7

Arrays

Introducing
JAVA SE 6 (Mustang) • Web Services •
Ajax-Enabled Web Applications •
JavaServer Faces

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

OBJECTIVES

In this chapter you will 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.

Student Solution Exercises

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

a) One-dimensional array `p` contains four elements. The array-access expressions for 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.

7.9 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 a series of statements that initializes each element of `t` to zero. Do not use a repetition statement.

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

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

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

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

ANS:
```
for ( int 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 `printf` statement that displays the elements of the first row of `t`. Do not use repetition.

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
```

```
// Exercise 7.9 Solution: Array.java
import java.util.Scanner;

public class Array
{
   public static void main( String args[] )
   {
      Scanner input = new Scanner( System.in );

      // a)
      int t[][] = new int[ 2 ][ 3 ];

      // g)
      t[ 0 ][ 1 ] = 0;

      // h)
      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)
      for ( int j = 0; j < t.length; j++ )
         for ( int k = 0; k < t[ j ].length; k++ )
            t[ j ][ k ] = 0;

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

      // k)
      int small = t[ 0 ][ 0 ];

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

      System.out.println( small );

      // l)
      System.out.printf(
         "%d %d %d\n", t[ 0 ][ 0 ], t[ 0 ][ 1 ], t[ 0 ][ 2 ] );

      // m
      int total = t[ 0 ][ 2 ] + t[ 1 ][ 2 ];

      // n
      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
   } // end main
} // end class Array
```

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

0       1       2       3
1       4       5       6
```

7.10 *(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 $5000 in sales in a week receives $200 plus 9% of $5000, 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) $1000 and over

Summarize the results in tabular format.

ANS:

```java
// Exercise 7.10 Solution: Sales.java
// Program calculates the amount of pay for a salesperson and counts the
// number of salespeople that earned salaries in given ranges.
import java.util.Scanner;

public class Sales
{
   // counts the number of people in given salary ranges
   public void countRanges()
   {
      Scanner input = new Scanner( System.in );

      int total[] = new int[ 9 ]; // totals for the various salaries

      // initialize the values in the array to zero
      for ( int counter = 0; counter < total.length; counter++ )
         total[ counter ] = 0;

      // read in values and assign them to the appropriate range
      System.out.print( "Enter sales amount (negative to end): " );
      double dollars = input.nextDouble();

      while ( dollars >= 0 )
      {
         double salary = dollars * 0.09 + 200;
         int range = ( int ) ( salary / 100 );

         if ( range > 10 )
            range = 10;

         ++total[ range - 2 ];

         System.out.print( "Enter sales amount (negative to end): " );
         dollars = input.nextDouble();
      } // end while

      // print chart
      System.out.println( "Range\t\tNumber" );

      for ( int range = 0; range < total.length - 1; range++ )
         System.out.printf( "$%d-$%d\t%d\n",
            (200 + 100 * range), (299 + 100 * range), total[ range ] );

      // special case for the last range
      System.out.printf( "$1000 and over\t%d\n",
         total[ total.length - 1 ] );
   } // end method countRanges
} // end class Sales
```

```java
// Exercise 7.10 Solution: SalesTest.java
// Test application for class Sales
public class SalesTest
{
   public static void main( String args[] )
   {
      Sales application = new Sales();
      application.countRanges();
   } // end main
} // end class SalesTest
```

```
Enter sales amount (negative to end): 5000
Enter sales amount (negative to end): -1
Range           Number
$200-$299       0
$300-$399       0
$400-$499       0
$500-$599       0
$600-$699       1
$700-$799       0
$800-$899       0
$900-$999       0
$1000 and over  0
```

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 ]`

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.

ANS:

```
// Exercise 7.14 Solution: VarargsTest.java
// Using variable-length argument lists.

public class VarargsTest
{
   // multiply numbers
   public static int product( int... numbers )
   {
      int product = 1;

