2.21 What does the following code print?

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hi");
        System.out.println("Hi");
        System.out.println("Hi");
        System.out.println("Hi");
    }
}
```

2.22 What does the following code print?

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hi");
        System.out.println("Hi");
        System.out.println("Hi");
        System.out.println("Hi");
    }
}
```

2.23 What does the following code print?

```java
public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println("Hello, world!");
        System.out.println("Hello, world!");
        System.out.println("Hello, world!");
    }
}
```

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

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

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

2.27 Write an application that displays a checkerboard pattern, as follows:

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

2.28 Here's a peek ahead. In this chapter, you have learned about integers and the type int. Java can also represent floating-point numbers that contain decimal points, such as 3.14159. Write an application that inputs from the user the radius of a circle as an integer and prints the circle's diameter, circumference and area using the floating-point value 3.14159 for \( \pi \). Use the techniques shown in Fig. 2.7. [Note: You may also use the predefined constant \( \pi \) in the value of \( \pi \). This constant is more precise than the value 3.14159. Class Math is defined in package java.lang. Classes in that package are imported automatically, so you do not need to import class Math to use it.] Use the following formulas (\( r \) is the radius):

\[
\begin{align*}
diameter &= 2r \\
circumference &= 2\pi r \\
area &= \pi r^2
\end{align*}
\]

Do not store the results of each calculation in a variable. Rather, specify each calculation as the value that will be output in a `System.out.println` statement. Note that the values produced by the circumference and area calculations are floating-point numbers. Such values can be output with the format specifier `%f` in a `System.out.println` statement. You will learn more about floating-point numbers in Chapter 3.
Exercises

3.5 What is the purpose of keyword new? Explain what happens when this keyword is used in an application.

3.6 What is a default constructor? How are an object's instance variables initialized if a class has only a default constructor?

3.7 Explain the purpose of an instance variable.

3.8 Most classes need to be imported before they can be used in an application. Why is every application allowed to use classes System and String without first importing them?

3.9 Explain how a program could use class Scanner without importing the class from package java.util.

3.10 Explain why a class might provide a set method and a get method for an instance variable.

3.11 Modify class Grade (Fig. 3.8) as follows:
   a) Include a second String instance variable that represents the name of the course’s instructor.
   b) Provide a set method to change the instructor’s name and a get method to retrieve it.
   c) Modify the constructor to specify two parameters—one for the course name and one for the instructor’s name.
   d) Modify method displayMessage such that it first outputs the welcome message and course name, then outputs “This course is presented by: ” followed by the instructor’s name.

Use your modified class in a test application that demonstrates the class’s new capabilities.

3.12 Modify class Account (Fig. 3.12) to provide a method called debit that withdraws money from an account. Ensure that the debit amount does not exceed the account’s balance. If it does, the balance should be left unchanged and the method should print a message indicating “Debit amount exceeded account balance.” Modify class AccountTest (Fig. 3.14) to test method debit.

3.13 Create a class called Invoice that a hardware store might use to represent an invoice for an item sold at the store. An Invoice should include four pieces of information as instance variables—a part number (type String), a part description (type String), a quantity of the item being purchased (type int) and a price per item (type Double). Your class should have a constructor that initializes the four instance variables. Provide a set and a get method for each instance variable. In addition, provide a method named getInvoiceAmount that calculates the invoice amount (i.e., multiplies the
quantity by the price per item), then return the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named InvoiceTest that demonstrates class Invoice's capabilities.

3.14 Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee's capabilities. Create two Employee objects and display each object's yearly salary. Then give each Employee a 10% raise and display each Employee's yearly salary again.

3.15 Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method display that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date's capabilities.
gram. We have eliminated the indentation from the given code to make the problem more challenging. [Note: It is possible that no modification is necessary for some of the parts.]

```
if (y == 3)
  System.out.println("output 1");

else
  System.out.println("output 2");
System.out.println("output 3");
```

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

```
output 2
```

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

```
output 1
```

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

```
output 3
```

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

```
output 3
```

4.29 Write an application that prompts the user to enter the side of the side of a square, then displays a hollow square of that size made of asterisks. Your program should work for squares of all side lengths between 1 and 20.

4.30 (Palindrome) A palindrome is a sequence of characters that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 65556, and 11881. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message and allow the user to enter a new value.

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

4.32 Write an application that uses only the output statements

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

to display the checkerboard pattern that follows. Note that a `System.out.println` method call with no arguments causes the program to output a single newline character. [Hint: Repetition statements are required.]
4.33 Write an application that prints the multiples of the integer 2—namely, 2, 4, 8, infinite (loop). What happens?

4.34 What is wrong with the sum of \( x \) and \( y \)?

```java
System.out.print();
```

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

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

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

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

\[
 n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1 \quad \text{(for values of } n \text{ greater than or equal to 1)}
\]

and

\[
 0! = 1 \quad \text{(for } n = 0)\]

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

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

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

\[
 e = 1 + \frac{1}{1} + \frac{1}{1} + \frac{1}{3} + \ldots
\]

c) Write an application that computes the value of \( e^2 \) by using the formula

\[
 e^2 = 1 + \frac{2}{1} + \frac{4}{2} + \frac{6}{3} + \ldots
\]
4.29 Write an application that simulates the "Toss a Coin" menu item. Display the results. The program should return false for tails and true for heads. The coin should appear on each side of the coin should appear.

4.30 Computers are really slow at multiplying large integers. The program should time how much it takes to multiply two large integers. The program should then display the answer, and then display the time it took to do that operation. The program should then ask if the user wants to do it again. The program should start all over again if the user wants to do it again. After the user enters the number 2, the program should stop and display the final time.

4.31 The use of computers in education is referred to as computer-assisted instruction (CAI). One problem that develops in CAI environments is student fatigue. This problem can be eliminated by varying the computer's responses to hold the student's attention. The program of Exercise 6.30 should keep track of the number of correct answers and incorrect answers. If the number of correct answers is greater than the number of incorrect answers, display "You're doing fine!" If the number of correct answers is less than the number of incorrect answers, display "You're doing better!"

4.32 More sophisticated computer-assisted instruction systems monitor the student's performance over a period of time. The decision to begin a new topic is often based on the student's success with previous topics. Modify the program of Exercise 6.31 to count the number of correct and incorrect responses typed by the student. After the student types 10 answers, your program should calculate the percentage of correct responses. If the percentage is lower than 75%, display "Please see your instructor for extra help and reset the program so another student can try it."

4.33 Write an application that plays the "guess the number" game as follows: Your program chooses the number to be guessed by selecting a random integer in the range 1 to 1000. The application displays the prompt "Guess a number between 1 and 1000." The player inputs a first guess. If the player's guess is incorrect, your program should display "Too high." Try again. If the player's guess is incorrect, your program should display "Too low." Try again. To help the player "zero in" on the correct answer, the program should prompt the user for the next guess. When the user enters the correct answer, display Congratulations. You guessed the number! and allow the user to choose whether to play again. [Note: The guessing technique employed in this problem is similar to a binary search, which is discussed in Chapter 16, Searching and Sorting.]
6.34 Modify the program of Exercise 6.33 to count the number of guesses the player makes. If the number is 10 or fewer, display "Either you know the secret or you got lucky!" If the player guesses the number in 10 tries, display "You know the secret!" If the player makes more than 10 guesses, display "You should be able to do better! Why should it take no more than 10 guesses? Well, with each "good guess," the player should be able to eliminate half of the numbers, then half of the remaining numbers, and so on.

6.35 Exercise 6.30 through Exercise 6.32 developed a computer-assisted instruction program to teach an elementary school student multiplication. Perform the following enhancements:

(a) Modify the program to allow the user to enter a school grade-level capability. A grade level of 1 means that the program should use only single-digit numbers in the problems; a grade level of 2 means that the program should use numbers as large as two digits, and so on.

(b) Modify the program to allow the user to pick the type of arithmetic problems he or she wishes to study. An option of 1 means addition problems only, 2 means subtraction problems only, 3 means multiplication problems only, 4 means division problems only and 5 means a random mixture of problems of all these types.

