CHAPTER

8

MULTIDIMENSIONAL ARRAYS

Objectives

- To give examples of representing data using two-dimensional arrays (§8.1).
- To declare variables for two-dimensional arrays, create arrays, and access array elements in a two-dimensional array using row and column indexes (§8.2).
- To program common operations for two-dimensional arrays (displaying arrays, summing all elements, finding the minimum and maximum elements, and random shuffling) (§8.3).
- To pass two-dimensional arrays to methods (§8.4).
- To write a program for grading multiple-choice questions using two-dimensional arrays (§8.5).
- To solve the closest-pair problem using two-dimensional arrays (§8.6).
- To check a Sudoku solution using two-dimensional arrays (§8.7).
- To use multidimensional arrays (§8.8).

8.1 Introduction

Data in a table or a matrix can be represented using a two-dimensional array.

The preceding chapter introduced how to use one-dimensional arrays to store linear collections of elements. You can use a two-dimensional array to store a matrix or a table. For example, the following table that lists the distances between cities can be stored using a two-dimensional array named `distances`.

problem

Distance Table (in miles)

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	1375	967	1087
Boston	983	0	214	1102	1763	1723	1842
New York	787	214	0	888	1549	1548	1627
Atlanta	714	1102	888	0	661	781	810
Miami	1375	1763	1549	661	0	1426	1187
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

```
double[][] distances = {
  {0, 983, 787, 714, 1375, 967, 1087},
  {983, 0, 214, 1102, 1763, 1723, 1842},
  {787, 214, 0, 888, 1549, 1548, 1627},
  {714, 1102, 888, 0, 661, 781, 810},
  {1375, 1763, 1549, 661, 0, 1426, 1187},
  {967, 1723, 1548, 781, 1426, 0, 239},
  {1087, 1842, 1627, 810, 1187, 239, 0},
};
```

8.2 Two-Dimensional Array Basics

An element in a two-dimensional array is accessed through a row and column index.

How do you declare a variable for two-dimensional arrays? How do you create a two-dimensional array? How do you access elements in a two-dimensional array? This section addresses these issues.

8.2.1 Declaring Variables of Two-Dimensional Arrays and Creating Two-Dimensional Arrays

The syntax for declaring a two-dimensional array is:

```
elementType[][] arrayRefVar;
```

or

```
elementType arrayRefVar[][]; // Allowed, but not preferred
```

As an example, here is how you would declare a two-dimensional array variable `matrix` of `int` values:

```
int[][] matrix;
```

or

```
int matrix[][]; // This style is allowed, but not preferred
```

You can create a two-dimensional array of 5-by-5 `int` values and assign it to `matrix` using this syntax:

```
matrix = new int[5][5];
```

Two subscripts are used in a two-dimensional array, one for the row and the other for the column. As in a one-dimensional array, the index for each subscript is of the `int` type and starts from `0`, as shown in Figure 8.1a.

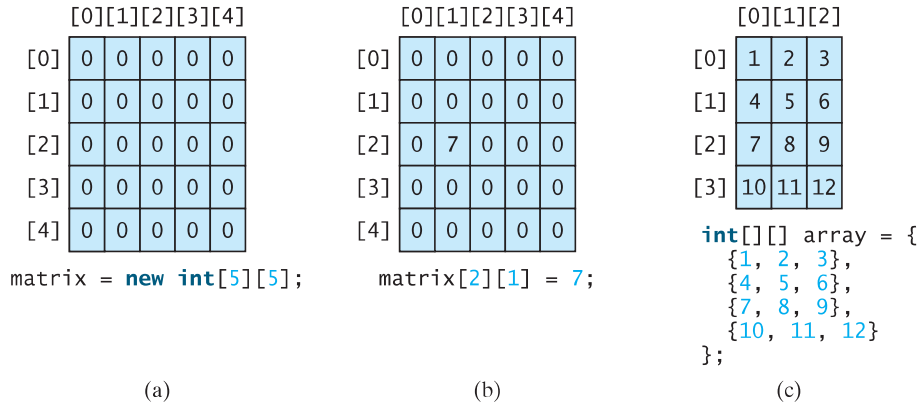


FIGURE 8.1 The index of each subscript of a two-dimensional array is an `int` value, starting from `0`.

To assign the value `7` to a specific element at row `2` and column `1`, as shown in Figure 8.1b, you can use the following syntax:

```
matrix[2][1] = 7;
```

Caution

It is a common mistake to use `matrix[2, 1]` to access the element at row `2` and column `1`. In Java, each subscript must be enclosed in a pair of square brackets.

You can also use an array initializer to declare, create, and initialize a two-dimensional array. For example, the following code in (a) creates an array with the specified initial values, as shown in Figure 8.1c. This is equivalent to the code in (b).

```
int[][] array = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9},
  {10, 11, 12}
};
```

(a)

Equivalent

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

(b)

8.2.2 Obtaining the Lengths of Two-Dimensional Arrays

A two-dimensional array is actually an array in which each element is a one-dimensional array. The length of an array `x` is the number of elements in the array, which can be obtained using `x.length`. `x[0]`, `x[1]`, . . . , and `x[x.length-1]` are arrays. Their lengths can be obtained using `x[0].length`, `x[1].length`, . . . , and `x[x.length-1].length`.

