Now go, write it before them in a table, and note it in a book.
—Isaiah 30:8

To go beyond is as wrong as to fall short.
—Confucius

Begin at the beginning, … and go on till you come to the end: then stop.
—Lewis Carroll

7

Arrays and ArrayLists

Objectives

In this Chapter you'll learn:

- What arrays are.
- To use arrays to store data in and retrieve data from lists and tables of values.
- To declare arrays, initialize arrays and refer to individual elements of arrays.
- To use the enhanced `for` statement to iterate through arrays.
- To pass arrays to methods.
- To declare and manipulate multidimensional arrays.
- To write methods that use variable-length argument lists.
- To read command-line arguments into a program.

Self-Review Exercises

7.1 Fill in the blank(s) in each of the following statements:

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

ANS: arrays.

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

ANS: variables, type.

c) The _____ allows programmers to iterate through the elements in an array without using a counter.

ANS: enhanced `for` statement.

d) The number used to refer to a particular array element is called the element's _____.

ANS: index (or subscript or position number).

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

ANS: two-dimensional.

f) Use the enhanced `for` statement _____ to walk through `double` array `numbers`.

ANS: `for ( double d : numbers )`.

g) Command-line arguments are stored in _____.

ANS: an array of `String`s, called `args` by convention.

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`.

ANS: `args.length`.

i) Given the command `java MyClass test`, the first command-line argument is _____.

ANS: `test`.

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

ANS: ellipsis (. . .)

7.2 Determine whether each of the following is *true* or *false*. If *false*, explain why.

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

ANS: False. An array can store only values of the same type.

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

ANS: False. An array index must be an integer or an integer expression.

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

ANS: 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) Command-line arguments are separated by commas.

ANS: False. Command-line arguments are separated by white space.

7.3 Perform the following tasks for an array called `fractions`:

a) Declare a constant `ARRAY_SIZE` that is initialized to 10.

ANS:
```
final int ARRAY_SIZE = 10;
```

b) Declare an array with `ARRAY_SIZE` elements of type `double`, and initialize the elements to `0`.

ANS:
```
double[] fractions = new double[ ARRAY_SIZE ];
```

c) Refer to array element 4.

ANS:
```
fractions[ 4 ]
```

d) Assign the value `1.667` to array element 9.

ANS: `fractions[ 9 ] = 1.667;`

e) Assign the value `3.333` to array element 6.

ANS: `fractions[ 6 ] = 3.333;`

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

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

7.4 Perform the following tasks for an array called `table`:

a) Declare and create the array as an integer array that has three rows and three columns. Assume that the constant `ARRAY_SIZE` has been declared to be `3`.

ANS: `int[][] table = new int[ ARRAY_SIZE ][ ARRAY_SIZE ];`

b) How many elements does the array contain?

ANS: Nine.

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

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

7.5 Find and correct the error in each of the following program segments:

a)
```
final int ARRAY_SIZE = 5;
ARRAY_SIZE = 10;
```

ANS: Error: Assigning a value to a constant after it has been initialized.
Correction: Assign the correct value to the constant in a `final int ARRAY_SIZE` declaration or declare another variable.

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

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

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

ANS: Error: Array indexing is performed incorrectly.
Correction: Change the statement to `a[ 1 ][ 1 ] = 5;`.

Exercises

NOTE: Solutions to the programming exercises are located in the `ch07solutions` folder. Each exercise has its own folder named `ex07_##` where ## is a two-digit number representing the exercise number. For example, exercise 7.10's solution is located in the folder `ex07_10`.

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 _____.

ANS: `p[ 0 ]`, `p[ 1 ]`, `p[ 2 ]`, and `p[ 3 ]`

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

ANS: declaring

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

ANS: row, column

d) An *m*-by-*n* array contains _____ rows, _____ columns and _____ elements.

ANS: `m`, `n`, `m * n`

e) The name of the element in row 3 and column 5 of array `d` is _____.

ANS: `d[ 3 ][ 5 ]`

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

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

ANS: False. The name of the array and the index are specified.

b) An array declaration reserves space for the array.

ANS: False. Arrays must be dynamically allocated with `new` in Java.

c) To indicate that 100 locations should be reserved for integer array `p`, the programmer writes the declaration

```
p[ 100 ];
```

ANS: False. The correct declaration is `int p[] = new int[ 100 ];`

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

ANS: False. Numeric arrays are automatically initialized to zero. Also, a member initializer list can be used.