6.36 Write a method distance to calculate the distance between two points \((x_1, y_1)\) and \((x_2, y_2)\). All numbers and return values should be of type double. Incorporate this method into an application that enables the user to enter the coordinates of the points.

6.37 Modify the craps program of Fig. 6.9 to allow wagering. Initialize variable bankBalance to 10000 dollars. Prompt the player to enter a wager. Check that wager is less than or equal to bankBalance, and if it is not, have the user reenter wager until a valid wager is entered. After a correct wager is entered, run one game of craps. If the player wins, increase bankBalance by wager and display the new bankBalance. If the player loses, decrease bankBalance by wager, display the new bankBalance, check whether bankBalance has become zero and, if so, display the message "Sorry, you bust!". As the game progresses, display various messages to create some "chatter," such as "oh, you're going for broke, huh?" or "An.chron, take a chance!" or "You're up big, now's the time to cash in your chips!". Implement the "chatter" as a separate method that randomly chooses the saying to display.

6.38 Write an application that displays a table of the binary, octal, and hexadecimal equivalents of the decimal numbers in the range 1 through 256. If you are not familiar with these number systems, read Appendix F first.
g) $800.00–999
h) $900.00–999
i) $1,000 and over

Summarize the results in tabular format.

7.11 Write statements that perform the following ones:
   a) Set the 10 elements of integer array counts to
   b) Add one to each of the 15 elements of integer
   c) Display the five values of integer array fastest.

7.12 (Duplicate Elimination) Use a one-dimensional array in an application that inputs five numbers, each between 1 and 10, display it only if it is not a duplicate of a number already input, which all five numbers are different. Use the smallest possible set of unique values input after the user enters

7.13 Label the elements of three by five two-dimension which they are set to zero by the following program segment:

    ```c
    for (int row = 0; row < sales.length; row++)
        for (int col = 0; col < sales[row].length; col++)
            sales[row][col] = 0;
    ```

7.14 Write an application that calculates the product of a series of integers that are passed to method product using a variable-length argument list. Test your method with several calls, each with a different number of arguments.

7.15 Rewrite Fig. 7.2 so that the size of the array is specified by the first command-line argument. If no command-line argument is supplied, use 10 as the default size of the array.

7.16 Write an application that uses an enhanced for statement to sum the double values passed by the command-line arguments. [Hint: Use the static method parseDouble of class Double to convert a string to a double value.]

7.17 (Dice rolling) Write an application to simulate the rolling of two dice. The application should use an object of class Random once to roll the first die and again to roll the second die. The sum of the two values should then be calculated. Each die can show an integer value from 1 to 6, so the sum of the values will vary from 2 to 12, with 7 being the most frequent sum, and 2 and 12 the least frequent. Figure 7.30 shows the 36 possible combinations of the two dice. Your application should roll the dice 36,000 times. Use a one-dimensional array to tally the number of times each possible sum appears. Display the results in tabular format. Determine whether the totals are reasonable (e.g., there are six ways to roll a 7, so approximately one-sixth of the rolls should be 7).

7.18 (Game of Craps) Write an application that runs 1000 games of craps (Fig. 6.9) and answers the following questions:
   a) How many games are won on the first roll, second roll, ...., twentieth roll?
   b) How many games are lost on the first roll, second roll, ..., twentieth roll?
   c) What are the chances of winning at craps? [Note: You should discover that craps is one of the fairest casino games. What do you suppose this means?]
   d) What is the average length of a game of craps?
   e) Do the chances of winning improve with the length of the game?
table of the number of tours of each length, and display this table when the first full tour is found. How many tours did your application attempt before producing a full tour? How much time did it take?

d) Compare the brute-force version of the Knight's Tour with the accessibility heuristic version. Which required a more careful study of the problem? Which algorithm was more difficult to develop? Which required more computer power? Could we be certain (in advance) of obtaining a full tour with the accessibility heuristic approach? Could we be certain (in advance) of obtaining a full tour with the brute-force approach? Argue the pros and cons of brute-force problem solving in general.

7.24 (Eight Queens) Another popular chess puzzle is the Eight Queens problem, which asks the following: Is it possible to place eight queens on an empty chessboard so that no queen is attacking any other (i.e., no two queens are in the same row, in the same column or along the same diagonal)? Use the thinking developed in Exercise 7.22 to formulate a heuristic for solving the Eight Queens problem. Run your application. [Hint: It is possible to assign a value to each square of the chessboard to indicate how many squares of an empty chessboard are “eliminated” if a queen is placed in that square. Each of the corners would be assigned the value 22, as demonstrated by Fig. 7.33. Once these “elimination numbers” are placed in all 64 squares, an appropriate heuristic might be as follows: Place the next queen in the square with the smallest elimination number. Why is this strategy intuitively appealing?]

7.25 (Eight Queens: Brute-Force Approaches) In this exercise, you will develop several brute-force approaches to solving the Eight Queens problem introduced in Exercise 7.24.

a) Use the random brute-force technique developed in Exercise 7.23 to solve the Eight Queens problem.

b) Use an exhaustive technique (i.e., try all possible combinations of eight queens on the chessboard) to solve the Eight Queens problem.

c) Why might the exhaustive brute-force approach not be appropriate for solving the Knight's Tour problem?

d) Compare and contrast the random brute-force and exhaustive brute-force approaches.

7.26 (Knight's Tour: Closed-Tour Test) In the Knight's Tour (Exercise 7.22), a full tour occurs when the knight makes 64 moves, touching each square of the chessboard once and only once. A closed tour occurs when the 64th move is one move away from the square in which the knight started the tour. Modify the application you wrote in Exercise 7.22 to test for a closed tour if a full tour has occurred.

Fig. 7.33 | The 22 squares eliminated by placing a queen in the upper left corner.
Begin the race by displaying

```
BANG       !!
AND THEY'RE OFF!!
```

Then, for each tick of the clock (i.e., each repetition of a loop), display a 70-position line showing the letter T in the position of the tortoise and the letter H in the position of the hare. Occasionally, the contenders will land on the same square. In this case, the tortoise bites the hare, and your application should display OUCH!!! beginning at that position. All output positions other than the T, the H or the OUCH!!! (in case of a tie) should be blank.

After each line is displayed, test for whether either animal has reached or passed square 70. If so, display the winner and terminate the simulation. If the tortoise wins, display TORTOISE WINS!! YAY!!! If the hare wins, display HARE WINS. Yuck. If both animals win on the same tick of the clock, you may want to favor the tortoise (the "underdog"), or you may want to display IT'S A TIE. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you are ready to run your application, assemble a group of fans to watch the race. You'll be amazed at how involved your audience gets!

Later in the book, we introduce a number of Java capabilities, such as graphics, images, animation, sound, and multithreading. As you study those features, you might enjoy enhancing your tortoise-and-hare contest simulation.

7.29 (Fibonacci Series) The Fibonacci series

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

begins with the terms 0 and 1 and has the property that each succeeding term is the sum of the two preceding terms.

a) Write a method fibonacci(n) that calculates the nth Fibonacci number. Incorporate this method into an application that enables the user to enter the value of n.

b) Determine the largest Fibonacci number that can be displayed on your system.

c) Modify the application you wrote in part (a) to use double instead of int to calculate and return Fibonacci numbers, and use this modified application to repeat part (b).

Exercises 7.30—7.33 are reasonably challenging. Once you have done these problems, you ought to be able to implement most popular card games easily.

7.30 (Card Shuffling and Dealing) Modify the application of Fig. 7.11 to deal a five-card poker hand. Then modify class handCards of Fig. 7.10 to include methods that determine whether a hand contains:

a) a pair

b) two pairs

c) three of a kind (e.g., three jacks)

d) four of a kind (e.g., four aces)

e) a flush (i.e., all five cards of the same suit)

f) a straight (i.e., five cards of consecutive face values)

g) a full house (i.e., two cards of one face value and three cards of another face value)

[Hint: Add methods getFace and getSuit to class Card of Fig. 7.9.]

