**Section 7.1 Introduction**

• Arrays (p. [244](http://proquest.safaribooksonline.com/9780133813036/ch07_html#page_244)) are fixed-length data structures consisting of related data items of the same type.

#### Section 7.2 Arrays

• An array is a group of variables (called elements or components; p. [245](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec1_html#page_245)) containing values that all have the same type. Arrays are objects, so they’re considered reference types.

• A program refers to any one of an array’s elements with an array-access expression (p. [245](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec1_html#page_245)) that includes the name of the array followed by the index of the particular element in square brackets ([]; p. [245](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec1_html#page_245)).

• The first element in every array has index zero (p. [245](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec1_html#page_245)) and is sometimes called the zeroth element.

• An index must be a nonnegative integer. A program can use an expression as an index.

• An array object knows its own length and stores this information in a length instance variable (p. [246](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec2_html#page_246)).

#### Section 7.3 Declaring and Creating Arrays

• To create an array object, specify the array’s element type and the number of elements as part of an array-creation expression (p. [246](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec2_html#page_246)) that uses keyword new.

• When an array is created, each element receives a default value—zero for numeric primitive-type elements, false for boolean elements and null for references.

• In an array declaration, the type and the square brackets can be combined at the beginning of the declaration to indicate that all the identifiers in the declaration are array variables.

• Every element of a primitive-type array contains a variable of the array’s declared type. Every element of a reference-type array is a reference to an object of the array’s declared type.

#### Section 7.4 Examples Using Arrays

• A program can create an array and initialize its elements with an array initializer (p. [248](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec3_html#page_248)).

• Constant variables (p. [250](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec5_html#page_250)) are declared with keyword final, must be initialized before they’re used and cannot be modified thereafter.

#### Section 7.5 Exception Handling: Processing the Incorrect Response

• An exception indicates a problem that occurs while a program executes. The name “exception” suggests that the problem occurs infrequently—if the “rule” is that a statement normally executes correctly, then the problem represents the “exception to the rule.”

• Exception handling (p. [256](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#page_256)) enables you to create fault-tolerant programs.

• When a Java program executes, the JVM checks array indices to ensure that they’re greater than or equal to 0 and less than the array’s length. If a program uses an invalid index, Java generates an exception (p. [256](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#page_256)) to indicate that an error occurred in the program at execution time.

• To handle an exception, place any code that might throw an exception (p. [256](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#page_256)) in a try statement.

• The try block (p. [256](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#page_256)) contains the code that might throw an exception, and the catch block (p. [256](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec9_html#page_256)) contains the code that handles the exception if one occurs.

• You can have many catch blocks to handle different types of exceptions that might be thrown in the corresponding tryblock.

• When a try block terminates, any variables declared in thetry block go out of scope.

• A catch block declares a type and an exception parameter. Inside the catch block, you can use the parameter’s identifier to interact with a caught exception object.

• When a program is executed, array element indices are checked for validity—all indices must be greater than or equal to 0 and less than the length of the array. If an attempt is made to use an invalid index to access an element, an ArrayIndexOutOfRangeException (p. [257](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec11_html#page_257)) exception occurs.

• An exception object’s toString method returns the exception’s error message.

#### Section 7.6 Case Study: Card Shuffling and Dealing Simulation

• The toString method of an object is called implicitly when the object is used where a String is expected (e.g., when printfoutputs the object as a String using the %s format specifier or when the object is concatenated to a String using the +operator).

#### Section 7.7 Enhanced for Statement

• The enhanced for statement (p. [262](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec7_html#page_262)) allows you to iterate through the elements of an array or a collection without using a counter. The syntax of an enhanced for statement is:

for (*parameter* : *arrayName*)  
   *statement*

where parameter has a type and an identifier (e.g., int number), and arrayName is the array through which to iterate.

• The enhanced for statement cannot be used to modify elements in an array. If a program needs to modify elements, use the traditional counter-controlled for statement.

#### Section 7.8 Passing Arrays to Methods

• When an argument is passed by value, a copy of the argument’s value is made and passed to the called method. The called method works exclusively with the copy.

#### Section 7.9 Pass-By-Value vs. Pass-By-Reference

• When an argument is passed by reference (p. [265](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec8_html#page_265)), the called method can access the argument’s value in the caller directly and possibly modify it.

• All arguments in Java are passed by value. A method call can pass two types of values to a method—copies of primitive values and copies of references to objects. Although an object’s reference is passed by value (p. [265](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec8_html#page_265)), a method can still interact with the referenced object by calling its publicmethods using the copy of the object’s reference.

• To pass an object reference to a method, simply specify in the method call the name of the variable that refers to the object.

• When you pass an array or an individual array element of a reference type to a method, the called method receives a copy of the array or element’s reference. When you pass an individual element of a primitive type, the called method receives a copy of the element’s value.

• To pass an individual array element to a method, use the indexed name of the array.

#### Section 7.11 Multidimensional Arrays

• Multidimensional arrays with two dimensions are often used to represent tables of values consisting of information arranged in rows and columns.

• A two-dimensional array (p. [272](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec11_html#page_272)) with m rows and n columns is called an m-by-n array. Such an array can be initialized with an array initializer of the form

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0296pro01a)

arrayType[][] arrayName = {{row1 initializer}, {row2 initializer}, ...};

• Multidimensional arrays are maintained as arrays of separate one-dimensional arrays. As a result, the lengths of the rows in a two-dimensional array are not required to be the same.