For example, suppose `x = new int[3][4]`, `x[0]`, `x[1]`, and `x[2]` are one-dimensional arrays and each contains four elements, as shown in Figure 8.2. `x.length` is `3`, and `x[0].length`, `x[1].length`, and `x[2].length` are `4`.

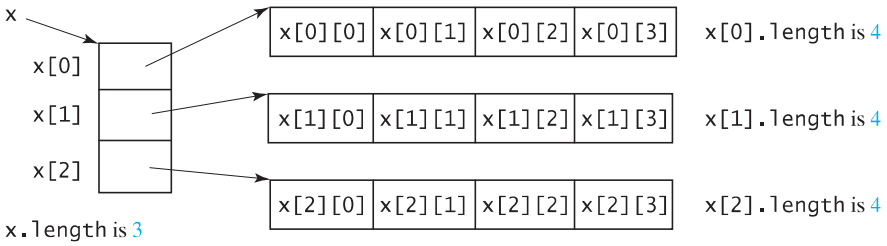
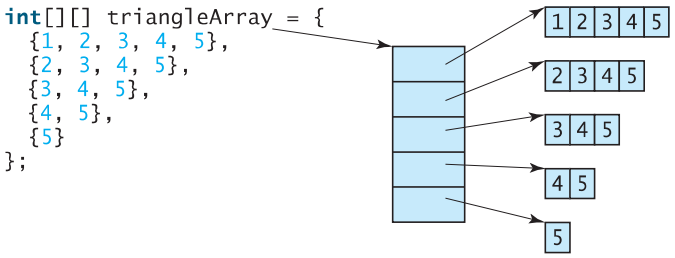


FIGURE 8.2 A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

8.2.3 Ragged Arrays

ragged array

Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths. An array of this kind is known as a *ragged array*. Here is an example of creating a ragged array:



As you can see, `triangleArray[0].length` is 5, `triangleArray[1].length` is 4, `triangleArray[2].length` is 3, `triangleArray[3].length` is 2, and `triangleArray[4].length` is 1.

If you don't know the values in a ragged array in advance, but do know the sizes—say, the same as before—you can create a ragged array using the following syntax:

```
int[][] triangleArray = new int[5][];
triangleArray[0] = new int[5];
triangleArray[1] = new int[4];
triangleArray[2] = new int[3];
triangleArray[3] = new int[2];
triangleArray[4] = new int[1];
```

You can now assign values to the array. For example,

```
triangleArray[0][3] = 50;
triangleArray[4][0] = 45;
```

Note

The syntax `new int[5][]` for creating an array requires the first index to be specified. The syntax `new int[][]` would be wrong.

8.1 Declare an array reference variable for a two-dimensional array of `int` values, create a 4-by-5 `int` matrix, and assign it to the variable.

8.2 Can the rows in a two-dimensional array have different lengths?

8.3 What is the output of the following code?

```
int[][] array = new int[5][6];
int[] x = {1, 2};
array[0] = x;
System.out.println("array[0][1] is " + array[0][1]);
```

8.4 Which of the following statements are valid?

```
int[][] r = new int[2];
int[] x = new int[];
int[][] y = new int[3][];
int[][] z = {{1, 2}};
int[][] m = {{1, 2}, {2, 3}};
int[][] n = {{1, 2}, {2, 3}, };
```

8.3 Processing Two-Dimensional Arrays

Nested `for` loops are often used to process a two-dimensional array.

Suppose an array `matrix` is created as follows:

```
int[][] matrix = new int[10][10];
```

The following are some examples of processing two-dimensional arrays.

1. *Initializing arrays with input values.* The following loop initializes the array with user input values:

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

2. *Initializing arrays with random values.* The following loop initializes the array with random values between `0` and `99`:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() * 100);
  }
}
```

3. *Printing arrays.* To print a two-dimensional array, you have to print each element in the array using a loop like the following:

```
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    System.out.print(matrix[row][column] + " ");
  }

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

4. *Summing all elements.* Use a variable named `total` to store the sum. Initially `total` is `0`. Add each element in the array to `total` using a loop like this:

```
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

5. *Summing elements by column.* For each column, use a variable named `total` to store its sum. Add each element in the column to `total` using a loop like this:

```
for (int column = 0; column < matrix[0].length; column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println("Sum for column " + column + " is "
    + total);
}
```

VideoNote
Find the row with the largest sum

6. *Which row has the largest sum?* Use variables `maxRow` and `indexOfMaxRow` to track the largest sum and index of the row. For each row, compute its sum and update `maxRow` and `indexOfMaxRow` if the new sum is greater.