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

ANS: False. It is possible to total the elements of a two-dimensional array with nested `while` statements, nested `do...while` statements or even individual totaling statements.

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

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

ANS: `System.out.print( f[ 6 ] );`

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

ANS: `int g[] = { 8, 8, 8, 8, 8 );`

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

ANS:

```
for ( int k = 0; k < c.length; k++ )
   total += c[ k ];
```

```
1  // Exercise 7.8c Solution: PartC.java
2  public class PartC
3  {
4     public static void main( String args[] )
5     {
6        double c[] = new double[ 100 ];
7        double total = 0;
8
9        // c)
10       for ( int k = 0; k < c.length; k++ )
11          total += c[ k ];
12    } // end main
13 } // end class PartC
```

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

ANS:
```
for ( int j = 0; j < a.length; j++ )
   b[ j ] = a[ j ];
```

```
1  // Exercise 7.8d Solution: PartD.java
2  public class PartD
3  {
4     public static void main( String args[] )
5     {
6        double a[] = new double[ 11 ];
7        double b[] = new double[ 34 ];
8
9        // d)
10       for ( int j = 0; j < a.length; j++ )
11          b[ j ] = a[ j ];
12    } // end main
13 } // end class PartD
```

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

ANS:

```
1  // Exercise 7.8e Solution: PartE.java
2  public class PartE
3  {
4     public static void main( String args[] )
5     {
6        double w[] = new double[ 99 ];
7        double small = w[ 0 ];
8        double large = w[ 0 ];
9
10       // e)
11       for ( int i = 0;  i < w.length; i++ )
12          if ( w[ i ] < small )
13             small = w[ i ];
14          else if ( w[ i ] > large )
15             large = w[ i ];
16
17       System.out.printf( "%f   %f\n", small, large );
18    } // end main
19 } // end class PartE
```

```
0.000000   0.000000
```

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

a) Write a statement that declares and creates t.

ANS: `int t[][] = new int[ 2 ][ 3 ];`

b) How many rows does t have?

ANS: two.

c) How many columns does t have?

ANS: three.

d) How many elements does t have?

ANS: six.

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

ANS: `t[ 1 ][ 0 ], t[ 1 ][ 1 ], t[ 1 ][ 2 ]`

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

ANS: `t[ 0 ][ 2 ], t[ 1 ][ 2 ]`

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

ANS:
```
t[ 0 ][ 1 ] = 0;
```

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

ANS:
```
t[ 0 ][ 0 ] = 0;
t[ 0 ][ 1 ] = 0;
t[ 0 ][ 2 ] = 0;
t[ 1 ][ 0 ] = 0;
t[ 1 ][ 1 ] = 0;
t[ 1 ][ 2 ] = 0;
```

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

ANS:
```
for ( int j = 0; j < t.length; j++ )
   for ( int k = 0; k < t[ j ].length; k++ )
      t[ j ][ k ] = 0;
```

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

ANS:
```
for ( int j = 0; j < t.length; j++ )
   for ( int k = 0; k < t[ j ].length; k++ )
      t[ j ][ k ] = input.nextInt();
```

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

ANS:
```
int smallest = t[ 0 ][ 0 ];

for ( int j = 0; j < t.length; j++ )
   for ( int k = 0; k < t[ j ].length; k++ )
      if ( t[ x ][ y ] < smallest )
         smallest = t[ x ][ y ];

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

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

ANS:
```
System.out.printf( "%d %d %d\n", t[ 0 ][ 0 ], t[ 0 ][ 1 ], t[ 0 ][ 2 ] );
```

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

ANS:
```
int total = t[ 0 ][ 2 ] + t[ 1 ][ 2 ];
```

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

ANS:
```
System.out.println( "\t0\t1\t2\n" );

for ( int e = 0; e < t.length; e++ )
{
   System.out.print( e );
   for ( int r = 0; r < t[ e ].length; r++ )
      System.out.printf( "\t%d", t[ e ][ r ] );