7.31 (Card Shuffling and Dealing) Use the methods developed in Exercise 7.30 to write an application that deals two five-card poker hands, evaluates each hand, and determines which is better.

7.32 (Card Shuffling and Dealing) Modify the application developed in Exercise 7.31 so that it can simulate the dealer. The dealer's five-card hand is dealt "face down," so the player cannot see it. The application should then evaluate the dealer's hand, and, based on the quality of the hand, the
8.14 (Enhanced Rectangle Class) Create a more sophisticated Rectangle class than the one you created in Exercise 8.4. This class stores only the Cartesian coordinates of the four corners of the rectangle. The constructor calls a set method that accepts four sets of coordinates and verifies that each of these is in the first quadrant with no single x- or y-coordinate larger than 20.0. The set method also verifies that the supplied coordinates specify a rectangle. Provide methods to calculate the length, width, perimeter, and area. The length is the larger of the two dimensions. Include a predicate method isSquare which determines whether the rectangle is a square. Write a program to test class Rectangle.

8.15 (Set of Integers) Create class IntegerSet. Each IntegerSet object can hold integers in the range 0–100. The set is represented by an array of booleans. Array element a[i] is true if integer i is in the set. Array element a[j] is false if integer j is not in the set. The no-argument constructor initializes the Java array to the "empty set" (i.e., a set whose array representation contains all false values).

Provide the following methods: Method union creates a third set that is the set-theoretic union of two existing sets (i.e., an element of the third set's array is set to true if that element is true in either or both of the existing sets—otherwise, the element of the third set is set to false). Method intersection creates a third set which is the set-theoretic intersection of two existing sets (i.e., an element of the third set's array is set to true if that element is false in either or both of the existing sets—otherwise, the element of the third set is set to false). Method insertElement inserts a new integer k into a set (by setting a[k] to true). Method deleteElement deletes integer k (by setting a[k] to false). Method toString returns a string containing a set as a list of numbers separated by spaces. Include only those elements that are present in the set. Use --- to represent an empty set. Method isEqual determines whether two sets are equal. Write a program to test class IntegerSet. Instantiate several IntegerSet objects. Test that all your methods work properly.

8.16 (Date Class) Create a class named Date with the following capabilities:

a) Output the date in multiple formats, such as:

MM/DD/YYYY
June 14, 1982
DD YYYY

b) Use overloaded constructors to create Date objects initialized with dates of the forms in part (a). In the first case the constructor should receive three integer values. In the second case it should receive a string and two integer values. In the third case it should receive two integer values, the first of which represents the day number in the year.

[Hint: To convert a string representation of the month to a numeric value, compare strings using the equals method. For example, if s1 and s2 are strings, the method call s1.equals(s2) returns true if the strings are identical and otherwise returns false.]

8.17 (Rational Numbers) Create a class called Rational for performing arithmetic with fractions. Write a program to test your class. Use integer variables to represent the private instance variables of the class—the numerator and the denominator. Provide a constructor that enables an object of this class to be initialized when it is declared. The constructor should store the fraction in reduced form. The fraction

\[ \frac{2}{4} \]

is equivalent to \( \frac{1}{2} \) and would be stored in the object as 1 in the numerator and 2 in the denominator. Provide a no-argument constructor with default values in case no initializers are provided. Provide public methods that perform each of the following operations:
a) Add two rational numbers: The result of the addition should be stored in reduced form.
b) Subtract two rational numbers: The result of the subtraction should be stored in reduced form.
c) Multiply two rational numbers: The result of the multiplication should be stored in reduced form.
d) Divide two rational numbers: The result of the division should be stored in reduced form.
e) Print rational numbers in the form \( \frac{a}{b} \), where \( a \) is the numerator and \( b \) is the denominator.
f) Print rational numbers in floating point format. (Consider providing formatting capabilities that enable the user of the class to specify the number of digits of precision to the right of the decimal point.)

8.18 \((\text{Huge Integer Class})\) Create a class HugeInteger which uses a 40-element array of digits to store integers as large as 40 digits each. Provide methods input, output, add, and subtract. For comparing hugeInteger objects, provide the following methods: isEqualTo, isNotEqualTo, isGreaterThan, isLessThan, isGreaterThanOrEqualTo and isLessThanOrEqualTo. Each of these is a predicate method that returns true if the relationship holds between the two hugeInteger objects and returns false if the relationship does not hold. Provide a predicate method isZero. If you feel ambitious, also provide methods multiply, divide and remainder. [Note: Primitive boolean values can be output as the word “true” or the word “false” with format specifier %b.]

8.19 \((\text{Tic-Tac-Toe})\) Create a class TicTacToe that will enable you to write a complete program to play the game of Tic-Tac-Toe. The class contains a private 3-by-3 two-dimensional array of integers. The constructor should initialize the empty board to all zeros. Allow two human players. Wherever the first player moves, place a 1 in the specified square, and place a 2 wherever the second player moves. Each move must be in an empty square. After each move, determine whether the game has been won and whether it is a draw. If you feel ambitious, modify your program so that the computer makes the moves for one of the players. Also, allow the player to specify whether he or she wants to go first or second. If you feel exceptionally ambitious, develop a program that will play three-dimensional Tic-Tac-Toe on a 4-by-4-by-4 board. [Note: This is a challenging project that could take many weeks of effort].
10.8 Compare and contrast abstract classes and interfaces. Why would you use an abstract class? Why would you use an interface?

10.9 **(Payroll System Modification)** Modify the payroll system of Figs. 10.4–10.9 to include private instance variable birthdate in class Employee. Use class Date of Fig. 8.7 to represent an employee’s birthday. Add get methods to class Date and replace method toString with method toString. Assume that payroll is processed once per month. Create an array of Employee variables to store references to the various employee objects. In a loop, calculate the payroll for each employee (polymorphically) and add a $1000.00 bonus to the person’s payroll amount if the current month is the one in which the employee’s birthday occurs.

10.10 **(Shape Hierarchy)** Implement the Shape hierarchy shown in Fig. 9.3. Each TwoDimensionalShape should contain method getArea to calculate the area of the two-dimensional shape. Each ThreeDimensionalShape should have methods getArea and getVolume to calculate the surface area and volume, respectively, of the three-dimensional shape. Create a program that uses an array of Shape references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a TwoDimensionalShape or a ThreeDimensionalShape. If it is a TwoDimensionalShape, display its area. If it is a ThreeDimensionalShape, display its area and volume.

10.11 **(Payroll System Modification)** Modify the payroll system of Figs. 10.4–10.9 to include an additional Employee subclass PieceWorker that represents an employee whose pay is based on the number of pieces of merchandise produced. Class PieceWorker should contain private instance variables wage (to store the employee’s wage per piece) and pieces (to store the number of pieces produced). Provide a concrete implementation of method earnings in class PieceWorker that calculates the employee’s earnings by multiplying the number of pieces produced by the wage per piece. Create an array of Employee variables to store references to objects of each concrete class in the new Employee hierarchy. For each employee, display its string representation and earnings.

10.12 **(Account Payable System Modification)** In this exercise, we modify the accounts payable application of Figs. 10.4–10.9 to include the complete functionality of the payroll application of Figs. 10.4–10.9. The application should still process two invoice objects, but now should process one object of each of the four Employee subclasses. If the object currently being processed is a basePlusCommissionEmployee, the application should increase the basePlusCommissionEmployee’s base salary by 10%. Finally, the application should output the payment amount for each object. Complete the following steps to create the new application:

a) Modify class HourlyEmployee (Fig. 10.6) and CommissionEmployee (Fig. 10.7) to place them in the Payable hierarchy as subclasses of the version of Employee (Fig. 10.13) that implements Payable. [Hint: Change the name of method earnings to getPaymentAmount in each subclass so that the class satisfies its inherited contract with interface Payable.]

b) Modify class BasePlusCommissionEmployee (Fig. 10.8) such that it extends the version of class CommissionEmployee created in part a.

c) Modify PayableInterfaceTest (Fig. 10.15) to polymorphically process two invoices: one SalaryEmployee, one HourlyEmployee, one CommissionEmployee, and one BasePlusCommissionEmployee. First output a string representation of each Payable object. Next, if an object is a BasePlusCommissionEmployee, increase its base salary by 10%. Finally, output the payment amount for each Payable object.
11.11 Create the following GUI. You do not have to provide any functionality.