```
int maxRow = 0;
int indexOfMaxRow = 0;

// Get sum of the first row in maxRow
for (int column = 0; column < matrix[0].length; column++) {
  maxRow += matrix[0][column];
}

for (int row = 1; row < matrix.length; row++) {
  int totalOfThisRow = 0;
  for (int column = 0; column < matrix[row].length; column++)
    totalOfThisRow += matrix[row][column];

  if (totalOfThisRow > maxRow) {
    maxRow = totalOfThisRow;
    indexOfMaxRow = row;
  }
}

System.out.println("Row " + indexOfMaxRow
  + " has the maximum sum of " + maxRow);
```

7. *Random shuffling.* Shuffling the elements in a one-dimensional array was introduced in Section 7.2.6. How do you shuffle all the elements in a two-dimensional array? To accomplish this, for each element `matrix[i][j]`, randomly generate indices `i1` and `j1` and swap `matrix[i][j]` with `matrix[i1][j1]`, as follows:

```
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    int i1 = (int)(Math.random() * matrix.length);
    int j1 = (int)(Math.random() * matrix[i].length);

    // Swap matrix[i][j] with matrix[i1][j1]
```

```
      int temp = matrix[i][j];
      matrix[i][j] = matrix[i1][j1];
      matrix[i1][j1] = temp;
    }
  }
```

8.5 Show the output of the following code:


```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
for (int i = array.length - 1; i >= 0; i--) {
  for (int j = array[i].length - 1; j >= 0; j--)
    System.out.print(array[i][j] + " ");
  System.out.println();
}
```

8.6 Show the output of the following code:

```
int[][] array = {{1, 2}, {3, 4}, {5, 6}};
int sum = 0;
for (int i = 0; i < array.length; i++)
  sum += array[i][0];
System.out.println(sum);
```

8.4 Passing Two-Dimensional Arrays to Methods

When passing a two-dimensional array to a method, the reference of the array is passed to the method.

You can pass a two-dimensional array to a method just as you pass a one-dimensional array. You can also return an array from a method. Listing 8.1 gives an example with two methods. The first method, `getArray()`, returns a two-dimensional array, and the second method, `sum(int[][] m)`, returns the sum of all the elements in a matrix.

LISTING 8.1 PassTwoDimensionalArray.java

```
1  import java.util.Scanner;
2
3  public class PassTwoDimensionalArray {
4    public static void main(String[] args) {
5      int[][] m = getArray(); // Get an array                    get array
6
7      // Display sum of elements
8      System.out.println("\nSum of all elements is " + sum(m));  pass array
9    }
10
11   public static int[][] getArray() {                           getArray method
12     // Create a Scanner
13     Scanner input = new Scanner(System.in);
14
15     // Enter array values
16     int[][] m = new int[3][4];
17     System.out.println("Enter " + m.length + " rows and "
18       + m[0].length + " columns: ");
19     for (int i = 0; i < m.length; i++)
20       for (int j = 0; j < m[i].length; j++)
21         m[i][j] = input.nextInt();
22
```

return array

```
23        return m;
24    }
25
26    public static int sum(int[][] m) {
27      int total = 0;
28      for (int row = 0; row < m.length; row++) {
29        for (int column = 0; column < m[row].length; column++) {
30          total += m[row][column];
31        }
32      }
33
34      return total;
35    }
36  }
```

sum method

```
Enter 3 rows and 4 columns:
1 2 3 4 Enter
5 6 7 8 Enter
9 10 11 12 Enter

Sum of all elements is 78
```

The method **getArray** prompts the user to enter values for the array (lines 11–24) and returns the array (line 23).

The method **sum** (lines 26–35) has a two-dimensional array argument. You can obtain the number of rows using **m.length** (line 28) and the number of columns in a specified row using **m[row].length** (line 29).

8.7 Show the output of the following code:

```
public class Test {
  public static void main(String[] args) {
    int[][] array = {{1, 2, 3, 4}, {5, 6, 7, 8}};
    System.out.println(m1(array)[0]);
    System.out.println(m1(array)[1]);
  }

  public static int[] m1(int[][] m) {
    int[] result = new int[2];
    result[0] = m.length;
    result[1] = m[0].length;
    return result;
  }
}
```

8.5 Case Study: Grading a Multiple-Choice Test

The problem is to write a program that will grade multiple-choice tests.

VideoNote
Grade multiple-choice test

Suppose you need to write a program that grades multiple-choice tests. Assume there are eight students and ten questions, and the answers are stored in a two-dimensional array. Each row records a student's answers to the questions, as shown in the following array.

Students' Answers to the Questions:

	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

The key is stored in a one-dimensional array:

Key to the Questions:

	0	1	2	3	4	5	6	7	8	9
Key	D	B	D	C	C	D	A	E	A	D

Your program grades the test and displays the result. It compares each student's answers with the key, counts the number of correct answers, and displays it. Listing 8.2 gives the program.

LISTING 8.2 GradeExam.java