• A multidimensional array with the same number of columns in every row can be created with an array-creation expression of the form

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0296pro02a)

*arrayType*[][] *arrayName* = new *arrayType*[*numRows*][*numColumns*];

#### Section 7.13 Variable-Length Argument Lists

• An argument type followed by an ellipsis (...; p. [281](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec13_html#page_281)) in a method’s parameter list indicates that the method receives a variable number of arguments of that particular type. The ellipsis can occur only once in a method’s parameter list. It must be at the end of the parameter list.

• A variable-length argument list (p. [281](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec13_html#page_281)) is treated as an array within the method body. The number of arguments in the array can be obtained using the array’s length field.

#### Section 7.14 Using Command-Line Arguments

• Passing arguments to main (p. [283](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec14_html#page_283)) from the command line is achieved by including a parameter of type String[] in the parameter list of main. By convention, main’s parameter is named args.

• Java passes command-line arguments that appear after the class name in the java command to the application’s mainmethod as Strings in the array args.

#### Section 7.15 Class Arrays

• Class Arrays (p. [285](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec15_html#page_285)) provides static methods that peform common array manipulations, including sort to sort an array,binarySearch to search a sorted array, equals to compare arrays and fill to place items in an array.

• Class System’s arraycopy method (p. [285](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec15_html#page_285)) enables you to copy the elements of one array into another.

#### Section 7.16 Introduction to Collections and ClassArrayList

• The Java API’s collection classes provide efficient methods that organize, store and retrieve data without requiring knowledge of how the data is being stored.

• An ArrayList<T> (p. [288](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec16_html#page_288)) is similar to an array but can be dynamically resized.

• The add method (p. [290](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec36_html#page_290)) with one argument appends an element to the end of an ArrayList.

• The add method with two arguments inserts a new element at a specified position in an ArrayList.

• The size method (p. [290](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec36_html#page_290)) returns the number of elements currently in an ArrayList.

• The remove method with a reference to an object as an argument removes the first element that matches the argument’s value.

• The remove method with an integer argument removes the element at the specified index, and all elements above that index are shifted down by one.

• The contains method returns true if the element is found in the ArrayList, and false otherwise.

### Self-Review Exercises

[**7.1**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec21_html#ch07ans1) Fill in the blank(s) in each of the following statements:

a) Lists and tables of values can be stored in \_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_.

b) An array is a group of \_\_\_\_\_\_\_\_ (called elements or components) containing values that all have the same \_\_\_\_\_\_\_.

c) The \_\_\_\_\_\_\_\_ allows you to iterate through an array’s elements without using a counter.

d) The number used to refer to a particular array element is called the element’s \_\_\_\_\_\_\_.

e) An array that uses two indices is referred to as a(n) \_\_\_\_\_\_\_\_ array.

f) Use the enhanced for statement \_\_\_\_\_\_\_\_ to walk through double array numbers.

g) Command-line arguments are stored in \_\_\_\_\_\_\_.

h) Use the expression \_\_\_\_\_\_\_\_ to receive the total number of arguments in a command line. Assume that command-line arguments are stored in String[] args.

i) Given the command java MyClass test, the first command-line argument is \_\_\_\_\_\_\_.

j) A(n) \_\_\_\_\_\_\_\_ in the parameter list of a method indicates that the method can receive a variable number of arguments.

[**7.2**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec21_html#ch07ans2) Determine whether each of the following is true or false. Iffalse, explain why.

a) An array can store many different types of values.

b) An array index should normally be of type float.

c) An individual array element that’s passed to a method and modified in that method will contain the modified value when the called method completes execution.

d) Command-line arguments are separated by commas.

[**7.3**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec21_html#ch07ans3) Perform the following tasks for an array called fractions:

a) Declare a constant ARRAY\_SIZE that’s initialized to 10.

b) Declare an array with ARRAY\_SIZE elements of typedouble, and initialize the elements to 0.

c) Refer to array element 4.

d) Assign the value 1.667 to array element 9.

e) Assign the value 3.333 to array element 6.

f) Sum all the elements of the array, using a for statement. Declare the integer variable x as a control variable for the loop.

[**7.4**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec21_html#ch07ans4) Perform the following tasks for an array called table:

a) Declare and create the array as an integer array that has three rows and three columns. Assume that the constantARRAY\_SIZE has been declared to be 3.

b) How many elements does the array contain?

c) Use a for statement to initialize each element of the array to the sum of its indices. Assume that the integer variablesx and y are declared as control variables.

[**7.5**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec21_html#ch07ans5) Find and correct the error in each of the following program segments:

a)

final int ARRAY\_SIZE = 5;  
ARRAY\_SIZE = 10;

b)

Assume int[] b = new int[10];  
for (int i = 0; i <= b.length; i++)  
   b[i] = 1;

c)

Assume int[][] a = {{1, 2}, {3, 4}};  
   a[1, 1] = 5;

### Answers to Self-Review Exercises

[**7.1**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec20_html#ch07que1)

a) arrays, collections.

b) variables, type.

c) enhanced for statement.

d) index (or subscript or position number).

e) two-dimensional.

f) for (double d : numbers).

g) an array of Strings, called args by convention.

h) args.length.

i) test.

j) ellipsis (...).

[**7.2**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec20_html#ch07que2)

a) False. An array can store only values of the same type.

b) False. An array index must be an integer or an integer expression.

c) For individual primitive-type elements of an array: False. A called method receives and manipulates a copy of the value of such an element, so modifications do not affect the original value. If the reference of an array is passed to a method, however, modifications to the array elements made in the called method are indeed reflected in the original. For individual elements of a reference type: True. A called method receives a copy of the reference of such an element, and changes to the referenced object will be reflected in the original array element.

d) False. Command-line arguments are separated by white space.