![Temperature Conversion GUI](image)

11.12 Write a temperature conversion application that converts from Fahrenheit to Celsius. The Fahrenheit temperature should be entered from the keyboard (via a `JTextField`). A `JLabel` should be used to display the converted temperature. Use the following formula for the conversion:

\[
\text{Celsius} = \frac{5}{9} \times (\text{Fahrenheit} - 32)
\]

11.13 Enhance the temperature conversion application of Exercise 11.12 by adding the Kelvin temperature scale. The application should also allow the user to make conversions between any two scales. Use the following formula for the conversion between Kelvin and Celsius (in addition to the formula in Exercise 11.12):

\[
\text{Kelvin} = \text{Celsius} + 273.15
\]

11.14 Write an application that displays events as they occur in a `JTextArea`. Provide a `JComboBox` with a minimum of four items. The user should be able to choose an event to monitor from the `JComboBox`. When that particular event occurs, display information about the event in the `JTextArea`. Use `method toString` on the event object to convert it to a string representation.

11.15 Write an application that plays "guess the number" as follows: Your application chooses the number to be guessed by selecting an integer at random in the range 1–1000. The application then displays the following in a `JLabel`:

```
I have a number between 1 and 1000. Can you guess my number?
Please enter your first guess.
```

A `JTextField` should be used to input the guess. As each guess is input, the background color should change to either red or blue. Red indicates that the user is getting warmer, and blue indicates that the user is getting colder. A `JLabel` should display either "Too High" or "Too Low" to help the user zero in on the correct answer. When the user gets the correct answer, "Correct!" should be displayed, and the `JTextField` used for input should be changed to be uneditable. A `JButton` should be provided to allow the user to play the game again. When the `JButton` is clicked, a new random number should be generated and the input `JTextField` changed to be editable.

11.16 It is often useful to display the events that occur during the execution of an application. This can help you understand when the events occur and how they are generated. Write an application that enables the user to generate and process every event discussed in this chapter. The application should provide methods from the `ActionListener`, `ItemListener`, `ListSelectionListener`, `MouseListener`, `MouseMotionListener`, and `KeyListener` interfaces to display messages when the events occur. Use `method toString` to convert the event objects received in each event handler into a `String` that can be displayed. Method `toString` creates a `String` containing all the information in the event object.

11.17 Modify the application of Section 6.10 to provide a GUI that enables the user to click a `JButton` to roll the dice. The application should also display four `JLabels` and four `JTextFields`,
b) Graphics method drawLine draws a line between two points.

c) Graphics method fillArc uses degrees to specify the angle.

d) In the Java coordinate system, values on the y-axis increase from left to right.

e) Graphics inherits directly from class Object.

f) Graphics is an abstract class.

The font class inherits directly from class Graphics.

12.6 (Concentric Circles Using Method drawArc) Write an application that draws a series of eight concentric circles. The circles should be separated by 10 pixels. Use Graphics method drawArc.

12.7 (Concentric Circles Using Class Ellipse2D.Double) Modify your solution to Exercise 12.6 to draw the circles by using class Ellipse2D.Double and method draw of class Graphics2D.

12.8 (Random Lines Using Class Line2D.Double) Modify your solution to Exercise 12.7 to draw random lines, in random colors and random line thicknesses. Use class Line2D.Double and method draw of class Graphics2D to draw the lines.

12.9 (Random Triangles) Write an application that displays randomly generated triangles in different colors. Each triangle should be filled with a different color. Use class GeneralPath and method fill of class Graphics2D to draw the triangles.

12.10 (Random Characters) Write an application that randomly draws characters in different font sizes and colors.


12.12 (Grid Using Class Line2D.Double) Modify your solution to Exercise 12.11 to draw the grid using instances of class Line2D.Double and method draw of class Graphics2D.

12.13 (Grid Using Method drawRect) Write an application that draws a 10-by-10 grid. Use the Graphics method drawRect.

12.14 (Grid Using Class Rectangle2D.Double) Modify your solution to Exercise 12.13 to draw the grid by using class Rectangle2D.Double and method draw of class Graphics2D.

12.15 (Drawing Tetrahedrons) Write an application that draws a tetrahedrons (a three-dimensional shape with four triangular faces). Use class GeneralPath and method draw of class Graphics2D.

12.16 (Drawing Cubes) Write an application that draws a cube. Use class GeneralPath and method draw of class Graphics2D.

12.17 (Circles Using Class Ellipse2D.Double) Write an application that asks the user to input the radius of a circle as a floating-point number and draws the circle, as well as the values of the circle's diameter, circumference and area. Use the value 3.14159 for π. [Note: You may also use the predefined constant Math.PI for the value of π. This constant is more precise than the value 3.14159. Class Math is declared in the java.lang package, so you do not need to import it.] Use the following formulas:

\[
\begin{align*}
\text{diameter} &= 2r \\
\text{circumference} &= 2\pi r \\
\text{area} &= \pi r^2
\end{align*}
\]

The user should also be prompted for a set of coordinates in addition to the radius. Then draw the circle, and display the circle's diameter, circumference and area, using an Ellipse2D.Double object to represent the circle and method draw of class Graphics2D to display the circle.

12.18 (Screen Saver) Write an application that simulates a screen saver. The application should randomly draw lines using method drawLine of class Graphics. After drawing 100 lines, the application should clear itself and start drawing lines again. To allow the program to draw continuously,
place a call to repaint at the last line in method paintComponent. Do you notice any problems with this on your system?

12.19 (Screen Saver Using Timer) Package JavaSwing contains a class called Timer that is capable of calling method actionPerformed of interface ActionListener at a fixed time interval (specified in milliseconds). Modify your solution to Exercise 12.18 to remove the call to repaint from method paintComponent. Declare a class to implement ActionListener. (The actionPerformed method should simply call repaint.) Declare an instance variable of type Timer called timer in your class.

In the constructor for your class, write the following statements:

```java
timer = new Timer(1000, this);
timer.start();
```

This creates an instance of class Timer that will call this object's actionPerformed method every 1000 milliseconds (i.e., every second).

12.20 (Screen Saver for a Random Number of Lines) Modify your solution to Exercise 12.19 to enable the user to enter the number of random lines that should be drawn before the application clears itself and starts drawing lines again. Use a JTextField to obtain the value. The user should be able to type a new number into the JTextField at any time during the program's execution. Use an inner class to perform event handling for the JTextField.

12.21 (Screen Saver with Shapes) Modify your solution to Exercise 12.19 such that it uses random-number generation to choose different shapes to display. Use methods of class Graphics.

12.22 (Screen Saver Using the Java 2D API) Modify your solution to Exercise 12.21 to use classes and drawing capabilities of the Java 2D API. Draw shapes like rectangles and ellipses, with randomly generated gradients. Use class GradientPaint to generate the gradient.

12.23 (Turtle Graphics) Modify your solution to Exercise 7.21—Turtle Graphics—to add a graphical user interface using JTextFields and JButton, Draw lines rather than arrows (→). When the turtle graphics program specifies a move, translate the number of positions into a number of pixels on the screen by multiplying the number of positions by 10 (or any value you choose). Implement the drawing with Java 2D API features.

12.24 (Knight's Tour) Produce a graphical version of the Knight's Tour problem (Exercise 7.22, Exercise 7.23, and Exercise 7.26). As each move is made, the appropriate cell of the chessboard should be updated with the proper move number. If the result of the program is a full tour or a closed tour, the program should display an appropriate message. If you like, use class Timer (see Exercise 12.19) to help animate the Knight's Tour.

12.25 (Tortoise and Hare) Produce a graphical version of the Tortoise and Hare simulation (Exercise 7.28). Simulate the movement by drawing an arc that extends from the bottom-left corner of the window to the top-right corner of the window. The tortoise and the hare should race up the mountain, and implement the graphical output to actually print the tortoise and the hare on the arc for every move. [Hint: Extend the length of the race from 70 to 500 to allow yourself a larger graphics area.]

