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 indices (§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.

A *two-dimensional array* is an array that contains other arrays as its elements. 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},
};
```

Each element in the `distances` array is another array, so `distances` is considered a *nested array*. In this example, a two-dimensional array is used to store two-dimensional data.

8.2 Two-Dimensional Array Basics

An element in a two-dimensional array is accessed through a row and a 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 will address 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 as follows:

```
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. The two subscripts are conveniently called *row index* and *column index*. 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.

row index
column index

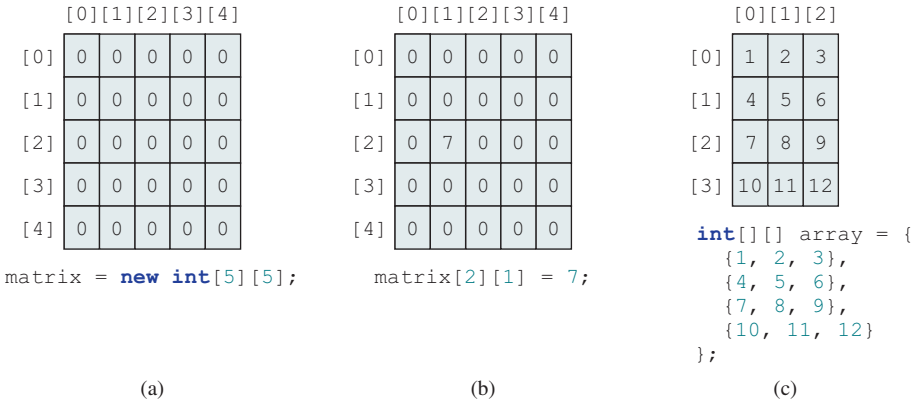


FIGURE 8.1 The index of each subscript of a two-dimensional array is an `int` value, starting from `0`.

To assign the value `7` to a specific element at row index `2` and column index `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 that `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`.

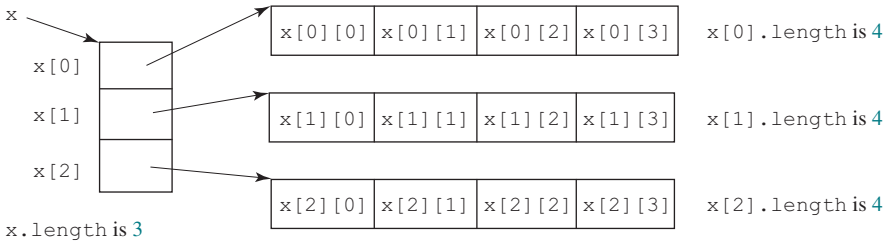
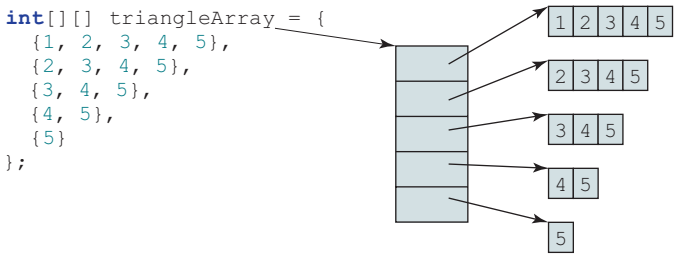


FIGURE 8.2 A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

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 in the preceding figure—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] = 4;
triangleArray[4][0] = 5;
```

Note

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

8.2.1 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.

8.2.2 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}, };
```

8.2.3 Write an expression to obtain the row size of a two-dimensional array `x` and an expression to obtain the size of the first row.