[**7.3**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec20_html#ch07que3)

a) final int ARRAY\_SIZE = 10;

b) double[] fractions = new double[ARRAY\_SIZE];

c) fractions[4]

d) fractions[9] = 1.667;

e) fractions[6] = 3.333;

f)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0298pro01a)

double total = 0.0;  
for (int x = 0; x < fractions.length; x++)  
   total += fractions[x];

[**7.4**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec20_html#ch07que4)

a) int[][] table = new int[ARRAY\_SIZE][ARRAY\_SIZE];

b) Nine.

c)

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0298pro02a)

for (int x = 0; x < table.length; x++)  
   for (int y = 0; y < table[x].length; y++)  
      table[x][y] = x + y;

[**7.5**](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec20_html#ch07que5)

a) Error: Assigning a value to a constant after it has been initialized. Correction: Assign the correct value to the constant in a final int ARRAY\_SIZE declaration or declare another variable.

b) Error: Referencing an array element outside the bounds of the array (b[10]). Correction: Change the <= operator to <.

c) Error: Array indexing is performed incorrectly.

Correction: Change the statement to a[1][1] = 5;.

### Exercises

**7.6** Fill in the blanks in each of the following statements:

a) One-dimensional array p contains four elements. The names of those elements are \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_.

b) Naming an array, stating its type and specifying the number of dimensions in the array is called \_\_\_\_\_\_\_\_ the array.

c) In a two-dimensional array, the first index identifies the \_\_\_\_\_\_\_\_ of an element and the second index identifies the \_\_\_\_\_\_\_\_ of an element.

d) An m-by-n array contains \_\_\_\_\_\_\_\_ rows, \_\_\_\_\_\_\_\_ columns and \_\_\_\_\_\_\_\_ elements.

e) The name of the element in row 3 and column 5 of array dis \_\_\_\_\_\_\_.

**7.7** Determine whether each of the following is true or false. Iffalse, explain why.

a) To refer to a particular location or element within an array, we specify the name of the array and the value of the particular element.

b) An array declaration reserves space for the array.

c) To indicate that 100 locations should be reserved for integer array p, you write the declaration

p[100];

d) An application that initializes the elements of a 15-element array to zero must contain at least one for statement.

e) An application that totals the elements of a two-dimensional array must contain nested for statements.

**7.8** Write Java statements to accomplish each of the following tasks:

a) Display the value of element 6 of array f.

b) Initialize each of the five elements of one-dimensional integer array g to 8.

c) Total the 100 elements of floating-point array c.

d) Copy 11-element array a into the first portion of array b, which contains 34 elements.

e) Determine and display the smallest and largest values contained in 99-element floatingpoint array w.

**7.9** Consider a two-by-three integer array t.

a) Write a statement that declares and creates t.

b) How many rows does t have?

c) How many columns does t have?

d) How many elements does t have?

e) Write access expressions for all the elements in row 1 of t.

f) Write access expressions for all the elements in column 2 oft.

g) Write a single statement that sets the element of t in row 0 and column 1 to zero.

h) Write individual statements to initialize each element of tto zero.

i) Write a nested for statement that initializes each element of t to zero.

j) Write a nested for statement that inputs the values for the elements of t from the user.

k) Write a series of statements that determines and displays the smallest value in t.

l) Write a single printf statement that displays the elements of the first row of t.

m) Write a statement that totals the elements of the third column of t. Do not use repetition.

n) Write a series of statements that displays the contents of tin tabular format. List the column indices as headings across the top, and list the row indices at the left of each row.

**7.10 (Sales Commissions)** Use a one-dimensional array to solve the following problem: A company pays its salespeople on a commission basis. The salespeople receive $200 per week plus 9% of their gross sales for that week. For example, a salesperson who grosses $5,000 in sales in a week receives $200 plus 9% of $5,000, or a total of $650. Write an application (using an array of counters) that determines how many of the salespeople earned salaries in each of the following ranges (assume that each salesperson’s salary is truncated to an integer amount):

a) $200-299

b) $300-399

c) $400-499

d) $500-599

e) $600-699

f) $700-799

g) $800-899

h) $900-999

i) $1,000 and over

Summarize the results in tabular format.

**7.11** Write statements that perform the following one-dimensional-array operations:

a) Set the 10 elements of integer array counts to zero.

b) Add one to each of the 15 elements of integer array bonus.

c) Display the five values of integer array bestScores in column format.

**7.12 (Duplicate Elimination)** Use a one-dimensional array to solve the following problem: Write an application that inputs five numbers, each between 10 and 100, inclusive. As each number is read, display it only if it’s not a duplicate of a number already read. Provide for the “worst case,” in which all five numbers are different. Use the smallest possible array to solve this problem. Display the complete set of unique values input after the user enters each new value.

**7.13** Label the elements of three-by-five two-dimensional arraysales to indicate the order in which they’re set to zero by the following program segment:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0300pro01a)

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

**7.14 (Variable-Length Argument List)** 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 (Command-Line Arguments)** Rewrite [Fig. 7.2](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec3_html#ch07fig02) 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 (Using the Enhanced** ***for*** **Statement)** Write an application that uses an enhanced for statement to sum thedouble values passed by the command-line arguments. [Hint: Use the static method parseDouble of class Double to convert aString 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 Randomonce 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 7.28](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig28) shows the 36 possible combinations of the two dice. Your application should roll the dice 36,000,000 times. Use a one-dimensional array to tally the number of times each possible sum appears. Display the results in tabular format.