      // process variable-length argument list
      for ( int number : numbers )
         product *= number;

      return product;
   } // end method product

   public static void main( String args[] )
   {
      // values to multiply
      int a = 1;
      int b = 2;
      int c = 3;
      int d = 4;
      int e = 5;

      // display integer values
      System.out.printf( "a = %d, b = %d, c = %d, d = %d, e = %d\n\n",
         a, b, c, d, e );

      // call product with different number of arguments in each call
      System.out.printf( "The product of a and b is: %d\n",
         product( a, b ) );
      System.out.printf( "The product of a, b and c is: %d\n",
         product( a, b, c ) );
      System.out.printf( "The product of a, b, c and d is: %d\n",
         product( a, b, c, d ) );
      System.out.printf( "The product of a, b, c, d and e is: %d\n",
         product( a, b, c, d, e ) );
   } // end main
} // end class VarargsTest
```

```
a = 1, b = 2, c = 3, d = 4, e = 5

The product of a and b is: 2
The product of a, b and c is: 6
The product of a, b, c and d is: 24
The product of a, b, c, d and e is: 120
```

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).

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Fig. 7.30 | The 36 possible sums of two dice.

ANS:

```
// Exercise 7.17 Solution: Roll36.java
// Program simulates rolling two six-sided dice 36,000 times.
import java.util.Random;

public class Roll36
{
   // simulate rolling of dice 36000 times
   public void rollDice()
   {
      Random randomNumbers = new Random();

      int face1; // number on first die
      int face2; // number on second die
      int totals[] = new int[ 13 ]; // frequencies of the sums

      // initialize totals to zero
      for ( int index = 0; index < totals.length; index++ )
         totals[ index ] = 0;

      // roll the dice
      for ( int roll = 1; roll <= 36000; roll++ ) {
         face1 = 1 + randomNumbers.nextInt( 6 );
         face2 = 1 + randomNumbers.nextInt( 6 );
         totals[ face1 + face2 ]++;
      } // end for

      // print the table
      System.out.printf( "%3s%12s%12s\n",
         "Sum", "Frequency", "Percentage" );

      // ignore subscripts 0 and 1
      for ( int k = 2; k < totals.length; k++ )
      {
         int percent = totals[ k ] / ( 360 );
         System.out.printf( "%3d%12d%12d\n", k, totals[ k ], percent );
      } // end for
   } // end method rollDice
} // end class Roll36
```

```
// Exercise 7.17 Solution: Roll36Test.java
// Test application for class Roll36
public class Roll36Test
{
   public static void main( String args[] )
   {
      Roll36 application = new Roll36();
      application.rollDice();
   } // end main
} // end class Roll36Test
```

```
Sum    Frequency  Percentage
  2         1007           2
  3         2012           5
  4         2959           8
  5         3946          10
  6         5020          13
  7         6055          16
  8         5014          13
  9         4022          11
 10         2993           8
 11         1997           5
 12          975           2
```

7.19 (*Airline Reservations System*) A small airline has just purchased a computer for its new automated reservations system. You have been asked to develop the new system. You are 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 first-class 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 is in the first-class or economy section of the plane.

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

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 is 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."`

ANS:

```java
// Exercise 7.19 Solution: Plane.java
// Program reserves airline seats.
import java.util.Scanner;

public class Plane
{
   // checks customers in and assigns them a boarding pass
   public void checkIn()
   {
      Scanner input = new Scanner( System.in );

      boolean seats[] = new boolean[ 10 ]; // array of seats
      int firstClass = 0; // next available first class seat
      int economy = 5; // next available economy seat

      while ( ( firstClass < 5 ) || ( economy < 10 ) )
      {
         System.out.println( "Please type 1 for First Class" );
         System.out.println( "Please type 2 for Economy" );
         System.out.print( "choice: " );
         int section = input.nextInt();