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

8.2.5 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]);
```

8.3 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 java.util.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 loop:

```
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;
      }
    }
```

8.3.1 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();
}
```

8.3.2 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

sum method

```
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  }
```

```
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).

8.4.1 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.

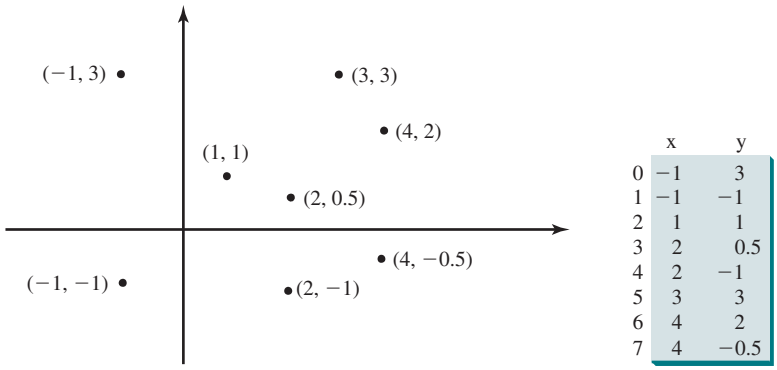
8.5.1 How do you modify the code so it also displays the highest count and the student with the highest count?

8.6 Case Study: Finding the Closest Pair

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

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.

closest-pair animation on the Companion Website



	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 and 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 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 run the program using the following command:

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

Check Point

8.6.1 What happens if the input has only one point?

8.7 Case Study: Sudoku

Key Point

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*.

Writing a program to solve a Sudoku problem is very challenging. 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.C.

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 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:

1. 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.
2. 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.

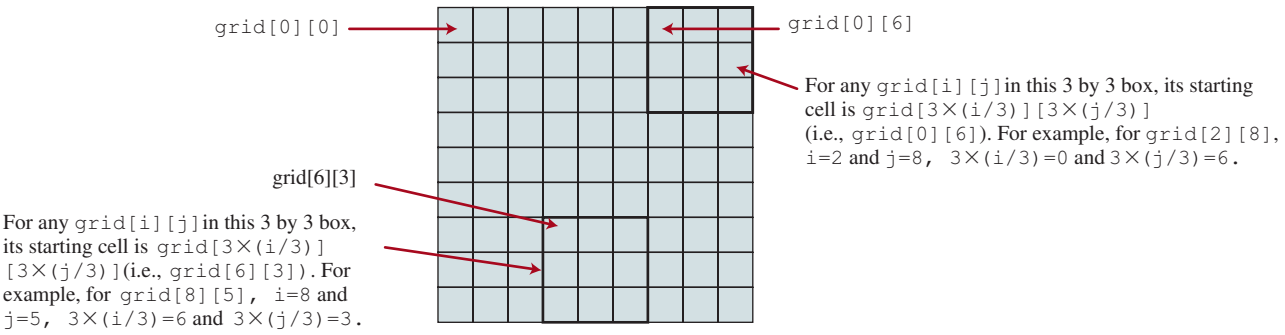


FIGURE 8.7 The location of the first cell in a 3 × 3 box determines the locations of other cells in the box.

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 liveexample.pearsoncmg.com/data/CheckSudokuSolution.txt) and run the program using the following command: input file

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

8.7.1 What happens if the code in line 51 in Listing 8.4 is changed to

```
if (row != i && col != j && grid[row][col] == grid[i][j])
```

Check Point

8.8 Multidimensional Arrays

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

Key Point

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 positive integer *n*.

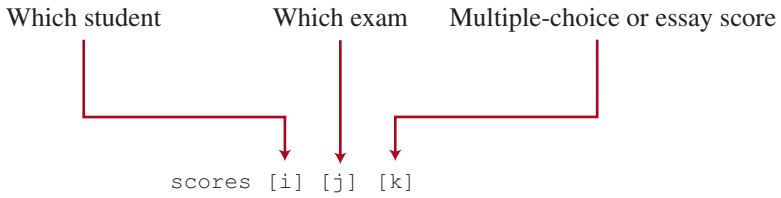
The way to declare two-dimensional array variables and create two-dimensional arrays can be generalized to declare *n*-dimensional array variables and create *n*-dimensional arrays for $n >= 3$. For example, you may use a three-dimensional array to store exam scores for a class of six students with five exams, and each exam has two parts (multiple-choice and essay type questions). 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 array initializer 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}}};
```