**Fig. 7.28** | The 36 possible sums of two dice.

**7.18 (Game of Craps)** Write an application that runs 1,000,000 games of craps ([Fig. 6.8](http://proquest.safaribooksonline.com/9780133813036/ch06lev1sec10_html#ch06fig08)) and answers the following questions:

a) How many games are won on the first roll, second roll, ..., twentieth roll and after the twentieth roll?

b) How many games are lost on the first roll, second roll, ..., twentieth roll and after the twentieth roll?

c) What are the chances of winning at craps? [Note: You should discover that craps is one of the fairest casino games. What do you suppose this means?]

d) What is the average length of a game of craps?

e) Do the chances of winning improve with the length of the game?

**7.19 (Airline Reservations System)** A small airline has just purchased a computer for its new automated reservations system. You’ve been asked to develop the new system. You’re to write an application to assign seats on each flight of the airline’s only plane (capacity: 10 seats).

Your application should display the following alternatives:Please type 1 for First Class and Please type 2 for Economy. If the user types 1, your application should assign a seat in the firstclass section (seats 1–5). If the user types 2, your application should assign a seat in the economy section (seats 6–10). Your application should then display a boarding pass indicating the person’s seat number and whether it’s in the first-class or economy section of the plane.

Use a one-dimensional array of primitive type boolean to represent the seating chart of the plane. Initialize all the elements of the array to false to indicate that all the seats are empty. As each seat is assigned, set the corresponding element of the array to true to indicate that the seat is no longer available.

Your application should never assign a seat that has already been assigned. When the economy section is full, your application should ask the person if it’s acceptable to be placed in the first-class section (and vice versa). If yes, make the appropriate seat assignment. If no, display the message "Next flight leaves in 3 hours."

**7.20 (Total Sales)** Use a two-dimensional array to solve the following problem: A company has four salespeople (1 to 4) who sell five different products (1 to 5). Once a day, each salesperson passes in a slip for each type of product sold. Each slip contains the following:

a) The salesperson number

b) The product number

c) The total dollar value of that product sold that day

Thus, each salesperson passes in between 0 and 5 sales slips per day. Assume that the information from all the slips for last month is available. Write an application that will read all this information for last month’s sales and summarize the total sales by salesperson and by product. All totals should be stored in the two-dimensional array sales. After processing all the information for last month, display the results in tabular format, with each column representing a salesperson and each row representing a particular product. Cross-total each row to get the total sales of each product for last month. Cross-total each column to get the total sales by salesperson for last month. Your output should include these cross-totals to the right of the totaled rows and to the bottom of the totaled columns.

**7.21 (Turtle Graphics)** The Logo language made the concept ofturtle graphics famous. Imagine a mechanical turtle that walks around the room under the control of a Java application. The turtle holds a pen in one of two positions, up or down. While the pen is down, the turtle traces out shapes as it moves, and while the pen is up, the turtle moves about freely without writing anything. In this problem, you’ll simulate the operation of the turtle and create a computerized sketchpad.

Use a 20-by-20 array floor that’s initialized to zeros. Read commands from an array that contains them. Keep track of the current position of the turtle at all times and whether the pen is currently up or down. Assume that the turtle always starts at position (0, 0) of the floor with its pen up. The set of turtle commands your application must process are shown in [Fig. 7.29](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig29).

**Fig. 7.29** | Turtle graphics commands.

Suppose that the turtle is somewhere near the center of the floor. The following “program” would draw and display a 12-by-12 square, leaving the pen in the up position:

2  
5,12  
3  
5,12  
3  
5,12  
3  
5,12  
1  
6  
9

As the turtle moves with the pen down, set the appropriate elements of array floor to 1s. When the 6 command (display the array) is given, wherever there’s a 1 in the array, display an asterisk or any character you choose. Wherever there’s a 0, display a blank.

Write an application to implement the turtle graphics capabilities discussed here. Write several turtle graphics programs to draw interesting shapes. Add other commands to increase the power of your turtle graphics language.

**7.22 (Knight’s Tour)** An interesting puzzler for chess buffs is the Knight’s Tour problem, originally proposed by the mathematician Euler. Can the knight piece move around an empty chessboard and touch each of the 64 squares once and only once? We study this intriguing problem in depth here.

The knight makes only L-shaped moves (two spaces in one direction and one space in a perpendicular direction). Thus, as shown in [Fig. 7.30](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig30), from a square near the middle of an empty chessboard, the knight (labeled K) can make eight different moves (numbered 0 through 7).

**Fig. 7.30** | The eight possible moves of the knight.

a) Draw an eight-by-eight chessboard on a sheet of paper, and attempt a Knight’s Tour by hand. Put a 1 in the starting square, a 2 in the second square, a 3 in the third, and so on. Before starting the tour, estimate how far you think you’ll get, remembering that a full tour consists of 64 moves. How far did you get? Was this close to your estimate?

b) Now let’s develop an application that will move the knight around a chessboard. The board is represented by an eight-by-eight two-dimensional array board. Each square is initialized to zero. We describe each of the eight possible moves in terms of its horizontal and vertical components. For example, a move of type 0, as shown in [Fig. 7.30](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig30), consists of moving two squares horizontally to the right and one square vertically upward. A move of type 2 consists of moving one square horizontally to the left and two squares vertically upward. Horizontal moves to the left and vertical moves upward are indicated with negative numbers. The eight moves may be described by two one-dimensional arrays, horizontal and vertical, as follows:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p303pro02a)

horizontal[0] = 2           vertical[0] = -1  
horizontal[1] = 1           vertical[1] = -2  
horizontal[2] = -1          vertical[2] = -2  
horizontal[3] = -2          vertical[3] = -1  
horizontal[4] = -2          vertical[4] = 1  
horizontal[5] = -1          vertical[5] = 2  
horizontal[6] = 1           vertical[6] = 2  
horizontal[7] = 2           vertical[7] = 1

