            else
                left.insert(nam, pho);
    } // insert()
```

21. **Challenge:** Design a `List` class, similar in functionality to the one we designed in this chapter, that uses an *array* to store the list's elements. Set it up so that the middle of the array is where the first element is put. That way you can still insert at both the front and rear of the list. One limitation of this approach is that unlike a linked list, an array has a fixed size. Allow the user to set the initial size of the array in a constructor, but if the array becomes full, don't allow any further insertions.

    Answer: The most difficult part of this exercise is designing a protocol to define when the list is empty and when it is full. This must be done in a consistent manner. The design we use here is described in the comments provided for the `List` class. An important feature of our design is that is starts inserting elements at the array's midpoint. This allows insertions at both the front and rear, as we had in our linked list implementation. Thus we get a list with the same functionality as in the linked-list version, but with a completely different implementation. This is a good example of *information hiding*.

```
/*
 * File: List.java
 * Author: Java, Java, Java
 * Description: This class implements a list using a
 * static array to store the list's elements. Integer
 * indices are used to determine the front and rear
 * of the list. Initially the front and rear start out
 * equal and set (roughly) to the midpoint of the array.
 * Insertions at the front are made at frontIndex.
 * Insertions at the rear are made at rearIndex + 1.
 * Removals from the front occur at frontIndex +1.
 * Removals at the rear occur at rearIndex. The
 * rearIndex is bound by the array's length. The
 * frontIndex is bound by 0. An empty list is defined
 * as frontIndex == rearIndex, which of course is the
 * initial state. The insertion and removal algorithms
 * must insure that this empty list condition holds
 * when the last element is removed regardless of
 * where it is removed from.

 * One limitation of using an array for this purpose
```

```
 * is that it is static. Plus there's a good chance
 * that array locations will go wasted. For example,
 * if we use the list as a queue, where insertions
 * always occur at the rear, then only half the
 * array elements will be used.
 */
public class List {

    private Object data[]; // The array of data
       // Initial midpoint of data
    private int startIndex;
       // Index of NEXT insertion or removal
    private int frontIndex;
       // Index of NEXT insertion or removal
    private int rearIndex;
       // NOTE: frontIndex <= rearIndex

     /**
      * List() initializes the list by creating an
      * array of initialSize elements and setting
      * the initial values of the indices.
      * @param initialSize -- an int giving the
      *   array's initial size
      */
    public List(int initialSize) {
      data = new Object[initialSize];
        // Set start to midpoint
      int startIndex = (initialSize / 2) - 1;
        // Empty: rearIndex == frontIndex
      frontIndex = startIndex;
      rearIndex = startIndex;
    } // List()

    /**
     * isEmpty() returns true when rearIndex ==
     *     frontIndex.
     */
    public boolean isEmpty() {
        return frontIndex == rearIndex;
    } // isEmpty()

    /**
     * print() prints each element of the list.
     * Note that frontIndex +1 gives the
     * location of first element in a nonempty
     * list
     * PRE: frontIndex <= rearIndex
     */
    public void print() {
        if (isEmpty()) {
```

```
            System.out.println("List is empty");
            return;
        }
        for (int j = frontIndex+1;
                            j <= rearIndex; j++)
            System.out.println(data[j].toString()
                                        + " ");
    } // print()

    /**
     * insertAtFront() inserts an object at
     * frontIndex as long as frontIndex is not
     * less than 0 (indicating a full list)
     * @param obj -- the Object to be inserted
     * @return true iff insertion was successful.
     */
    public boolean insertAtFront(Object obj) {
        if(frontIndex >= 0) {
            data[frontIndex] = obj;
            frontIndex--;
            return true;
        } // if
        return false;
    } // insertAtFront()

    /**
     * insertAtRear() inserts an object at
     * rearIndex +1 as long as the list is not
     * full (rearIndex >= length-1
     * @param obj -- the object to be inserted
     * @return true iff the insertion was successful
     */
    public boolean insertAtRear(Object obj) {
        if (rearIndex < data.length - 1) {
            rearIndex++;
            data[rearIndex] = obj;
            return true;
        }
        return false;
    } // insertAtRear()

    /**
     * removeFirst() removes the element at
     * frontIndex+1 provided the list is not empty
     * @return the Object removed or null
     */
    public Object removeFirst() {
        Object obj = null;
        if (!isEmpty()) {
            obj = data[frontIndex + 1];
```

```
            frontIndex++;
        }
        return obj;
    } // removeFirst()

    /**
     * removeLast() removes the object at rearIndex,
     * provided the list is not empty
     * @return the Object removed or null
     */
    public Object removeLast() {
        Object obj = null;
        if (!isEmpty()) {
            obj = data[rearIndex];
            rearIndex--;
        }
        return obj;
    } // removeLast()