```
1  public class GradeExam {
2    /** Main method */
3    public static void main(String[] args) {
4      // Students' answers to the questions
5      char[][] answers = {                                          2-D array
6        {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
7        {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
8        {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
9        {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
10       {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
11       {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
12       {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
13       {'E', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'}};
14
15     // Key to the questions
16     char[] keys = {'D', 'B', 'D', 'C', 'C', 'D', 'A', 'E', 'A', 'D'};   1-D array
17
18     // Grade all answers
19     for (int i = 0; i < answers.length; i++) {
20       // Grade one student
21       int correctCount = 0;
22       for (int j = 0; j < answers[i].length; j++) {
23         if (answers[i][j] == keys[j])                              compare with key
24           correctCount++;
25       }
26
27       System.out.println("Student " + i + "'s correct count is " +
28         correctCount);
29     }
30   }
31 }
```

```
Student 0's correct count is 7
Student 1's correct count is 6
Student 2's correct count is 5
Student 3's correct count is 4
Student 4's correct count is 8
Student 5's correct count is 7
Student 6's correct count is 7
Student 7's correct count is 7
```

The statement in lines 5–13 declares, creates, and initializes a two-dimensional array of characters and assigns the reference to `answers` of the `char[][]` type.

The statement in line 16 declares, creates, and initializes an array of `char` values and assigns the reference to `keys` of the `char[]` type.

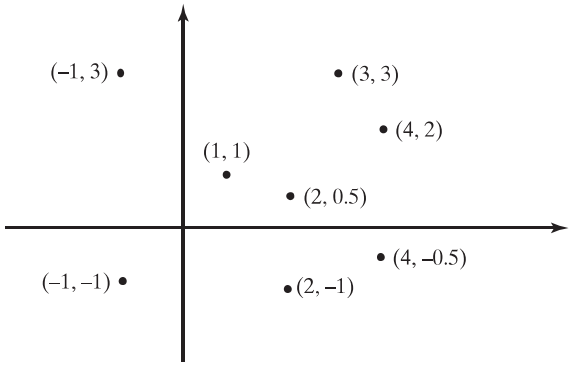
Each row in the array `answers` stores a student's answer, which is graded by comparing it with the key in the array `keys`. The result is displayed immediately after a student's answer is graded.

8.6 Case Study: Finding the Closest Pair

This section presents a geometric problem for finding the closest pair of points.

closest-pair animation on the Companion Website

Given a set of points, the closest-pair problem is to find the two points that are nearest to each other. In Figure 8.3, for example, points `(1, 1)` and `(2, 0.5)` are closest to each other. There are several ways to solve this problem. An intuitive approach is to compute the distances between all pairs of points and find the one with the minimum distance, as implemented in Listing 8.3.



	x	y
0	–1	3
1	–1	–1
2	1	1
3	2	0.5
4	2	–1
5	3	3
6	4	2
7	4	–0.5

FIGURE 8.3 Points can be represented in a two-dimensional array.

LISTING 8.3 FindNearestPoints.java

```
1  import java.util.Scanner;
2
3  public class FindNearestPoints {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6      System.out.print("Enter the number of points: ");
7      int numberOfPoints = input.nextInt();
8
9      // Create an array to store points
```

number of points

```
10      double[][] points = new double[numberOfPoints][2];                  2-D array
11      System.out.print("Enter " + numberOfPoints + " points: ");
12      for (int i = 0; i < points.length; i++) {                           read points
13        points[i][0] = input.nextDouble();
14        points[i][1] = input.nextDouble();
15      }
16
17      // p1 and p2 are the indices in the points' array
18      int p1 = 0, p2 = 1; // Initial two points                           track two points
19      double shortestDistance = distance(points[p1][0], points[p1][1],    track shortestDistance
20        points[p2][0], points[p2][1]); // Initialize shortestDistance
21
22      // Compute distance for every two points
23      for (int i = 0; i < points.length; i++) {                           for each point i
24        for (int j = i + 1; j < points.length; j++) {                     for each point j
25          double distance = distance(points[i][0], points[i][1],          distance between i and j
26            points[j][0], points[j][1]); // Find distance                 distance between two points
27
28          if (shortestDistance > distance) {
29            p1 = i; // Update p1
30            p2 = j; // Update p2
31            shortestDistance = distance; // Update shortestDistance      update shortestDistance
32          }
33        }
34      }
35
36      // Display result
37      System.out.println("The closest two points are " +
38        "(" + points[p1][0] + ", " + points[p1][1] + ") and (" +
39        points[p2][0] + ", " + points[p2][1] + ")");
40    }
41
42    /** Compute the distance between two points (x1, y1) and (x2, y2)*/
43    public static double distance(
44        double x1, double y1, double x2, double y2) {
45      return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
46    }
47  }
```