   System.out.println();
} // end for
```

```
1  // Exercise 7.9 Solution: Array.java
2  import java.util.Scanner;
3
4  public class Array
5  {
6     public static void main( String args[] )
7     {
8        Scanner input = new Scanner( System.in );
9
10       // a)
11       int t[][] = new int[ 2 ][ 3 ];
12
13       // g)
14       t[ 0 ][ 1 ] = 0;
15
16       // h)
17       t[ 0 ][ 0 ] = 0;
18       t[ 0 ][ 1 ] = 0;
19       t[ 0 ][ 2 ] = 0;
20       t[ 1 ][ 0 ] = 0;
21       t[ 1 ][ 1 ] = 0;
22       t[ 1 ][ 2 ] = 0;
23
24       // i)
25       for ( int j = 0; j < t.length; j++ )
26          for ( int k = 0; k < t[ j ].length; k++ )
27             t[ j ][ k ] = 0;
28
29       // j)
30       for ( int j = 0; j < t.length; j++ )
31          for ( int k = 0; k < t[ j ].length; k++ )
32             t[ j ][ k ] = input.nextInt();
33
34       // k)
35       int small = t[ 0 ][ 0 ];
36
37       for ( int j = 0; j < t.length; j++ )
38          for ( int k = 0; k < t[ j ].length; k++ )
39             if ( t[ j ][ k ] < small )
40                small = t[ j ][ k ];
41
42       System.out.println( small );
43
44       // l)
45       System.out.printf(
46          "%d %d %d\n", t[ 0 ][ 0 ], t[ 0 ][ 1 ], t[ 0 ][ 2 ] );
47
48       // m
49       int total = t[ 0 ][ 2 ] + t[ 1 ][ 2 ];
50
51       // n
52       System.out.println( "\t0\t1\t2\n" );
53       for ( int e = 0; e < t.length; e++ )
54       {
55          System.out.print( e );
56
```

```
57            for ( int r = 0; r < t[ e ].length; r++ )
58               System.out.printf( "\t%d", t[ e ][ r ] );
59
60            System.out.println();
61         } // end for
62      } // end main
63   } // end class Array
```

```
1
2
3
4
5
6
1
1 2 3
         0       1       2

0        1       2       3
1        4       5       6
```

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

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

ANS:
```
for ( int u = 0; u < counts.length; u++ )
   counts[ u ] = 0;
```

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

ANS:
```
for ( int v = 0; v < bonus.length; v++ )
   bonus[ v ]++;
```

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

ANS:
```
for ( int w = 0; w < bestScores.length; w++ )
   System.out.println( bestScores[ w ] );
```

7.13 Label the elements of three-by-five two-dimensional array `sales` to indicate the order in which they are set to zero by the following program segment:

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

ANS:
```
sales[ 0 ][ 0 ], sales[ 0 ][ 1 ], sales[ 0 ][ 2 ], sales[ 0 ][ 3 ],
sales[ 0 ][ 4 ], sales[ 1 ][ 0 ], sales[ 1 ][ 1 ], sales[ 1 ][ 2 ],
sales[ 1 ][ 3 ], sales[ 1 ][ 4 ], sales[ 2 ][ 0 ], sales[ 2 ][ 1 ],
sales[ 2 ][ 2 ], sales[ 2 ][ 3 ], sales[ 2 ][ 4 ]
```

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.

7.35 ***(Machine-Language Programming)*** Let's create a computer called the Simpletron. As its name implies, it's a simple, but powerful, machine. 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`, `+0007` and `-0001`. The Simpletron is equipped with a 100-word memory, and these words are referenced by their location numbers `00`, `01`, …, `99`.

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

Each instruction written in SML occupies one word of the Simpletron's memory (and hence instructions are signed four-digit decimal numbers). We shall assume that the sign of an SML instruction is always plus, but the sign of a data word may be either plus or minus. Each location in the Simpletron's memory may contain an instruction, a data value used by a program or an unused (and hence undefined) area of memory. The first two digits of each SML instruction are the *operation code* specifying the operation to be performed. SML operation codes are summarized in Fig. 7.33.

Operation code	Meaning
Input/output operations:	
`final int READ = 10;`	Read a word from the keyboard into a specific location in memory.
`final int WRITE = 11;`	Write a word from a specific location in memory to the screen.
Load/store operations:	
`final int LOAD = 20;`	Load a word from a specific location in memory into the accumulator.
`final int STORE = 21;`	Store a word from the accumulator into a specific location in memory.
Arithmetic operations:	
`final int ADD = 30;`	Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).
`final int SUBTRACT = 31;`	Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator).
`final int DIVIDE = 32;`	Divide a word from a specific location in memory into the word in the accumulator (leave result in the accumulator).
`final int MULTIPLY = 33;`	Multiply a word from a specific location in memory by the word in the accumulator (leave the result in the accumulator).
Transfer-of-control operations:	
`final int BRANCH = 40;`	Branch to a specific location in memory.
`final int BRANCHNEG = 41;`	Branch to a specific location in memory if the accumulator is negative.

Fig. 7.33 | Simpletron Machine Language (SML) operation codes. (Part 1 of 2.)

Operation code	Meaning
`final int BRANCHZERO = 42;`	Branch to a specific location in memory if the accumulator is zero.
`final int HALT = 43;`	Halt. The program has completed its task.

Fig. 7.33 | Simpletron Machine Language (SML) operation codes. (Part 2 of 2.)

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) 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 *load* instruction, `+2007`, puts the first number into the accumulator, and the *add* instruction, `+3008`, adds the second number to the number in the accumulator. *All SML arithmetic instructions leave their results in the accumulator.* The *store* instruction, `+2109`, places the result back into memory location `09`, from which the *write* instruction, `+1109`, takes the number and displays it (as a signed four-digit decimal number). The *halt* instruction, `+4300`, terminates execution.

Location	Number	Instruction
00	+1007	(Read A)
01	+1008	(Read B)
02	+2007	(Load A)
03	+3008	(Add B)
04	+2109	(Store C)
05	+1109	(Write C)
06	+4300	(Halt)
07	+0000	(Variable A)
08	+0000	(Variable B)
09	+0000	(Result C)

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

The second SML program (Fig. 7.35) 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.

Location	Number	Instruction
00	+1009	(Read A)
01	+1010	(Read B)
02	+2009	(Load A)
03	+3110	(Subtract B)

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

Location	Number	Instruction
04	+4107	(Branch negative to 07)
05	+1109	(Write A)
06	+4300	(Halt)
07	+1110	(Write B)
08	+4300	(Halt)
09	+0000	(Variable A)
10	+0000	(Variable B)

Fig. 7.35 | 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.

ANS: Note: This program terminates when a negative number is input. The problem statement should state that only positive numbers should be input.