    /**
     * main() creates a list of different types
     * of objects and tests this class's methods.
     * This example shows that a array of 4 slots
     * can store 4 different objects.
     */
    public static void main( String argv[] ) {
      // Create list and insert heterogeneous objects
        List list = new List(4);
        list.print();
        list.insertAtFront(new Character('h'));
        list.insertAtFront(new Integer(8647));
        list.insertAtRear(new Double(8647.05));
        list.insertAtRear(new String("Hello World"));

          // Print the list
        System.out.println("Generic List");
        list.print();
          // Remove objects and print the resulting list
        Object o;
        o = list.removeFirst();
        System.out.println(" Removed " + o.toString());
        System.out.println("Generic List:");
        list.print();
        o = list.removeFirst();
        System.out.println(" Removed " + o.toString());
        System.out.println("Generic List:");
        list.print();
        o = list.removeFirst();
        System.out.println(" Removed " + o.toString());
        System.out.println("Generic List:");
```

```
        list.print();
           // Insert another object
        list.insertAtFront(new Integer(1234));

        o = list.removeFirst();
        System.out.println(" Removed " + o.toString());
        System.out.println("Generic List:");
        list.print();

        o = list.removeFirst();
        System.out.println(" Removed " + o.toString());
        System.out.println("Generic List:");
        list.print();
    } // main()

} // List
```

22. **Challenge:** Add a method to the program in the previous exercise that lets the user increase the size of the array used to store the list. You may want to review the In the Laboratory section of Chapter 11.

    Answer: The `setSize()` method increases the size of the array and then copies all of the existing elements into the new array. In this case we extend the array by growing it out from the middle. This takes some additional work, but it is an important design consideration given the fact that in our original design the midpoint of an empty array served as the location of the first insertion.

```
    /**
     * setSize() increases the size of the array by
     * creating a new array and copying the elements
     * from the old array to the new. Note that old
     * elements are copied into the new array from
     * the middle to the back and from the middle to
     * the front, thereby allowing it to expand in
     * both directions.
     * @param size -- an int giving the new size
     * @return true iff the size was increased
     */
    public boolean setSize(int size) {
          // Don't shrink the list
        if(size <= data.length)
            return false;
        Object newData[] = new Object[size];
        int newMid = (size / 2) - 1;

        if (isEmpty()) { // Special case: empty list
            data = newData;
            startIndex = frontIndex = rearIndex = newMid;
            return true;
```

```
        }

        int k = newMid;  // Copy back half
        for (int j = startIndex; j < data.length ; j++) {
            newData[k] = data[j];
            k++;
        }
        k = newMid - 1;    // Copy front half
        for (int j = startIndex - 1; j >= 0; j--) {
            newData[k] = data[j];
            k--;
        }
        data = newData;
        startIndex = newMid;
        return true;
    } // setSize()
```

23. **Challenge:** Recursion is a useful technique for list processing. Write recursive versions of the `print()` method and the lookup by name method for the `PhoneList`. (*Hint*: The base case in processing a list is the empty list. The recursive case should handle the head of the list and then recurse on the tail of the list. The tail of the list is everything but the first element.)

    Answer: Treating the list as a head and a tail, we can process the list recursively by repeatedly passing the tail of the original list to our processing method. The methods defined here illustate the details of how this is done.

```
/**
 * getPhoneRecursive() recursively searches for a
 * phone number
 * @param name -- a String giving name of person
 *   whose number is desired
 * @param node -- a reference to the current node
 */
private String getPhoneRecursive(String name,
                            PhoneListNode node) {
    if (node == null)                 // Base case
        return ("Sorry. No entry for " + name);
    else if (node.getName().equals(name)) // Base case
        return node.getData();
    else return getPhoneRecursive(name, node.getNext());
                                      // Recursive case
} // getPhoneRecursive()

/**
 * getPhone() returns the phone number given a
 * person's name
 * @param name -- a String giving a person's name
 * Algorithm: Traverse the list until the node
```

```
     * containing the  given name is found. Return
     * the corresponding phone number.
     */
    public String getPhone(String name) {
        if (isEmpty())        // Case 1: empty list
            return "Phone list is empty";
        else
            return getPhoneRecursive(name, head);
    } // getPhone()

    /**
     * recursivePrint() recursively prints the list
     * given a reference to the current node
     * @param current -- a reference to a node.
     */
    public void recursivePrint(PhoneListNode current) {
        if (current != null) {
            System.out.println(current.toString());
            recursivePrint(current.getNext());
        } // if
    } // recursivePrint()