12.26 (Drawing Spirals) Write an application that uses Graphics method drawPolygon to draw a spiral similar to the one shown in Fig. 12.33.

12.27 (Pie Charts) Write a program that inputs four numbers and graphs them as a pie chart. Use class Arc2D.Double and method fillOf class Graphics2D to perform the drawing. Draw each piece of the pie in a separate color.

12.28 (Selecting Shapes) Write an application that allows the user to select a shape from a JComboBox and draws it 20 times with random locations and dimensions in method paintComponent. The first item in the JComboBox should be the default shape that is displayed the first time paintComponent is called.
300  83.89
700  80.78
700  1.53

Fig. 14.26 | Additional transaction records.

14.10  *(File Matching with Object Serialization)* Recreate your solution for Exercise 14.9 using object serialization. Use the statements from Exercise 14.4 as your basis for this program. You may want to create applications to read the data stored in the .ser files—the code in Section 14.6.2 can be modified for this purpose.

14.11  *(Telephone-Number Word Generator)* Standard telephone keypads contain the digits zero through nine. The numbers two through nine each have three letters associated with them (Fig. 14.27). Many people find it difficult to memorize phone numbers, so they use the correspondence between digits and letters to develop seven-letter words that correspond to their phone numbers. For example, a person whose telephone number is 686-2577 might use the correspondence indicated in Fig. 14.27 to develop the seven-letter word "NUMBERS." Every seven-letter word corresponds to exactly one seven-digit telephone number. A restaurant wishing to increase its takeout business could surely do so with the number 825-3688 (i.e., "TAKEOUT").

<table>
<thead>
<tr>
<th>Digit</th>
<th>Letters</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ABC</td>
</tr>
<tr>
<td>3</td>
<td>DEF</td>
</tr>
<tr>
<td>4</td>
<td>GHI</td>
</tr>
<tr>
<td>5</td>
<td>JKL</td>
</tr>
<tr>
<td>6</td>
<td>MNO</td>
</tr>
<tr>
<td>7</td>
<td>PRS</td>
</tr>
<tr>
<td>8</td>
<td>TUV</td>
</tr>
<tr>
<td>9</td>
<td>WXY</td>
</tr>
</tbody>
</table>

Fig. 14.27 | Telephone keypad digits and letters.

Every seven-letter phone number corresponds to many different seven-letter words. Unfortunately, most of these words represent unrecognizable juxtapositions of letters. It is possible, however, that the owner of a barbershop would be pleased to know that the shop's telephone number, 424-7228, corresponds to "HAIRCUT." The owner of a liquor store would, no doubt, be delighted to find that the store's number, 233-7226, corresponds to "BEERCAN." A veterinarian with the phone number 738-2273 would be pleased to know that the number corresponds to the letters "PETCARE." An automobile dealership would be pleased to know that the dealership number, 639-3777, corresponds to "NEWCARS."
15.14 (Palindrome) A palindrome is a string that is spelled the same way forward and backward. Some examples of palindromes are “racecar,” “able was I saw Elba” and (if spaces are ignored) “madam, I’m Adam.” Write a recursive method testPalindrome that returns true if the string stored in the array is a palindrome and false otherwise. The method should ignore spaces and punctuation in the string.

15.15 (Eight Queens) A puzzle for chess buffs is the Eight Queens problem, which asks the following: Is it possible to place eight queens on an empty chessboard so that no queen is “attacking” any other (i.e., no two queens are in the same row, in the same column or along the same diagonal)? For instance, if a queen is placed in the upper-left corner of the board, no other queens could be placed in any of the marked squares shown in Fig. 15.23. Solve the problem recursively. [Hint: Your solution should begin with the first column and look for a location in that column where a queen can be placed—initially, place the queen in the first row. The solution should then recursively search the remaining columns. In the first few columns, there will be several locations where a queen may be placed. Take the first available location. If a column is reached with no possible location for a queen, the program should return to the previous column and move the queen in that column to a new row. This continues backtracking and trying new alternatives is an example of recursive backtracking.]

15.16 (Print an Array) Write a recursive method printArray that displays all the elements in an array of integers, separated by spaces.

15.17 (Print an Array Backward) Write a recursive method printArrayReverse that takes a character array containing a string as an argument and prints the string backward. [Hint: Use String method toCharArray, which takes no arguments, to get a char array containing the characters in the String.]

15.18 (Find the Minimum Value in an Array) Write a recursive method recursiveMinimum that determines the smallest element in an array of integers. The method should return when it receives an array of one element.

15.19 (Fractal) Repeat the fractal pattern in Section 15.8 to form a star. Begin with five lines in a star whose each line is a different arm of the star. Apply the “La fràccal” pattern to each arm of the star.

15.20 (Maze Traversal Using Recursive Backtracking) The grid of #s and dots (.) in Fig. 15.24 is a two-dimensional array representation of a maze. The #s represent the walls of the maze, and the dots represent locations in the possible paths through the maze. Moves can be made only to a location in the array that contains a dot.

Fig. 15.23 | Squares eliminated by placing a queen in the upper-left corner of a chessboard.
Self-Review Exercises

16.1 Fill in the blanks in each of the following statements:
   a) A selection sort application would take approximately _______ times as long to run
      on a 128-element array as on a 32-element array.
      b) The efficiency of merge sort is _______.

16.2 What key aspect of both the binary search and the merge sort accounts for the logarithmic
    portion of their respective Big Os?

16.3 In what sense is the insertion sort superior to the merge sort? In what sense is the merge sort
    superior to the insertion sort?

16.4 In the text, we say that after the merge sort splits the array into two subarrays, it then sorts
    these two subarrays and merges them. Why might someone be puzzled by our statement that "it
    then sorts these two subarrays"?

Answers to Self-Review Exercises

16.1 a) 16 because an $O(n^2)$ algorithm takes 16 times as long to sort four times as much
      information. b) $O(n \log n)$.

16.2 Both of these algorithms incorporate "halving"—somehow reducing something by half.
      The binary search eliminates from consideration one half of the array after each comparison.
      The merge sort splits the array in half each time it is called.

16.3 The insertion sort is easier to understand and to program than the merge sort. The merge
      sort is far more efficient [$O(n \log n)$] than the insertion sort [$O(n^2)$].

16.4 In a sense, it does not really sort these two subarrays. It simply keeps splitting the original
      array in half until it provides a one-element subarray, which is, of course, sorted. It then builds
      up the original two subarrays by merging these one-element arrays to form larger subarrays,
      which are then merged, and so on.

Exercises

16.5 (Bubble Sort) Implement bubble sort—another simple yet inefficient sorting technique. It
      is called bubble sort or sinking sort because smaller values gradually "bubble" their way to the top
      of the array (i.e., toward the first element) like air bubbles rising in water, while the larger values
      sink to the bottom (end) of the array. The technique uses nested loops to make several passes
      through the array. Each pass compares successive pairs of elements. If a pair is in increasing order
      (or the values are equal), the bubble sort leaves the values as they are. If a pair is in decreasing order,
      the bubble sort swaps their values in the array.
      The first pass compares the first two elements of the array and swaps their values if necessary.
      It then compares the second and third elements in the array. The end of this pass compares the last
      two elements in the array and swaps them if necessary. After one pass, the largest element will be in
      the last index. After two passes, the largest two elements will be in the last two indices. Explain why
      bubble sort is an $O(n^2)$ algorithm.

16.6 (Enhanced Bubble Sort) Make the following simple modifications to improve the performance
      of the bubble sort you developed in Exercise 16.5:
      a) After the first pass, the largest number is guaranteed to be in the highest-numbered
         element of the array; after the second pass, the two highest numbers are "in place"; and
         so on. Instead of making nine comparisons on every pass, modify the bubble sort to
         make eight comparisons on the second pass, seven on the third pass, and so on.
b) The data in the array may already be in proper or near-proper order, so why make
nine passes if fewer will suffice? Modify the sort to check at the end of each pass whether any
swaps have been made. If none have been made, the data must already be in the proper
order, so the program should terminate. If swaps have been made, at least one more pass
is needed.