Let the variables currentRow and currentColumn indicate the row and column, respectively, of the knight’s current position. To make a move of type moveNumber, wheremoveNumber is between 0 and 7, your application should use the statements

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0303pro01a)

currentRow += vertical[moveNumber];  
currentColumn += horizontal[moveNumber];

Write an application to move the knight around the chessboard. Keep a counter that varies from 1 to 64. Record the latest count in each square the knight moves to. Test each potential move to see if the knight has already visited that square. Test every potential move to ensure that the knight does not land off the chessboard. Run the application. How many moves did the knight make?

c) After attempting to write and run a Knight’s Tour application, you’ve probably developed some valuable insights. We’ll use these insights to develop a heuristic (i.e., a common-sense rule) for moving the knight. Heuristics do not guarantee success, but a carefully developed heuristic greatly improves the chance of success. You may have observed that the outer squares are more troublesome than the squares nearer the center of the board. In fact, the most troublesome or inaccessible squares are the four corners.

Intuition may suggest that you should attempt to move the knight to the most troublesome squares first and leave open those that are easiest to get to, so that when the board gets congested near the end of the tour, there will be a greater chance of success.

We could develop an “accessibility heuristic” by classifying each of the squares according to how accessible it is and always moving the knight (using the knight’s L-shaped moves) to the most inaccessible square. We label a two-dimensional array accessibility with numbers indicating from how many squares each particular square is accessible. On a blank chessboard, each of the 16 squares nearest the center is rated as 8, each corner square is rated as 2, and the other squares have accessibility numbers of 3, 4 or 6 as follows:

2  3  4  4  4  4  3  2  
3  4  6  6  6  6  4  3  
4  6  8  8  8  8  6  4  
4  6  8  8  8  8  6  4  
4  6  8  8  8  8  6  4  
4  6  8  8  8  8  6  4  
3  4  6  6  6  6  4  3  
2  3  4  4  4  4  3  2

Write a new version of the Knight’s Tour, using the accessibility heuristic. The knight should always move to the square with the lowest accessibility number. In case of a tie, the knight may move to any of the tied squares. Therefore, the tour may begin in any of the four corners. [Note: As the knight moves around the chessboard, your application should reduce the accessibility numbers as more squares become occupied. In this way, at any given time during the tour, each available square’s accessibility number will remain equal to precisely the number of squares from which that square may be reached.] Run this version of your application. Did you get a full tour? Modify the application to run 64 tours, one starting from each square of the chessboard. How many full tours did you get?

d) Write a version of the Knight’s Tour application that, when encountering a tie between two or more squares, decides what square to choose by looking ahead to those squares reachable from the “tied” squares. Your application should move to the tied square for which the next move would arrive at a square with the lowest accessibility number.

**7.23 (Knight’s Tour: Brute-Force Approaches)** In part (c) of[Exercise 7.22](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que22), we developed a solution to the Knight’s Tour problem. The approach used, called the “accessibility heuristic,” generates many solutions and executes efficiently.

As computers continue to increase in power, we’ll be able to solve more problems with sheer computer power and relatively unsophisticated algorithms. Let’s call this approach “brute-force” problem solving.

a) Use random-number generation to enable the knight to walk around the chessboard (in its legitimate L-shaped moves) at random. Your application should run one tour and display the final chessboard. How far did the knight get?

b) Most likely, the application in part (a) produced a relatively short tour. Now modify your application to attempt 1,000 tours. Use a one-dimensional array to keep track of the number of tours of each length. When your application finishes attempting the 1,000 tours, it should display this information in neat tabular format. What was the best result?

c) Most likely, the application in part (b) gave you some “respectable” tours, but no full tours. Now let your application run until it produces a full tour. [Caution: This version of the application could run for hours on a powerful computer.] Once again, keep a 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 puzzler for chess buffs is the Eight Queens problem, which asks the following: Is it possible to place eight queens on an empty chessboard so that no queen is “attacking” any other (i.e., no two queens are in the same row, in the same column or along the same diagonal)? Use the thinking developed in [Exercise 7.22](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que22) to formulate a heuristic for solving the Eight Queens problem. Run your application. [Hint: It’s 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. 7.31](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig31). Once these “elimination numbers” are placed in all 64 squares, an appropriate heuristic might be as follows: Place the next queen in the square with the smallest elimination number. Why is this strategy intuitively appealing?]

**Fig. 7.31** | The 22 squares eliminated by placing a queen in the upper left corner.

**7.25 (Eight Queens: Brute-Force Approaches)** In this exercise, you’ll develop several brute-force approaches to solving the Eight Queens problem introduced in [Exercise 7.24](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que24).

a) Use the random brute-force technique developed in[Exercise 7.23](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que23) 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](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que22)), 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](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07que22) to test for a closed tour if a full tour has occurred.

**7.27 (Sieve of Eratosthenes)** A prime number is any integer greater than 1 that’s evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

a) Create a primitive-type boolean array with all elements initialized to true. Array elements with prime indices will remain true. All other array elements will eventually be set to false.

b) Starting with array index 2, determine whether a given element is true. If so, loop through the remainder of the array and set to false every element whose index is a multiple of the index for the element with value true. Then continue the process with the next element with valuetrue. For array index 2, all elements beyond element 2 in the array that have indices which are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to false; for array index 3, all elements beyond element 3 in the array that have indices which are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to false; and so on.