    /**
     *  print() prints the entire by calling
     *  recursivePrint()
     */
    public void print() {
        if (isEmpty())
            System.out.println("Phone list is empty");
        else
            recursivePrint(head);
    } // print()
```

24. **Challenge:** Design an `OrderedList` class. An ordered list is one that keeps its elements in order. For example, if it's an ordered list of integers, then the first integer is less than or equal to the second, the second is less than or equal to the third, and so on. If it's an ordered list of employees, then perhaps the employees are stored in order according to their social security numbers. The `OrderedList` class should contain an `insert(Object o)` method that inserts its object in the proper order. One major challenge in this project is designing your class so that it will work for any kind of object. (Hint: Define an `Orderable` interface that defines an abstract `precedes()` method. Then define a subclass of `Node` that implements `Orderable`. This will let you compare any two `Node`s to see which one comes before the other.)

    Answer: Our design to this solution follows the design of the polymorphic sorting algorithm described in Chapter 9. It defines an `Orderable` interface, which contains an abstract `precedes()` method. Any objects to be stored in an ordered list must implement the `Orderable` interface. For example, an

`OrderedInt` (defined below) is an integer that defines the `Orderable` interface. An `OrderedList` is a list that uses the `precedes()` method to put its elements in the correct order when they are inserted into the list. So it always maintains the list in the proper order. See the `insert()` method for the details. As evident there, what makes this type of polymorphism somewhat difficult is the awkward syntax needed to compare two objects stored in the list using the `precedes()` method.

```
/*
 * File: Orderable.java
 * Author: Java, Java, Java
 * Description: An abstract interface that defines
 * the precedes() method.
 */

public interface Orderable {
    public boolean precedes(Object obj);
} // Orderable

/*
 * File: OrderedInt.java
 * Author: Java, Java, Java
 * Description: An OrderedInt is an integer that
 * implements the Orderable interface. This requires
 * an implementation of the precedes method for
 * integer data.
 *
 * Note that we use an Integer instance variable to
 * store the number. It is not possible to subclass
 * the Integer class because it is declared final.
 */

public class OrderedInt implements Orderable {

    private Integer number;

    /**
     * OrderedInt() creates an instance with its
     * parameter value
     * @param n -- an int that gives the value for
     * this instance
     */
    public OrderedInt (int n) {
        number = new Integer(n);
    } // OrderedInt()

    /**
     * intValue() returns the integer value of this
     * instance
```

```
     * @return an int representing this instance's
     *   value
     */
    public int intValue() {
        return number.intValue();
    }

    /**
     * precedes() is part of the Orderable interface.
     * It defines what it means for one OrderedInt
     * to precede another.
     * @param obj -- Object to compare to this instance
     * @return true iff this instance is less than
     *   the parameter
     */
    public boolean precedes(Object obj)  {
        return number.intValue() <
                        ((OrderedInt)obj).intValue();
    } // precedes()
} // OrderedInt

/*
 * File: OrderedList.java
 * Author: Java, Java, Java
 * Description: This subclass of Vector implements
 * an ordered list. For this example only OrderedInt's
 * (integer values) may be stored in the list, because
 * that's the parameter of the insert() methods. To
 * generalize this, create insert() methods for the
 * other data types.
 *
 * What makes this work is that an OrderedInt implements
 * the the precedes method, which defines what it means
 * for one OrderedInt to preceded another.
 */

import java.util.Vector;

public class OrderedList extends Vector {

    /**
     * OrderList() constructor invokes the superclass
     * constructor
     */
    public OrderedList () {
        super();
    }

    /**
     * print() prints the datum of each element of the
```

```
 * list. Since the list stores Nodes, it is
 * necessary to get the data out  of the node and
 * then to convert it to OrderedInt and then to
 * an integer value.
 */
public void print() {
    for (int k = 0; k < size(); k++) {
        Node node = (Node)elementAt(k);  // Get Node
        OrderedInt ordInt =
             (OrderedInt)node.getData(); // Get data
        int num = ordInt.intValue();     // Get int
        System.out.println(num);
    }
} // print()

/**
 * insert() inserts an OrderedInt into the list in its
 * proper order,  where proper order is determined by
 * the precedes() method.  The algorithm traverses the
 * list until it finds the proper insertion point. Note
 * again that it is necessary to get the OrderedInt out
 * of the Node before precedes() can be invoked.
 * @param obj -- the OrderedInt to be inserted
 */
public void insert(OrderedInt obj) {
    int currentIndex = 0;
    boolean foundPosition = false;
    Node node = null;
    Node newNode = new Node(obj);
    int j = 0;
    while (! foundPosition && j < size()) {
        Node temp = (Node) elementAt(j);
        if (((OrderedInt)newNode.getData()).precedes(
                        (OrderedInt)temp.getData()))
            foundPosition = true;
        else j++;
    }
    insertElementAt(newNode, j);
} // insert()

/**
 * main() creates an instance of this class and then
 * tests it by inserting 20 random integer values into
 * the list.
 */
public static void main(String argv[]) {
    OrderedList list = new OrderedList();
    System.out.print("Inserting: ");
    for (int k = 0; k < 20; k++)  {
        int n = (int)(Math.random() * 25);
```

```
            System.out.print(" " + n);
            list.insert( new OrderedInt(n));
        }
        System.out.println();

        System.out.println("The resulting list");
        list.print();
    } // main()

} // OrderedList
```