16.7 (Bucket Sort) A bucket sort begins with a one-dimensional array of positive integers to be
sorted and a two-dimensional array of integers with rows indexed from 0 to 9 and columns indexed
from 0 to n - 1, where n is the number of values to be sorted. Each row of the two-dimensional
array is referred to as a bucket. Write a class named Bucketsort containing a method called sort that op-
erates as follows:

a) Place each value of the one-dimensional array into a row of the bucket array, based on
the value's "ones" (rightmost) digit. For example, 97 is placed in row 7, 5 is placed in
row 3 and 100 is placed in row 0. This procedure is called a distribution pass.

b) Loop through the bucket array row by row, and copy the values back to the original ar-
ray. This procedure is called a gathering pass. The new order of the preceding values in
the one-dimensional array is 100, 3 and 97.

c) Repeat this process for each subsequent digit position (tens, hundreds, thousands, etc.).
On the second (tens digit) pass, 100 is placed in row 0, 3 is placed in row 0 (because 3
has no tens digit) and 97 is placed in row 9. After the gathering pass, the order of the
values in the one-dimensional array is 100, 3 and 97. On the third (hundreds digit)
pass, 100 is placed in row 1, 3 is placed in row 0 and 97 is placed in row 0 (after the 3).
After this last gathering pass, the original array is in sorted order.

Note that the two-dimensional array of buckets is 10 times the length of the inte-
ger array being sorted. This sorting technique provides better performance than a bub-
ble sort, but requires much more memory—the bubble sort requires space for only one
additional element of data. This comparison is an example of the space–time trade-off.
The bucket sort uses more memory than the bubble sort, but performs better. This
version of the bucket sort requires copying all the data back to the original array on
each pass. Another possibility is to create a second two-dimensional bucket array and
repeatedly swap the data between the two bucket arrays.

16.8 (Recursive Linear Search) Modify Fig. 16.2 to use recursive method recursiveLinearSearch
to perform a linear search of the array. The method should receive the search key and start
ing index as arguments. If the search key is found, return its index in the array; otherwise, return –1. Each call
to the recursive method should check one index in the array.

16.9 (Recursive Binary Search) Modify Fig. 16.4 to use recursive method recursiveBinarySearch
to perform a binary search of the array. The method should receive the search key, start
ning index and ending index as arguments. If the search key is found, return its index in the array. If the search
key is not found, return –1.

16.10 (Quicksort) The recursive sorting technique called quicksort uses the following basic algo-

rithm for a one-dimensional array of values:

a) Partitioning Step: Take the first element of the unsorted array and determine its final
location in the sorted array (i.e., all values to the left of the element in the array are less
than the element, and all values to the right of the element in the array are greater than
the element—we show how to do this below). We now have one element in its proper
location and two unsorted subarrays.

b) Recursive Step: Perform Step 1 on each unsorted subarray. Each time Step 1 is performed
on a subarray, another element is placed in its final location of the sorted array, and two
unsorted subarrays are created. When a subarray consists of one element, that element
is in its final location (because a one-element array is already sorted).
Exercises 841

(Note: In b) above (based on the sample expression at the beginning of this exercise), if the operator is '/', the top of the stack is 2 and the next element in the stack is 8, then pop 2 into 4, pop 8 into 8, evaluate 8/2 and push the result, 4, back on the stack. This case also applies to operator '^-'.] The arithmetic operations allowed in an expression are:

- Addition
- Subtraction
- Multiplication
- Division
- Exponentiation
- Modulus

The stack should be maintained with one of the stack classes introduced in this chapter. You may want to provide the following methods:

a) Method evaluatePostfixExpression, which evaluates the postfix expression.
b) Method calculate, which evaluates the expression opt operator opt.
c) Method push, which pushes a value onto the stack.
d) Method pop, which pops a value off the stack.
e) Method isEmpty, which determines whether the stack is empty.
f) Method printStack, which prints the stack.

17.14 Modify the postfix evaluator program of Exercise 17.13 so that it can process integer operands larger than 9.

17.15 (Supermarket Simulation) Write a program that simulates a checkout line at a supermarket. The line is a queue object. Customers (i.e., customer objects) arrive in random integer intervals of from 1 to 4 minutes. Also, each customer is serviced in random integer intervals of from 1 to 4 minutes. Obviously, the rates need to be balanced. If the average arrival rate is larger than the average service rate, the queue will grow infinitely. Even with "balanced" rates, randomness can still cause long lines. Run the supermarket simulation for a 12-hour day (720 minutes), using the following algorithm:

a) Choose a random integer between 1 and 4 to determine the minute at which the first customer arrives.
b) At the first customer's arrival time, do the following:
   Determine customer's service time (random integer from 1 to 4).
   Begin servicing the customer.
   Schedule arrival time of next customer (random integer 1 to 4 added to the current time).
c) For each minute of the day, consider the following:
   If the next customer arrives, proceed as follows:
   Say so.
   Enqueue the customer.
   Schedule the arrival time of the next customer.
   If service was completed for the last customer, do the following:
   Say so.
   Dequeue next customer to be serviced.
   Determine customer's service completion time (random integer from 1 to 4 added to the current time).

Now run your simulation for 720 minutes and answer each of the following:

a) What is the maximum number of customers in the queue at any time?
b) What is the longest wait any one customer experiences?
c) What happens if the arrival interval is changed from 1 to 4 minutes to 1 to 3 minutes?

17.16 Modify Figs. 17.17 and 17.18 to allow the binary tree to contain duplicates.
The deletion steps for a replacement node with a left child are similar to those for a replacement node with no children, but the algorithm also must move the child into the replacement node's position in the tree. If the replacement node is a node with a left child, the steps to perform the deletion are as follows:

1. Store the reference to the node to be deleted in a temporary reference variable.
2. Set the reference in the parent of the node being deleted to reference the replacement node.
3. Set the reference in the parent of the replacement node to reference the left child of the replacement node.
4. Set the reference to the right subtree in the replacement node to reference the right subtree of the node to be deleted.
5. Set the reference to the left subtree in the replacement node to reference the left subtree of the node to be deleted.

Write method `deleteNode`, which takes as its argument the value to delete. Method `deleteNode` should locate in the tree the node containing the value to delete and use the algorithms discussed here to delete the node. If the value is not found in the tree, the method should print a message that indicates whether the value is deleted. Modify the program of Figs. 17.17 and 17.18 to use this method. After deleting an item, call the methods `inorderTraversal`, `preorderTraversal` and `postorderTraversal` to confirm that the delete operation was performed correctly.

17.23 (Binary Tree Search) Write method `binaryTreeSearch`, which attempts to locate a specified value in a binary search tree object. The method should take as an argument a search key to be located. If the node containing the search key is found, the method should return a reference to that node; otherwise, it should return a null reference.

17.24 (Level-Order Binary Tree Traversal) The program of Figs. 17.17 and 17.18 illustrated three recursive methods of traversing a binary tree—inorder, preorder and postorder traversals. This exercise presents the level-order traversal of a binary tree, in which the node values are printed level by level, starting at the root node level. The nodes on each level are printed from left to right. The level-order traversal is not a recursive algorithm. It uses a queue object to control the output of the nodes. The algorithm is as follows:

1. Insert the root node in the queue.
2. While there are nodes left in the queue, do the following:
   a. Get the next node in the queue.
   b. Print the node’s value.
   c. If the reference to the left child of the node is not null:
      i. Insert the left child node in the queue.
   d. If the reference to the right child of the node is not null:
      i. Insert the right child node in the queue.

Write method `levelOrder` to perform a level-order traversal of a binary tree object. Modify the program of Figs. 17.17 and 17.18 to use this method. [Note: You will also need to use queue-processing methods of Fig. 17.13 in this program.]

17.25 (Printing Tree) Write a recursive method `outputTree` to display a binary tree object on the screen. The method should output the tree row by row, with the top of the tree at the left of the screen and the bottom of the tree toward the right of the screen. Each row is output vertically. For example, the binary tree illustrated in Fig. 17.20 is output as shown in Fig. 17.21.