When this process completes, the array elements that are stilltrue indicate that the index is a prime number. These indices can be displayed. Write an application that uses an array of 1,000 elements to determine and display the prime numbers between 2 and 999. Ignore array elements 0 and 1.

**7.28 (Simulation: The Tortoise and the Hare)** In this problem, you’ll re-create the classic race of the tortoise and the hare. You’ll use random-number generation to develop a simulation of this memorable event.

Our contenders begin the race at square 1 of 70 squares. Each square represents a possible position along the race course. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground.

A clock ticks once per second. With each tick of the clock, your application should adjust the position of the animals according to the rules in [Fig. 7.32](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig32). Use variables to keep track of the positions of the animals (i.e., position numbers are 1–70). Start each animal at position 1 (the “starting gate”). If an animal slips left before square 1, move it back to square 1.

**Fig. 7.32** | Rules for adjusting the positions of the tortoise and the hare.

Generate the percentages in [Fig. 7.32](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec22_html#ch07fig32) by producing a random integer i in the range 1 ≤ i ≤ 10. For the tortoise, perform a “fast plod” when 1 ≤ i ≤ 5, a “slip” when 6 ≤ i ≤ 7 or a “slow plod” when 8 ≤ i ≤ 10. Use a similar technique to move the hare.

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. Yuch. If both animals win on the same tick of the clock, you may want to favor the tortoise (the “underdog”), or you may want to displayIt's a tie. If neither animal wins, perform the loop again to simulate the next tick of the clock. When you’re 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 doubleinstead of int to calculate and return Fibonacci numbers, and use this modified application to repeat part (b).

#### Exercises 7.30-7.34 are reasonably challenging. Once you’ve done them, 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](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec18_html#ch07fig11) to deal a five-card poker hand. Then modify classDeckOfCards of [Fig. 7.10](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec14_html#ch07fig10) 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](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec6_html#ch07fig09).]

**7.31 (Card Shuffling and Dealing)** Use the methods developed in [Exercise 7.30](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec53_html#ch07que30) to write an application that deals two five-card poker hands, evaluates each hand and determines which is better.

**7.32 (Project: Card Shuffling and Dealing)** Modify the application developed in [Exercise 7.31](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec53_html#ch07que31) 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 dealer should draw one, two or three more cards to replace the corresponding number of unneeded cards in the original hand. The application should then reevaluate the dealer’s hand. [Caution:This is a difficult problem!]

**7.33 (Project: Card Shuffling and Dealing)** Modify the application developed in [Exercise 7.32](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec53_html#ch07que32) so that it can handle the dealer’s hand automatically, but the player is allowed to decide which cards of the player’s hand to replace. The application should then evaluate both hands and determine who wins. Now use this new application to play 20 games against the computer. Who wins more games, you or the computer? Have a friend play 20 games against the computer. Who wins more games? Based on the results of these games, refine your poker-playing application. (This, too, is a difficult problem.) Play 20 more games. Does your modified application play a better game?

**7.34****(Project: Card Shuffling and Dealing)** Modify the application of [Figs. 7.9](http://proquest.safaribooksonline.com/9780133813036/ch07lev1sec6_html#ch07fig09)–[7.11](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec18_html#ch07fig11) to use Face and Suit enum types to represent the faces and suits of the cards. Declare each of theseenum types as a public type in its own source-code file. Each Cardshould have a Face and a Suit instance variable. These should be initialized by the Card constructor. In class DeckOfCards, create an array of Faces that’s initialized with the names of the constants in the Face enum type and an array of Suits that’s initialized with the names of the constants in the Suit enum type. [Note: When you output an enum constant as a String, the name of the constant is displayed.]

**7.35 (Fisher-Yates Shuffling Algorithm)** Research the Fisher-Yates shuffling algorithm online, then use it to reimplement theshuffle method in [Fig. 7.10](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec14_html#ch07fig10).

### Special Section: Building Your Own Computer

In the next several problems, we take a temporary diversion from the world of high-level language programming to “peel open” a computer and look at its internal structure. We introduce machine-language programming and write several machine-language programs. To make this an especially valuable experience, we then build a computer (through the technique of software-based simulation) on which you can execute your machine-language programs.

**7.36 (Machine-Language Programming)** Let’s create a computer called the Simpletron. As its name implies, it’s a simple machine, but powerful. The Simpletron runs programs written in the only language it directly understands: Simpletron Machine Language (SML).

The Simpletron contains an accumulator—a special register in which information is put before the Simpletron uses that information in calculations or examines it in various ways. All the information in the Simpletron is handled in terms of words. A word is a signed four-digit decimal number, such as +3364, -1293, +0007and -0001. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers 00, 01, ..., 99.

Before running an SML program, we must load, or place, the program into memory. The first instruction (or statement) of every SML program is always placed in location 00. The simulator will start executing at this location.

Each instruction written in SML occupies one word of the Simpletron’s memory (so instructions are signed four-digit decimal numbers). We shall assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron’s memory may contain an instruction, a data value used by a program or an unused (and so undefined) area of memory. The first two digits of each SML instruction are the operation code specifying the operation to be performed. SML operation codes are summarized in [Fig. 7.33](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig33).

**Fig. 7.33** | Simpletron Machine Language (SML) operation codes.

The last two digits of an SML instruction are the operand—the address of the memory location containing the word to which the operation applies. Let’s consider several simple SML programs.