```
Enter the number of points: 8 Enter
Enter 8 points: -1 3  -1 -1  1 1  2 0.5  2 -1  3 3  4 2 4 -0.5 Enter
The closest two points are (1, 1) and (2, 0.5)
```

The program prompts the user to enter the number of points (lines 6–7). The points are read from the console and stored in a two-dimensional array named **points** (lines 12–15). The program uses the variable **shortestDistance** (line 19) to store the distance between the two nearest points, and the indices of these two points in the **points** array are stored in **p1** and **p2** (line 18).

For each point at index **i**, the program computes the distance between **points[i]** and **points[j]** for all **j > i** (lines 23–34). Whenever a shorter distance is found, the variable **shortestDistance** and **p1** and **p2** are updated (lines 28–32).

The distance between two points **(x1, y1)** and **(x2, y2)** can be computed using the formula $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ (lines 43–46).

The program assumes that the plane has at least two points. You can easily modify the program to handle the case if the plane has zero or one point.

multiple closest pairs

Note that there might be more than one closest pair of points with the same minimum distance. The program finds one such pair. You may modify the program to find all closest pairs in Programming Exercise 8.8.

input file

Tip

It is cumbersome to enter all points from the keyboard. You may store the input in a file, say **FindNearestPoints.txt**, and compile and run the program using the following command:

```
java FindNearestPoints < FindNearestPoints.txt
```

8.7 Case Study: Sudoku

The problem is to check whether a given Sudoku solution is correct.

VideoNote
Sudoku

This section presents an interesting problem of a sort that appears in the newspaper every day. It is a number-placement puzzle, commonly known as *Sudoku*. This is a very challenging problem. To make it accessible to the novice, this section presents a simplified version of the Sudoku problem, which is to verify whether a Sudoku solution is correct. The complete program for finding a Sudoku solution is presented in Supplement VI.A.

fixed cells
free cells

Sudoku is a 9 × 9 grid divided into smaller 3 × 3 boxes (also called *regions* or *blocks*), as shown in Figure 8.4a. Some cells, called *fixed cells*, are populated with numbers from **1** to **9**. The objective is to fill the empty cells, also called *free cells*, with the numbers **1** to **9** so that every row, every column, and every 3 × 3 box contains the numbers **1** to **9**, as shown in Figure 8.4b.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6							
			4	1	9			5
				8			7	9

(a) Puzzle

Solution

5	3	*4*	*6*	7	*8*	*9*	*1*	*2*
6	*7*	*2*	1	9	5	*3*	*4*	*8*
1	9	8	*3*	*4*	*2*	*5*	6	*7*
8	*5*	*9*	*7*	6	*1*	*4*	*2*	3
4	*2*	*6*	8	*5*	3	*7*	*9*	1
7	*1*	*3*	*9*	2	*4*	*8*	*5*	6
9	6	*1*	*5*	*3*	*7*	*2*	*8*	*4*
2	*8*	*7*	4	1	9	6	*3*	5
3	*4*	*5*	*2*	8	6	*1*	7	9

(b) Solution

FIGURE 8.4 The Sudoku puzzle in (a) is solved in (b).

representing a grid

For convenience, we use value **0** to indicate a free cell, as shown in Figure 8.5a. The grid can be naturally represented using a two-dimensional array, as shown in Figure 8.5b.

5	3	0	0	7	0	0	0	0
6	0	0	1	9	5	0	0	0
0	9	8	0	0	0	0	6	0
8	0	0	0	6	0	0	0	3
4	0	0	8	0	3	0	0	1
7	0	0	0	2	0	0	0	6
0	6	0	0	0	0	0	0	0
0	0	0	4	1	9	0	0	5
0	0	0	0	8	0	0	7	9

(a)

```
int[][] grid =
  {{5, 3, 0, 0, 7, 0, 0, 0, 0},
   {6, 0, 0, 1, 9, 5, 0, 0, 0},
   {0, 9, 8, 0, 0, 0, 0, 6, 0},
   {8, 0, 0, 0, 6, 0, 0, 0, 3},
   {4, 0, 0, 8, 0, 3, 0, 0, 1},
   {7, 0, 0, 0, 2, 0, 0, 0, 6},
   {0, 6, 0, 0, 0, 0, 2, 8, 0},
   {0, 0, 0, 4, 1, 9, 0, 0, 5},
   {0, 0, 0, 0, 8, 0, 0, 7, 9}
  };
```

(b)

FIGURE 8.5 A grid can be represented using a two-dimensional array.

To find a solution for the puzzle, we must replace each 0 in the grid with an appropriate number from 1 to 9. For the solution to the puzzle in Figure 8.5, the grid should be as shown in Figure 8.6.

Once a solution to a Sudoku puzzle is found, how do you verify that it is correct? Here are two approaches:

- Check if every row has numbers from 1 to 9, every column has numbers from 1 to 9, and every small box has numbers from 1 to 9.
- Check each cell. Each cell must be a number from 1 to 9 and the cell must be unique on every row, every column, and every small box.