The rightmost leaf node appears at the top of the output in the rightmost column and the root node appears at the left of the output. Each column starts five spaces to the right of the preceding column. Method `outputTree` should receive an argument `totalSpaces` representing the number of spaces preceding the value to be output. (This variable should start at zero so that the root node is output at the left of the screen.) The method uses a modified inorder traversal to output the tree—starting at the rightmost node in the tree and working back to the left. The algorithm is as follows:
Fig. 17.21 | Sample output of recursive method outputTree.

While the reference to the current node is not null, perform the following:
Recursively call outputTree with the right subtree of the current node and totalSpaces + 5.
Use a for statement to count from 1 to totalSpaces and output spaces.
Output the value in the current node.
Set the reference to the current node to refer to the left subtree of the current node.
Increment totalSpaces by 5.

Special Section: Building Your Own Compiler

In Exercises 7.34—7.35, we introduced Simpleton Machine Language (SML), and you implemented a Simpleton computer simulator to execute programs written in SML. In this section, we build a compiler that converts programs written in a high-level programming language to SML. This section "ties" together the entire programming process. You will write programs in this new high-level language, compile them on the compiler you build and run them on the simulator you built in Exercise 7.35. You should make every effort to implement your compiler in an object-oriented manner.

17.26 (The Simple Language) Before we begin building the compiler, we discuss a simple, yet powerful high-level language similar to early versions of the popular language BASIC. We call the language Simple. Every Simple statement consists of a line number and a Simple instruction. Line numbers must appear in ascending order. Each instruction begins with one of the following Simple commands: com, input, let, print, goto, if/goto or end (see Fig. 17.22). All commands except end can be used repeatedly. Simple evaluates only integer expressions using the +, −, * and / operators. These operators have the same precedence as in Java. Parentheses can be used to change the order of evaluation of an expression.

Our Simple compiler recognizes only lowercase letters. All characters in a Simple file should be lowercase. (Uppercase letters result in a syntax error unless they appear in a com statement, in which case they are ignored.) A variable name is a single letter. Simple does not allow descriptive variable names, so variables should be explained in remarks to indicate their use in a program. Simple uses only integer variables. Simple does not have variable declarations—merely mentioning a variable name in a program causes the variable to be declared and initialized to zero. The syntax of Simple does not allow string manipulation (reading a string, writing a string, comparing strings, and so on). If a string is encountered in a Simple program (after a command other than com), the compiler generates a syntax error. The first version of our compiler assumes that Simple programs are entered correctly. Exercise 17.29 asks the reader to modify the compiler to perform syntax error checking.
brackets (c and d), c) type parameters d) a

Java program:

1. on the sort program of Figs. 16.6-16.7.
integer array and a float array. \[Hint: Use
tion for method selection sort, so that you
no generic types.\]

3.3 so that it takes two additional integer ar-
his method prints only the designated par-
absorb. If either is out of range, or if
the overloaded printArray method should
array should return the number of ele-
ems of printArray on arrays integerArray,
doubleArray and characterArray. Test all capabilities of both versions of printArray.

18.5 Overload generic method printArray of Fig. 16.3 with a non-generic version that specifically
prints an array of strings in neat, tabular format, as shown in the sample output that follows:

<table>
<thead>
<tr>
<th>Array</th>
<th>stringArray contains:</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
<td>two</td>
</tr>
<tr>
<td>five</td>
<td>six</td>
</tr>
<tr>
<td>three</td>
<td>four</td>
</tr>
<tr>
<td>seven</td>
<td>eight</td>
</tr>
</tbody>
</table>

18.7 Write a simple generic version of method \( \text{isEqual}() \) that compares its two arguments with
the equals method and returns true if they are equal and false otherwise. Use this generic method
in a program that calls \( \text{isEqual}() \) with a variety of built-in types, such as object or Integer. What
result do you get when you attempt to run this program?

18.8 Write a generic class \( \text{Pair} \) which has two type parameters E and F, each representing
the type of the first and second element of the pair, respectively. Add \( \text{get} \) and \( \text{set} \) methods for the first
and second elements of the pair. \[Hint: The class header should be public class \text{Pair} \{E, F \} .\]

18.9 Convert classes \( \text{Treenode} \) and \( \text{Tree} \) from Fig. 17.17 into generic classes. To insert an object
in a Tree, the object must be compared to the objects in existing TreeNode. For this reason, classes
\( \text{Treenode} \) and \( \text{Tree} \) should specify \( \text{Comparable} \) as the upper bound of each class's type parameter.
After modifying classes \( \text{Treenode} \) and \( \text{Tree} \), write a new method that creates three Tree objects—one that stores Integers, one that stores doubles and one that stores Strings. Insert 10 values
into each tree. Then output the preorder, inorder and postorder traversals for each Tree.

18.10 Modify your test program from Exercise 18.9 to use a generic method named \( \text{testTree} \) to
test the three Tree objects. The method should be called three times—one for each Tree object.

18.11 How can generic methods be overloaded?

18.12 The compiler performs a matching process to determine which method to call when a
method is invoked. Under what circumstances does an attempt to make a match result in a compile-
time error?

18.13 Explain why a Java program might use the statement

```java
ArrayList< Employee > workerList = new ArrayList< Employee >();
```
19.10 Determine whether each of the following statements is true or false. If false, explain why.
   a) Elements in a Collection must be sorted in ascending order before a binary search may be performed.
   b) Method first gets the first element in a TreeSet.
   c) A List created with the ArrayList method is resizable.
   d) Clazz Arrays provides static methods sort for sorting array elements.

19.11 Explain the operation of each of the following methods of the Properties class:
   a) load
   b) store
   c) getProperty
   d) list

19.12 Rewrite lines 17–26 in Fig. 19.4 to be more concise by using the add method and the
    LinkedList constructor that takes a Collection argument.

19.13 Write a program that reads in a series of first names and stores them in a List. Do not store duplicate names. Allow the user to search for a first name.

19.14 Modify the program of Fig. 19.20 to count the number of occurrences of each letter rather than of each word. For example, the string "HELLO THERE" contains two H, three E, two L, one O, one T and one R. Display the results.

19.15 Use a HashMap to create a reusable class for choosing one of the 13 predefined colors in class Color. The names of the colors should be used as keys, and the predefined Color objects should be used as values. Place this class in a package that can be imported into any Java program. Use your new class in an application that allows the user to select a color and draw a shape in that color.

19.16 Write a program that determines and prints the number of duplicate words in a sentence. Treat uppercase and lowercase letters the same. Ignore punctuation.

19.17 Rewrite your solution to Exercise 17.8 to use a LinkedList collection.

19.18 Rewrite your solution to Exercise 17.9 to use a LinkedList collection.

19.19 Write a program that takes a whole number input from a user and determines whether it is prime. If the number is not prime, display its unique prime factors. Remember that a prime number's factors are only 1 and the prime number itself. Every number that is not prime has a unique prime factorization. For example, consider the number 54. The prime factors of 54 are 2, 3, 3 and 3. When the values are multiplied together, the result is 54. For the number 54, the prime factor output should be 2 and 3. Use Set s as part of your solution.

19.20 Write a program that uses a StringTokenizer to tokenize a line of text input by the user and places each token in a TreeSet. Print the elements of the TreeSet. [Note: This should cause the elements to be printed in ascending sorted order.]

19.21 The output of Fig. 19.17 (PriorityQueueTest) shows that PriorityQueue orders Double elements in ascending order. Rewrite Fig. 19.17 so that it orders Double elements in descending order (i.e., 3.8 should be the highest-priority element rather than 3.2).
Exercises

20.2 Write an applet that asks the user to enter two floating-point numbers, obtains the two numbers from the user and draws their sum, product (multiplication), difference and quotient (division). Use the techniques shown in Fig. 20.10.

20.3 Write an applet that asks the user to enter two floating-point numbers, obtains the numbers from the user and displays the two numbers first and then the larger number followed by the words "is larger" as a string on the applet. If the numbers are equal, the applet should print the message "These numbers are equal." Use the techniques shown in Fig. 20.10.