The first SML program ([Fig. 7.34](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig34)) reads two numbers from the keyboard and computes and displays their sum. The instruction+1007 reads the first number from the keyboard and places it into location 07 (which has been initialized to 0). Then instruction+1008 reads the next number into location 08. The loadinstruction, +2007, puts the first number into the accumulator, and the add instruction, +3008, adds the second number to the number in the accumulator. All SML arithmetic instructions leave their results in the accumulator. The store instruction, +2109, places the result back into memory location 09, from which thewrite instruction, +1109, takes the number and displays it (as a signed four-digit decimal number). The halt instruction, +4300, terminates execution.

**Fig. 7.34** | SML program that reads two integers and computes their sum.

The second SML program ([Fig. 7.35](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig35)) reads two numbers from the keyboard and determines and displays the larger value. Note the use of the instruction +4107 as a conditional transfer of control, much the same as Java’s if statement.

**Fig. 7.35** | SML program that reads two integers and determines the larger.

Now write SML programs to accomplish each of the following tasks:

a) Use a sentinel-controlled loop to read 10 positive numbers. Compute and display their sum.

b) Use a counter-controlled loop to read seven numbers, some positive and some negative, and compute and display their average.

c) Read a series of numbers, and determine and display the largest number. The first number read indicates how many numbers should be processed.

**7.37 (Computer Simulator)** In this problem, you’re going to build your own computer. No, you’ll not be soldering components together. Rather, you’ll use the powerful technique of software-based simulation to create an object-oriented software model of the Simpletron of [Exercise 7.36](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07que36). Your Simpletron simulator will turn the computer you’re using into a Simpletron, and you’ll actually be able to run, test and debug the SML programs you wrote in [Exercise 7.36](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07que36).

When you run your Simpletron simulator, it should begin by displaying:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0310pro01a)

\*\*\* Welcome to Simpletron! \*\*\*  
\*\*\* Please enter your program one instruction    \*\*\*  
\*\*\* (or data word) at a time. I will display     \*\*\*  
\*\*\* the location number and a question mark (?). \*\*\*  
\*\*\* You then type the word for that location.    \*\*\*  
\*\*\* Type -99999 to stop entering your program.   \*\*\*

Your application should simulate the memory of the Simpletron with a one-dimensional array memory that has 100 elements. Now assume that the simulator is running, and let’s examine the dialog as we enter the program of [Fig. 7.35](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig35) ([Exercise 7.36](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07que36)):

00 ? +1009  
01 ? +1010  
02 ? +2009  
03 ? +3110  
04 ? +4107  
05 ? +1109  
06 ? +4300  
07 ? +1110  
08 ? +4300  
09 ? +0000  
10 ? +0000  
11 ? -99999

Your program should display the memory location followed by a question mark. Each value to the right of a question mark is input by the user. When the sentinel value -99999 is input, the program should display the following:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0311pro01a)

\*\*\* Program loading completed \*\*\*  
\*\*\* Program execution begins  \*\*\*

The SML program has now been placed (or loaded) in arraymemory. Now the Simpletron executes the SML program. Execution begins with the instruction in location 00 and, as in Java, continues sequentially, unless directed to some other part of the program by a transfer of control.

Use the variable accumulator to represent the accumulator register. Use the variable instructionCounter to keep track of the location in memory that contains the instruction being performed. Use the variable operationCode to indicate the operation currently being performed (i.e., the left two digits of the instruction word). Use the variable operand to indicate the memory location on which the current instruction operates. Thus,operand is the rightmost two digits of the instruction currently being performed. Do not execute instructions directly from memory. Rather, transfer the next instruction to be performed from memory to a variable called instructionRegister. Then pick off the left two digits and place them in operationCode, and pick off the right two digits and place them in operand. When the Simpletron begins execution, the special registers are all initialized to zero.

Now, let’s walk through execution of the first SML instruction,+1009 in memory location 00. This procedure is called aninstruction-execution cycle.

The instructionCounter tells us the location of the next instruction to be performed. We fetch the contents of that location from memory by using the Java statement

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0311pro02a)

instructionRegister = memory[instructionCounter];

The operation code and the operand are extracted from the instruction register by the statements

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0311pro03a)

operationCode = instructionRegister / 100;  
operand = instructionRegister % 100;

Now the Simpletron must determine that the operation code is actually a read (versus a write, a load, and so on). A switchdifferentiates among the 12 operations of SML. In the switchstatement, the behavior of various SML instructions is simulated as shown in [Fig. 7.36](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig36). We discuss branch instructions shortly and leave the others to you.

**Fig. 7.36** | Behavior of several SML instructions in the Simpletron.

When the SML program completes execution, the name and contents of each register as well as the complete contents of memory should be displayed. Such a printout is often called a computer dump (no, a computer dump is not a place where old computers go). To help you program your dump method, a sample dump format is shown in [Fig. 7.37](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07fig37). A dump after executing a Simpletron program would show the actual values of instructions and data values at the moment execution terminated.

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p07fig37a)

REGISTERS:  
accumulator             +0000  
instructionCounter         00  
instructionRegister     +0000  
operationCode              00  
operand                    00  
MEMORY:  
       0     1     2     3     4     5     6     7     8     9  
 0 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
10 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
20 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
30 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
40 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
50 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
60 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
70 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
80 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000  
90 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000 +0000

**Fig. 7.37** | A sample dump.

Let’s proceed with the execution of our program’s first instruction—namely, the +1009 in location 00. As we’ve indicated, the switch statement simulates this task by prompting the user to enter a value, reading the value and storing it in memory location memory[operand]. The value is then read into location 09.