```
A solution grid is
  {{5, 3, 4, 6, 7, 8, 9, 1, 2},
   {6, 7, 2, 1, 9, 5, 3, 4, 8},
   {1, 9, 8, 3, 4, 2, 5, 6, 7},
   {8, 5, 9, 7, 6, 1, 4, 2, 3},
   {4, 2, 6, 8, 5, 3, 7, 9, 1},
   {7, 1, 3, 9, 2, 4, 8, 5, 6},
   {9, 6, 1, 5, 3, 7, 2, 8, 4},
   {2, 8, 7, 4, 1, 9, 6, 3, 5},
   {3, 4, 5, 2, 8, 6, 1, 7, 9}
  };
```

FIGURE 8.6 A solution is stored in `grid`.

The program in Listing 8.4 prompts the user to enter a solution and reports whether it is valid. We use the second approach in the program to check whether the solution is correct.

LISTING 8.4 CheckSudokuSolution.java

```
1  import java.util.Scanner;
2
3  public class CheckSudokuSolution {
4    public static void main(String[] args) {
5      // Read a Sudoku solution
6      int[][] grid = readASolution();                          read input
7
8      System.out.println(isValid(grid) ? "Valid solution" :    solution valid?
9        "Invalid solution");
10   }
11
12   /** Read a Sudoku solution from the console */
13   public static int[][] readASolution() {                    read solution
14     // Create a Scanner
15     Scanner input = new Scanner(System.in);
16
17     System.out.println("Enter a Sudoku puzzle solution:");
18     int[][] grid = new int[9][9];
19     for (int i = 0; i < 9; i++)
20       for (int j = 0; j < 9; j++)
21         grid[i][j] = input.nextInt();
22
23     return grid;
24   }
25
26   /** Check whether a solution is valid */
27   public static boolean isValid(int[][] grid) {              check solution
```

```
28      for (int i = 0; i < 9; i++)
29        for (int j = 0; j < 9; j++)
30          if (grid[i][j] < 1 || grid[i][j] > 9
31              || !isValid(i, j, grid))
32            return false;
33        return true; // The solution is valid
34    }
35
36    /** Check whether grid[i][j] is valid in the grid */
37    public static boolean isValid(int i, int j, int[][] grid) {
38      // Check whether grid[i][j] is unique in i's row
39      for (int column = 0; column < 9; column++)
40        if (column != j && grid[i][column] == grid[i][j])
41          return false;
42
43      // Check whether grid[i][j] is unique in j's column
44      for (int row = 0; row < 9; row++)
45        if (row != i && grid[row][j] == grid[i][j])
46          return false;
47
48      // Check whether grid[i][j] is unique in the 3-by-3 box
49      for (int row = (i / 3) * 3; row < (i / 3) * 3 + 3; row++)
50        for (int col = (j / 3) * 3; col < (j / 3) * 3 + 3; col++)
51          if (row != i && col != j && grid[row][col] == grid[i][j])
52            return false;
53
54      return true; // The current value at grid[i][j] is valid
55    }
56  }
```

check rows

check columns

check small boxes

```
Enter a Sudoku puzzle solution:
9 6 3 1 7 4 2 5 8 Enter
1 7 8 3 2 5 6 4 9 Enter
2 5 4 6 8 9 7 3 1 Enter
8 2 1 4 3 7 5 9 6 Enter
4 9 6 8 5 2 3 1 7 Enter
7 3 5 9 6 1 8 2 4 Enter
5 8 9 7 1 3 4 6 2 Enter
3 1 7 2 4 6 9 8 5 Enter
6 4 2 5 9 8 1 7 3 Enter
Valid solution
```

The program invokes the **`readASolution()`** method (line 6) to read a Sudoku solution and return a two-dimensional array representing a Sudoku grid.

`isValid` method

The **`isValid(grid)`** method checks whether the values in the grid are valid by verifying that each value is between **`1`** and **`9`** and that each value is valid in the grid (lines 27–34).

overloaded `isValid` method

The **`isValid(i, j, grid)`** method checks whether the value at **`grid[i][j]`** is valid. It checks whether **`grid[i][j]`** appears more than once in row **`i`** (lines 39–41), in column **`j`** (lines 44–46), and in the 3 × 3 box (lines 49–52).

How do you locate all the cells in the same box? For any **`grid[i][j]`**, the starting cell of the 3 × 3 box that contains it is **`grid[(i / 3) * 3][(j / 3) * 3]`**, as illustrated in Figure 8.7.

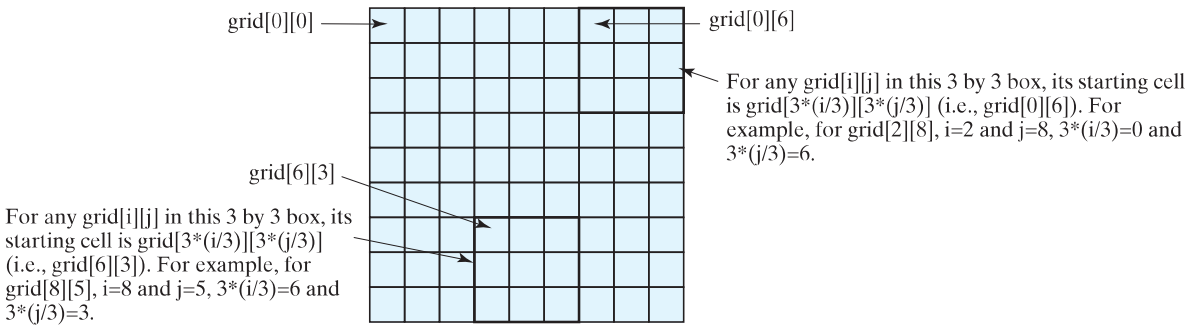


FIGURE 8.7 The location of the first cell in a 3 × 3 box determines the locations of other cells in the box.

With this observation, you can easily identify all the cells in the box. For instance, if `grid[r][c]` is the starting cell of a 3 × 3 box, the cells in the box can be traversed in a nested loop as follows:

```
// Get all cells in a 3-by-3 box starting at grid[r][c]
for (int row = r; row < r + 3; row++)
  for (int col = c; col < c + 3; col++)
    // grid[row][col] is in the box
```

It is cumbersome to enter 81 numbers from the console. When you test the program, you may store the input in a file, say **CheckSudokuSolution.txt** (see www.cs.armstrong.edu/liang/data/CheckSudokuSolution.txt), and run the program using the following command: input file

```
java CheckSudokuSolution < CheckSudokuSolution.txt
```

8.8 Multidimensional Arrays

A two-dimensional array consists of an array of one-dimensional arrays and a three-dimensional array consists of an array of two-dimensional arrays.

In the preceding section, you used a two-dimensional array to represent a matrix or a table. Occasionally, you will need to represent *n*-dimensional data structures. In Java, you can create *n*-dimensional arrays for any integer *n*.

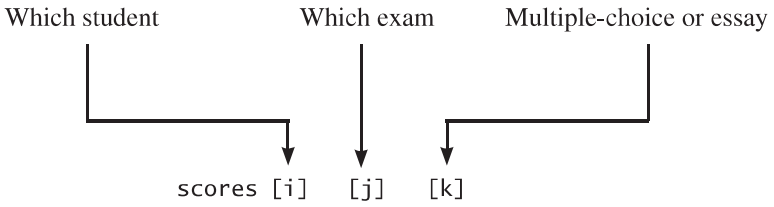
The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare *n*-dimensional array variables and create *n*-dimensional arrays for *n* >= 3. For example, you may use a three-dimensional array to store exam scores for a class of six students with five exams, and each exam has two parts (multiple-choice and essay). The following syntax declares a three-dimensional array variable `scores`, creates an array, and assigns its reference to `scores`.

```
double[][][] scores = new double[6][5][2];
```

You can also use the short-hand notation to create and initialize the array as follows:

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```

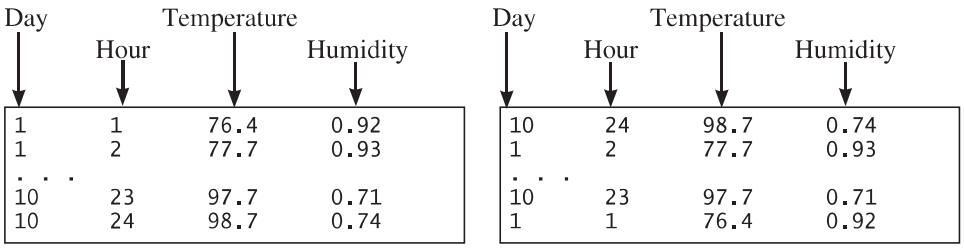
`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is `9.0`. `scores[0][1][1]` refers to the essay score for the first student's second exam, which is `22.5`. This is depicted in the following figure:



A multidimensional array is actually an array in which each element is another array. A three-dimensional array consists of an array of two-dimensional arrays. A two-dimensional array consists of an array of one-dimensional arrays. For example, suppose `x = new int[2][2][5]`, and `x[0]` and `x[1]` are two-dimensional arrays. `X[0][0]`, `x[0][1]`, `x[1][0]`, and `x[1][1]` are one-dimensional arrays and each contains five elements. `x.length` is `2`, `x[0].length` and `x[1].length` are `2`, and `X[0][0].length`, `x[0][1].length`, `x[1][0].length`, and `x[1][1].length` are `5`.

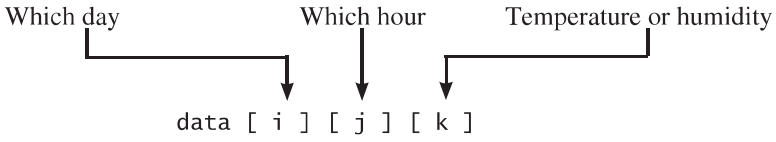
8.8.1 Case Study: Daily Temperature and Humidity

Suppose a meteorology station records the temperature and humidity every hour of every day and stores the data for the past ten days in a text file named **Weather.txt** (see www.cs.armstrong.edu/liang/data/Weather.txt). Each line of the file consists of four numbers that indicate the day, hour, temperature, and humidity. The contents of the file may look like the one in (a).



Note that the lines in the file are not necessarily in increasing order of day and hour. For example, the file may appear as shown in (b).

Your task is to write a program that calculates the average daily temperature and humidity for the `10` days. You can use the input redirection to read the file and store the data in a three-dimensional array named `data`. The first index of `data` ranges from `0` to `9` and represents `10` days, the second index ranges from `0` to `23` and represents `24` hours, and the third index ranges from `0` to `1` and represents temperature and humidity, as depicted in the following figure:



Note that the days are numbered from 1 to 10 and the hours from 1 to 24 in the file. Because the array index starts from 0, `data[0][0][0]` stores the temperature in day 1 at hour 1 and `data[9][23][1]` stores the humidity in day 10 at hour 24.

The program is given in Listing 8.5.

LISTING 8.5 Weather.java