```
00  +1009       (Read Value)
01  +2009       (Load Value)
02  +4106       (Branch negative to 06)
03  +3008       (Add Sum)
04  +2108       (Store Sum)
05  +4000       (Branch 00)
06  +1108       (Write Sum)
07  +4300       (Halt)
08  +0000       (Storage for Sum)
09  +0000       (Storage for Value)
```

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

ANS:

```
00  +2018       (Load Counter)
01  +3121       (Subtract Termination)
02  +4211       (Branch zero to 11)
03  +2018       (Load Counter)
04  +3019       (Add Increment)
05  +2118       (Store Counter)
06  +1017       (Read Value)
07  +2016       (Load Sum)
08  +3017       (Add Value)
09  +2116       (Store Sum)
10  +4000       (Branch 00)
11  +2016       (Load Sum)
12  +3218       (Divide Counter)
13  +2120       (Store Result)
```

```
14  +1120       (Write Result)
15  +4300       (Halt)
16  +0000       (Variable Sum)
17  +0000       (Variable Value)
18  +0000       (Variable Counter)
19  +0001       (Variable Increment)
20  +0000       (Variable Result)
21  +0007       (Variable Termination)
```

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

ANS:

```
00  +1017       (Read Endvalue)
01  +2018       (Load Counter)
02  +3117       (Subtract Endvalue)
03  +4215       (Branch zero to 15)
04  +2018       (Load Counter)
05  +3021       (Add Increment)
06  +2118       (Store Counter)
07  +1019       (Read Value)
08  +2020       (Load Largest)
09  +3119       (Subtract Value)
10  +4112       (Branch negative to 12)
11  +4001       (Branch 01)
12  +2019       (Load Value)
13  +2120       (Store Largest)
14  +4001       (Branch 01)
15  +1120       (Write Largest)
16  +4300       (Halt)
17  +0000       (Variable EndValue)
18  +0000       (Variable Counter)
19  +0000       (Variable Value)
20  +0000       (Variable Largest)
21  +0001       (Variable Increment)
```