At this point, simulation of the first instruction is completed. All that remains is to prepare the Simpletron to execute the next instruction. Since the instruction just performed was not a transfer of control, we need merely increment the instruction-counter register as follows:

instructionCounter++;

This action completes the simulated execution of the first instruction. The entire process (i.e., the instruction-execution cycle) begins anew with the fetch of the next instruction to execute.

Now let’s consider how the branching instructions—the transfers of control—are simulated. All we need to do is adjust the value in the instruction counter appropriately. Therefore, the unconditional branch instruction (40) is simulated within theswitch as

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p312pro01a)

instructionCounter = operand;

The conditional “branch if accumulator is zero” instruction is simulated as

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p312pro02a)

if (accumulator == 0)  
   instructionCounter = operand;

At this point, you should implement your Simpletron simulator and run each of the SML programs you wrote in [Exercise 7.36](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07que36). If you desire, you may embellish SML with additional features and provide for these features in your simulator.

Your simulator should check for various types of errors. During the program-loading phase, for example, each number the user types into the Simpletron’s memory must be in the range -9999 to+9999. Your simulator should test that each number entered is in this range and, if not, keep prompting the user to re-enter the number until the user enters a correct number.

During the execution phase, your simulator should check for various serious errors, such as attempts to divide by zero, attempts to execute invalid operation codes, and accumulator overflows (i.e., arithmetic operations resulting in values larger than +9999 or smaller than -9999). Such serious errors are calledfatal errors. When a fatal error is detected, your simulator should display an error message, such as

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0313pro01a)

\*\*\* Attempt to divide by zero \*\*\*  
\*\*\* Simpletron execution abnormally terminated \*\*\*

and should display a full computer dump in the format we discussed previously. This treatment will help the user locate the error in the program.

**7.38 (Simpletron Simulator Modifications)** In [Exercise 7.37](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec54_html#ch07que37), you wrote a software simulation of a computer that executes programs written in Simpletron Machine Language (SML). In this exercise, we propose several modifications and enhancements to the Simpletron Simulator. In the exercises of [Chapter 21](http://proquest.safaribooksonline.com/9780133813036/ch21_html#ch21), we propose building a compiler that converts programs written in a high-level programming language (a variation of Basic) to Simpletron Machine Language. Some of the following modifications and enhancements may be required to execute the programs produced by the compiler:

a) Extend the Simpletron Simulator’s memory to contain 1,000 memory locations to enable the Simpletron to handle larger programs.

b) Allow the simulator to perform remainder calculations. This modification requires an additional SML instruction.

c) Allow the simulator to perform exponentiation calculations. This modification requires an additional SML instruction.

d) Modify the simulator to use hexadecimal rather than integer values to represent SML instructions.

e) Modify the simulator to allow output of a newline. This modification requires an additional SML instruction.

f) Modify the simulator to process floating-point values in addition to integer values.

g) Modify the simulator to handle string input. [Hint: Each Simpletron word can be divided into two groups, each holding a two-digit integer. Each two-digit integer represents the ASCII (see [Appendix B](http://proquest.safaribooksonline.com/9780133813036/app02_html#app02)) decimal equivalent of a character. Add a machine language instruction that will input a string and store the string, beginning at a specific Simpletron memory location. The first half of the word at that location will be a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction converts each character into its ASCII equivalent and assigns it to a half-word.]

h) Modify the simulator to handle output of strings stored in the format of part (g). [Hint: Add a machine-language instruction that will display a string, beginning at a certain Simpletron memory location. The first half of the word at that location is a count of the number of characters in the string (i.e., the length of the string). Each succeeding half-word contains one ASCII character expressed as two decimal digits. The machine-language instruction checks the length and displays the string by translating each two-digit number into its equivalent character.]

**7.39 (Enhanced** ***GradeBook*)** Modify the GradeBook class of [Fig. 7.18](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec27_html#ch07fig18) so that the constructor accepts as parameters the number of students and the number of exams, then builds an appropriately sized two-dimensional array, rather than receiving a preinitialized two-dimensional array as it does now. Set each element of the new two-dimensional array to -1 to indicate that no grade has been entered for that element. Add a setGrade method that sets one grade for a particular student on a particular exam. Modify class GradeBookTest of [Fig. 7.19](http://proquest.safaribooksonline.com/9780133813036/ch07lev2sec29_html#ch07fig19) to input the number of students and number of exams for the GradeBook and to allow the instructor to enter one grade at a time.

### Making a Difference

**7.40 (Polling)** The Internet and the web are enabling more people to network, join a cause, voice opinions, and so on. Recent presidential candidates have used the Internet intensively to get out their messages and raise money for their campaigns. In this exercise, you’ll write a simple polling program that allows users to rate five social-consciousness issues from 1 (least important) to 10 (most important). Pick five causes that are important to you (e.g., political issues, global environmental issues). Use a one-dimensional array topics (of type String) to store the five causes. To summarize the survey responses, use a 5-row, 10-column two-dimensional array responses (of type int), each row corresponding to an element in the topics array. When the program runs, it should ask the user to rate each issue. Have your friends and family respond to the survey. Then have the program display a summary of the results, including:

a) A tabular report with the five topics down the left side and the 10 ratings across the top, listing in each column the number of ratings received for each topic.

b) To the right of each row, show the average of the ratings for that issue.

c) Which issue received the highest point total? Display both the issue and the point total.

d) Which issue received the lowest point total? Display both the issue and the point total.