         if ( section == 1 ) // user chose first class
         {
            if ( firstClass < 5 )
            {
               firstClass++;
               System.out.printf( "First Class. Seat #%d\n", firstClass );
            } // end if
            else if ( economy < 10 ) // first class is full
            {
               System.out.println(
                  "First Class is full, Economy Class?" );
               System.out.print( "1. Yes, 2. No. Your choice: " );
               int choice = input.nextInt();

               if ( choice == 1 )
               {
                  economy++;
                  System.out.printf( "Economy Class. Seat #%d\n",
                     economy );
               }
               else
                  System.out.println( "Next flight leaves in 3 hours." );
            } // end else if
         } // end if
         else if ( section == 2 ) // user chose economy
         {
            if ( economy < 10 )
            {
               economy++;
               System.out.printf( "Economy Class. Seat #%d\n", economy );
            } // end if
            else if ( firstClass < 5 ) // economy class is full
            {
               System.out.println(
                  "Economy Class is full, First Class?" );
               System.out.print( "1. Yes, 2. No. Your choice: " );
               int choice = input.nextInt();

               if ( choice == 1 )
               {
                  firstClass++;
                  System.out.printf( "First Class. Seat #%d\n",
                     firstClass );
               } // end if
               else
                  System.out.println( "Next flight leaves in 3 hours." );
            } // end else if
         } // end else if

         System.out.println();
      } // end while

      System.out.println( "The plane is now full." );
   } // end method checkIn
} // end class Plane
```

```java
// Exercise 7.19 Solution: PlaneTest.java
// Test application for class Plane
public class PlaneTest
{
   public static void main( String args[] )
   {
      Plane application = new Plane();
      application.checkIn();
   } // end main
} // end class PlaneTest
```

```
Please type 1 for First Class
Please type 2 for Economy
choice: 1
First Class. Seat #1

Please type 1 for First Class
Please type 2 for Economy
choice: 2
Economy Class. Seat #6

.
.
.

Please type 1 for First Class
Please type 2 for Economy
choice: 1
First Class is full, Economy Class?
1. Yes, 2. No. Your choice: 1
Economy Class. Seat #7
```

7.27 (*Sieve of Eratosthenes*) A prime number is any integer greater than 1 that is 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 value `true`. For array index 2, all elements beyond element 2 in the array that have indices which are multiples of 2 (indices 4, 6, 8, 10, etc.) will be set to `false`; for array index 3, all elements beyond element 3 in the array that have indices which are multiples of 3 (indices 6, 9, 12, 15, etc.) will be set to `false`; and so on.

When this process completes, the array elements that are still `true` indicate that the index is a prime number. These indices can be displayed. Write an application that uses an array of 1000 elements to determine and display the prime numbers between 2 and 999. Ignore array elements 0 and 1.

ANS:

```java
// Exercise 7.27 Solution: Sieve.java
// Sieve of Eratosthenes
public class Sieve
{
   public static void main( String args[] )
   {
      int count = 0; // the number of primes found

      boolean primes[] = new boolean[ 1000 ]; // array of primes

      // initialize all array values to true
      for ( int index = 0; index < primes.length; index++ )
         primes[ index ] = true;

      // starting at the third value, cycle through the array and put 0
      // as the value of any greater number that is a multiple
      for ( int i = 2; i < primes.length; i++ )
         if ( primes[ i ] )
         {
            for ( int j = i + i; j < primes.length; j += i )
               primes[ j ] = false;
         } // end if

      // cycle through the array one last time to print all primes
      for ( int index = 2; index < primes.length; index++ )
         if ( primes[ index ] )
         {
            System.out.printf( "%d is prime.\n", index );
            ++count;
         } // end if

      System.out.printf( "\n%d primes found.\n", count );
   } // end main
} // end class Sieve
```

```
2 is prime.
3 is prime.
5 is prime.
7 is prime.

.
.
.

977 is prime.
983 is prime.
991 is prime.
997 is prime.

168 primes found.
```