```
1  import java.util.Scanner;
2
3  public class Weather {
4    public static void main(String[] args) {
5      final int NUMBER_OF_DAYS = 10;
6      final int NUMBER_OF_HOURS = 24;
7      double[][][] data
8        = new double[NUMBER_OF_DAYS][NUMBER_OF_HOURS][2];
9
10     Scanner input = new Scanner(System.in);
11     // Read input using input redirection from a file
12     for (int k = 0; k < NUMBER_OF_DAYS * NUMBER_OF_HOURS; k++) {
13       int day = input.nextInt();
14       int hour = input.nextInt();
15       double temperature = input.nextDouble();
16       double humidity = input.nextDouble();
17       data[day - 1][hour - 1][0] = temperature;
18       data[day - 1][hour - 1][1] = humidity;
19     }
20
21     // Find the average daily temperature and humidity
22     for (int i = 0; i < NUMBER_OF_DAYS; i++) {
23       double dailyTemperatureTotal = 0, dailyHumidityTotal = 0;
24       for (int j = 0; j < NUMBER_OF_HOURS; j++) {
25         dailyTemperatureTotal += data[i][j][0];
26         dailyHumidityTotal += data[i][j][1];
27       }
28
29       // Display result
30       System.out.println("Day " + i + "'s average temperature is "
31         + dailyTemperatureTotal / NUMBER_OF_HOURS);
32       System.out.println("Day " + i + "'s average humidity is "
33         + dailyHumidityTotal / NUMBER_OF_HOURS);
34     }
35   }
36 }
```

three-dimensional array

```
Day 0's average temperature is 77.7708
Day 0's average humidity is 0.929583
Day 1's average temperature is 77.3125
Day 1's average humidity is 0.929583
. . .
Day 9's average temperature is 79.3542
Day 9's average humidity is 0.9125
```

You can use the following command to run the program:

```
java Weather < Weather.txt
```

A three-dimensional array for storing temperature and humidity is created in line 8. The loop in lines 12–19 reads the input to the array. You can enter the input from the keyboard, but

doing so will be awkward. For convenience, we store the data in a file and use input redirection to read the data from the file. The loop in lines 24–27 adds all temperatures for each hour in a day to `dailyTemperatureTotal` and all humidity for each hour to `dailyHumidityTotal`. The average daily temperature and humidity are displayed in lines 30–33.

8.8.2 Case Study: Guessing Birthdays

Listing 3.3, GuessBirthday.java, gives a program that guesses a birthday. The program can be simplified by storing the numbers in five sets in a three-dimensional array, and it prompts the user for the answers using a loop, as shown in Listing 8.6. The sample run of the program can be the same as shown in Listing 4.3.

LISTING 8.6 GuessBirthdayUsingArray.java

three-dimensional array

Set i

add to day

```
1  import java.util.Scanner;
2
3  public class GuessBirthdayUsingArray {
4    public static void main(String[] args) {
5      int day = 0; // Day to be determined
6      int answer;
7
8      int[][][] dates = {
9        {{ 1,  3,  5,  7},
10        { 9, 11, 13, 15},
11        {17, 19, 21, 23},
12        {25, 27, 29, 31}},
13       {{ 2,  3,  6,  7},
14        {10, 11, 14, 15},
15        {18, 19, 22, 23},
16        {26, 27, 30, 31}},
17       {{ 4,  5,  6,  7},
18        {12, 13, 14, 15},
19        {20, 21, 22, 23},
20        {28, 29, 30, 31}},
21       {{ 8,  9, 10, 11},
22        {12, 13, 14, 15},
23        {24, 25, 26, 27},
24        {28, 29, 30, 31}},
25       {{16, 17, 18, 19},
26        {20, 21, 22, 23},
27        {24, 25, 26, 27},
28        {28, 29, 30, 31}}};
29
30     // Create a Scanner
31     Scanner input = new Scanner(System.in);
32
33     for (int i = 0; i < 5; i++) {
34       System.out.println("Is your birthday in Set" + (i + 1) + "?");
35       for (int j = 0; j < 4; j++) {
36         for (int k = 0; k < 4; k++)
37           System.out.printf("%4d", dates[i][j][k]);
38         System.out.println();
39       }
40
41       System.out.print("\nEnter 0 for No and 1 for Yes: ");
42       answer = input.nextInt();
43
44       if (answer == 1)
45         day += dates[i][0][0];
```