20.4 Write an applet that inputs three floating-point numbers from the user and displays the sum, average, product, smallest and largest of those numbers as strings on the applet. Use the techniques shown in Fig. 20.10.

20.5 Write an applet that asks the user to input the radius of a circle as a floating-point number and draws the circle’s diameter, circumference and area. Use the value $3.14159$ for $\pi$. Use the techniques shown in Fig. 20.10. [Note: You may also use the predefined constant $\text{Math.PI}$ for the value of $\pi$. This constant is more precise than the value $3.14159$. Class $\text{Math}$ is defined in the java.lang package, so you do not need to import it.] Use the following formulas ($r$ is the radius):

\[
\begin{align*}
\text{diameter} & = 2r \\
\text{circumference} & = 2\pi r \\
\text{area} & = \pi r^2
\end{align*}
\]

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

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

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

20.8 Write an applet that draws rectangles of different sizes and locations.

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

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

20.11 Modify the solution to Exercise 20.10 to output ovals of different shapes and sizes.

20.12 Write an applet that allows the user to input the four arguments required by method drawOval, then draws an oval using the four input values.
21.27 (Fireworks Designer) Create a Java program that someone might use to create a fireworks display. Create a variety of fireworks demonstrations. Then orchestrate the firing of the fireworks for maximum effect.

21.28 (Floor Planner) Develop a Java program that will help someone arrange furniture in a home. Add features that enable the person to achieve the best possible arrangement.

21.29 (Crossword) Crossword puzzles are among the most popular pastimes. Develop a multimedia-based crossword-puzzle program. Your program should enable the player to place and erase words easily. Tie your program to a large computerized dictionary. Your program should also be able to suggest words on which letters have already been filled in. Provide other features that will make the crossword-puzzle enthusiast’s job easier.

21.30 (15 Puzzle) Write a multimedia-based Java program that enables the user to play the game of 15. The game is played on a 4 by 4 board for a total of 16 slots. One slot is empty, the others are occupied by 15 tiles numbered 1 through 15. Any tile next to the currently empty slot can be moved into that slot by clicking on the tile. Your program should create the board with the tiles out of order. The goal is to arrange the tiles into sequential order, row by row.

21.31 (Reaction/Time/Reaction Speed Test) Create a Java program that measures a randomly created shape around the screen. The user moves the mouse to catch and click on the shape. The speed of the shape can be varied. Keep statistics on how much time the user typically takes to catch a shape of a given size. The user will probably have trouble noticing slower-moving, smaller shapes.

21.32 (Calendar/Tickler File) Using both audio and images, create a general-purpose calendar and “tickler” file. For example, the program should sing “Happy Birthday” when you use it on your birthday. Have the program display images and play audio associated with important events. Also have the program remind you in advance of these important events. It would be nice, for example, to have the program give you a week’s notice so you can pick up an appropriate greeting card for that special person.

21.33 (Rotating Images) Create a Java program that lets you rotate an image through some number of degrees (out of a maximum of 360 degrees). The program should let you specify that you want to spin the image continuously. It should let you adjust the spin speed dynamically.

21.34 (Coloring Black-and-White Photographs and Images) Create a Java program that lets you paint a black-and-white photograph with color. Provide a color palette for selecting colors. Your program should let you apply different colors to different regions of the image.

21.35 (Multimedia-based Simpletron Simulator) Modify the Simpletron simulator that you developed in the exercises from the previous chapters (Exercises 7.34–7.36 and Exercises 17.2–17.30) to include multimedia features. Add computer-like sounds to indicate that the Simpletron is executing instructions. Add a breaking-glass sound when a fatal error occurs. Use flashing lights to indicate which cells of memory or which registers are currently being manipulated. Use other multimedia techniques, as appropriate, to make your Simpletron simulator a more valuable tool as an educational tool.
23.5 Discuss each of the following terms in the context of Java’s threading mechanisms:
   a) synchronized
   b) producer
   c) consumer
   d) wait
   e) notify
   f) Lock
   g) Condition

23.6 List the reasons for entering the blocked state. For each of these, describe how the program will normally leave the blocked state and enter the runnable state.

23.7 Two problems that can occur in systems that allow threads to wait are deadlock, in which one or more threads will wait forever for an event that cannot occur, and indefinite postponement, in which one or more threads will be delayed for some unpredictably long time. Give an example of how each of these problems can occur in multithreaded Java programs.

23.8 Write a program that bounces a blue ball inside a JPanel. The ball should begin moving with a mousePressed event. When the ball hits the edge of the JPanel, it should bounce off the edge and continue in the opposite direction. The ball should be updated using a Runnable.

23.9 Modify the program in Exercise 23.8 to add a new ball each time the user clicks the mouse. Provide for a minimum of 20 balls. Randomly choose the color for each new ball.

23.10 Modify the program in Exercise 23.9 to add shadows. As a ball moves, draw a solid black oval at the bottom of the JPanel. You may consider adding a 3-D effect by increasing or decreasing the size of each ball when it hits the edge of the JPanel.
24.21 (Modifications to the Multithreaded Tic-Tac-Toe Program) The programs in Figs. 24.13 and 24.15 implemented a multithreaded, client/server version of the game of Tic-Tac-Toe. Our goal in developing this game was to demonstrate a multithreaded server that could process multiple connections from clients at the same time. The server in the example is really a mediator between the two client applets—making sure that each move is valid and that each client moves in the proper order. The server does not determine who won or lost or whether there was a draw. Also, there is no capability to allow a new game to be played or to terminate an existing game.

The following is a list of suggested modifications to Figs. 24.13 and 24.15:

a) Modify the TicTacToServer class to test for a win, loss or draw on each move in the game. Send a message to each client applet that indicates the result of the game when the game is over.

b) Modify the TicTacToClient class to display a button that allows the client to play another game. The button should be enabled only when a game completes. Note that both class TicTacToClient and class TicTacToServer must be modified to reset the board and all state information. Also, the other TicTacToClient should be notified that a new game is about to begin so that its board and state can be reset.

c) Modify the TicTacToClient class to provide a button that allows a client to terminate the program at any time. When the user clicks the button, the server and the other client should be notified. The server should then wait for a connection from another client so that a new game can begin.

d) Modify the TicTacToClient class and the TicTacToServer class so that the winner of a game can choose game piece X or O for the next game. Remember: X always goes first.

e) If you would like to be ambitious, allow a client to play against the server while the server waits for a connection from another client.

24.22 (3D Multithreaded Tic-Tac-Toe) Modify the multithreaded, client/server Tic-Tac-Toe program to implement a three-dimensional 4-by-4-by-4 version of the game. Implement the server application to mediate between the two clients. Display the three-dimensional board as four boards containing four rows and four columns each. If you would like to be ambitious, try the following modifications:

a) Draw the board in a three-dimensional manner.

b) Allow the server to test for a win, loss or draw. Beware! There are many possible ways to win on a 4-by-4-by-4 board!

24.23 (Networked Morse Code) Perhaps the most famous of all coding schemes is the Morse code, developed by Samuel Morse in 1832 for use with the telegraph system. The Morse code assigns a series of dots and dashes to each letter of the alphabet, each digit, and a few special characters (e.g., period, comma, colon and semicolon). In sound-oriented systems, the dot represents a short sound and the dash represents a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems. Separation between words is indicated by a space, simply the absence of a dot or dash. In a sound-oriented system, a space is indicated by a short time during which no sound is transmitted. The international version of the Morse code appears in Fig. 24.18.

Write a client/server application in which two clients can send Morse-code messages to each other through a multithreaded server application. The client application should allow the user to type English-language phrases in a JTextField. When the user sends the message, the client application encodes the text into Morse code and sends the coded message through the server to the other client. Use one blank between each Morse-coded letter and three blanks between each Morse-coded word. When messages are received, they should be decoded and displayed as normal characters and as Morse code. The client should have one JTextField for typing and one JTextArea for displaying the other client's messages.