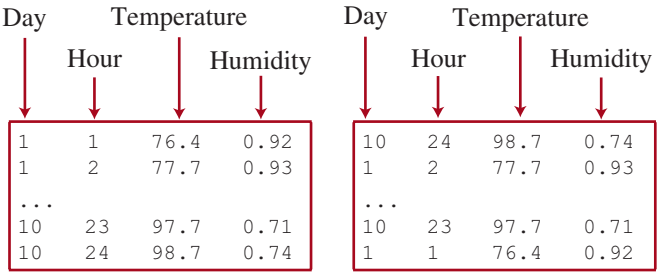
`scores[0][1][0]` refers to the multiple-choice score for the first student's second exam, which is `9.0`. `scores[0][1][1]` refers to the essay score for the first student's second exam, which is `22.5`. This is depicted in the following figure:



A multidimensional array is actually an array in which each element is another array. A three-dimensional array is an array of two-dimensional arrays. A two-dimensional array is an array of one-dimensional arrays. For example, suppose that `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`.

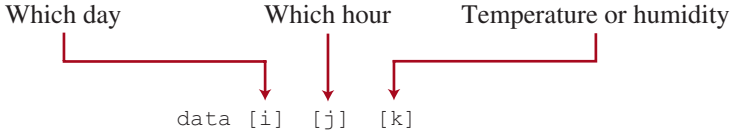
8.8.1 Case Study: Daily Temperature and Humidity

Suppose a meteorology station records the temperature and humidity every hour of every day, and stores the data for the past 10 days in a text file named **Weather.txt** (see liveexample.pearsoncmg.com/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 those in (a).



Note 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 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 4.3, 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 given in Listing 8.6. The sample run of the program can be the same as given 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];
```

```
46      }
47
48      System.out.println("Your birthday is " + day);
49    }
50  }
```

A three-dimensional array `dates` is created in lines 8–28. This array stores five sets of numbers. Each set is a 4-by-4 two-dimensional array.

The loop starting from line 33 displays the numbers in each set and prompts the user to answer whether the birthday is in the set (lines 41 and 42). If the day is in the set, the first number (`dates[i][0][0]`) in the set is added to variable `day` (line 45).

8.8.1 Declare an array variable for a three-dimensional array, create a $4 \times 6 \times 5$ `int` array, and assign its reference to the variable.

8.8.2 Assume `char[][][] x = new char[12][5][2]`, how many elements are in the array? What are `x.length`, `x[2].length`, and `x[0][0].length`?

8.8.3 Show the output of the following code:

```
int[][][] array = {{{1, 2}, {3, 4}}, {{5, 6},{7, 8}}};
System.out.println(array[0][0][0]);
System.out.println(array[1][1][1]);
```

Key Terms

column index 313
multidimensional array 325
nested array 312
row index 313
two-dimensional array 312

Chapter Summary

1. A two-dimensional array can be used to store a table.

2. A variable for two-dimensional arrays can be declared using the syntax: `elementType[][] arrayVar`.

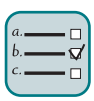
3. A two-dimensional array can be created using the syntax: `new elementType[ROW_SIZE][COLUMN_SIZE]`.

4. Each element in a two-dimensional array is represented using the syntax: `arrayVar[rowIndex][columnIndex]`.

5. You can create and initialize a two-dimensional array using an array initializer with the syntax: `elementType[][] arrayVar = {{row values}, . . ., {row values}}`.

6. You can use arrays of arrays to form multidimensional arrays. For example, a variable for three-dimensional arrays can be declared as `elementType[][][] arrayVar` and a three-dimensional array can be created using `new elementType[size1][size2][size3]`.

Quiz



Answer the quiz for this chapter online at the book Companion Website.

MyProgrammingLab™

PROGRAMMING EXERCISES

***8.1** (*Sum elements row by row*) Write a method that returns the sum of all the elements in a specified row in a matrix using the following header:

```
public static double sumRow(double[][] m, int rowIndex)
```

Write a test program that reads a 3-by-4 matrix and displays the sum of each row. Here is a sample run:

```
Enter a 3-by-4 matrix row by row:
1.5 2 3 4 Enter
5.5 6 7 8 Enter
9.5 1 3 1 Enter
Sum of the elements at row 0 is 10.5
Sum of the elements at row 1 is 26.5
Sum of the elements at row 2 is 14.5
```

***8.2** (*Average the major diagonal in a matrix*) Write a method that averages all the numbers in the major diagonal in an $n \times n$ matrix of `double` values using the following header:

```
public static double averageMajorDiagonal(double[][] m)
```

Write a test program that reads a 4-by-4 matrix and displays the average of all its elements on the major diagonal. Here is a sample run:

```
Enter a 4-by-4 matrix row by row:
1 2 3 4.0 Enter
5 6.5 7 8 Enter
9 10 11 12 Enter
13 14 15 16 Enter
Average of the elements in the major diagonal is 8.625
```

***8.3** (*Sort students on grades*) Rewrite Listing 8.2, GradeExam.java, to display students in decreasing order of the number of correct answers.

****8.4** (*Compute the weekly hours for each employee*) Suppose the weekly hours for all employees are stored in a two-dimensional array. Each row records an employee's seven-day work hours with seven columns. For example, the following array stores the work hours for eight employees. Write a program that displays employees and their total hours in increasing order of the total hours.

	Su	M	T	W	Th	F	Sa
Employee 0	2	4	3	4	5	8	8
Employee 1	7	3	4	3	3	4	4
Employee 2	3	3	4	3	3	2	2
Employee 3	9	3	4	7	3	4	1
Employee 4	3	5	4	3	6	3	8
Employee 5	3	4	4	6	3	4	4
Employee 6	3	7	4	8	3	8	4
Employee 7	6	3	5	9	2	7	9

8.5 (*Algebra: add two matrices*) Write a method to add two matrices. The header of the method is as follows:

```
public static double[][] addMatrix(double[][] a, double[][] b)
```

In order to be added, the two matrices must have the same dimensions and the same or compatible types of elements. Let **c** be the resulting matrix. Each element c_{ij} is $a_{ij} + b_{ij}$. For example, for two 2×2 matrices **a** and **b**, **c** is

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{pmatrix}$$

Write a test program that prompts the user to enter two 2×2 matrices and displays their sum. Here is a sample run:

```
Enter matrix1: 1 2 3 4 Enter
Enter matrix2: 0 2 4 1 Enter
The matrices are added as follows
 1.0 2.0      0.0 2.0      1.0 4.0
 3.0 4.0  +   4.0 1.0  =   7.0 5.0
```

****8.6** (*Algebra: multiply two matrices*) Write a method to multiply two matrices. The header of the method is:

VideoNote
Multiply two matrices

```
public static double[][]
    multiplyMatrix(double[][] a, double[][] b)
```

To multiply matrix **a** by matrix **b**, the number of columns in **a** must be the same as the number of rows in **b**, and the two matrices must have elements of the same or compatible types. Let **c** be the result of the multiplication. Assume the column size of matrix a is **n**. Each element c_{ij} is $a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + \cdots + a_{in} \times b_{nj}$. For example, for two 3×3 matrices **a** and **b**, **c** is

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

where $c_{ij} = a_{i1} \times b_{1j} + a_{i2} \times b_{2j} + a_{i3} \times b_{3j}$.

Write a test program that prompts the user to enter two 3×3 matrices and displays their product. Here is a sample run:

```
Enter matrix1: 1 2 3 4 5 6 7 8 9 Enter
Enter matrix2: 0 2 4 1 4.5 2.2 1.1 4.3 5.2 Enter
The multiplication of the matrices is
 1 2 3      0 2.0 4.0      5.3 23.9 24
 4 5 6   *  1 4.5 2.2  =   11.6 56.3 58.2
 7 8 9      1.1 4.3 5.2    17.9 88.7 92.4
```

***8.7** (*Points nearest to each other*) Listing 8.3 gives a program that finds two points in a two-dimensional space nearest to each other. Revise the program so it finds two points in a three-dimensional space nearest to each other. Use a two-dimensional array to represent the points. Test the program using the following points:

```
double[][] points = {{-1, 0, 3}, {-1, -1, -1}, {4, 1, 1},
  {2, 0.5, 9}, {3.5, 2, -1}, {3, 1.5, 3}, {-1.5, 4, 2},
  {5.5, 4, -0.5}};
```

The formula for computing the distance between two points `(x1, y1, z1)` and `(x2, y2, z2)` is $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

****8.8** (*All closest pairs of points*) Revise Listing 8.3, FindNearestPoints.java, to display all closest pairs of points with the same minimum distance. Here is a sample run:

```
Enter the number of points: 8 Enter
Enter 8 points: 0 0 1 1 -1 -1 2 2 -2 -2 -3 -3 -4 -4 5 5 Enter
The closest two points are (0.0, 0.0) and (1.0, 1.0)
The closest two points are (0.0, 0.0) and (-1.0, -1.0)
The closest two points are (1.0, 1.0) and (2.0, 2.0)
The closest two points are (-1.0, -1.0) and (-2.0, -2.0)
The closest two points are (-2.0, -2.0) and (-3.0, -3.0)
The closest two points are (-3.0, -3.0) and (-4.0, -4.0)
Their distance is 1.4142135623730951
```

*****8.9** (*Game: play a tic-tac-toe game*) In a game of tic-tac-toe, two players take turns marking an available cell in a 3 × 3 grid with their respective tokens (either X or O). When one player has placed three tokens in a horizontal, vertical, or diagonal row on the grid, the game is over and that player has won. A draw (no winner) occurs when all the cells on the grid have been filled with tokens and neither player has achieved a win. Create a program for playing a tic-tac-toe game.

The program prompts two players to alternately enter an X token and O token. Whenever a token is entered, the program redisplays the board on the console and determines the status of the game (win, draw, or continue). Here is a sample run:

```
-------------
|   |   |   |
-------------
|   |   |   |
-------------
|   |   |   |
-------------
Enter a row (0, 1, or 2) for player X: 1 Enter
Enter a column (0, 1, or 2) for player X: 1 Enter

-------------
|   |   |   |
-------------
|   | X |   |
-------------
|   |   |   |
-------------
Enter a row (0, 1, or 2) for player O: 1 Enter
Enter a column (0, 1, or 2) for player O: 2 Enter

-------------
|   |   |   |
-------------
|   | X | O |
-------------
|   |   |   |
-------------
```

```
Enter a row (0, 1, or 2) for player X:

. . .

-------------
| X |   |   |
-------------
| 0 | X | 0 |
-------------
|   |   | X |
-------------
X player won
```

**8.10 (*Largest row and column*) Write a program that randomly fills in 0s and 1s into a 5-by-5 matrix, prints the matrix, and finds the first row and column with the most 1s. Here is a sample run of the program:

```
01101
01011
10010
11111
00101
The largest row index: 3
The largest column index: 4
```

8.11 (*Game: nine heads and tails*) Nine coins are placed in a 3-by-3 matrix with some face up and some face down. You can represent the state of the coins using a 3-by-3 matrix with values **0 (heads) and **1** (tails). Here are some examples:

```
0 0 0    1 0 1    1 1 0    1 0 1    1 0 0
0 1 0    0 0 1    1 0 0    1 1 0    1 1 1
0 0 0    1 0 0    0 0 1    1 0 0    1 1 0
```

Each state can also be represented using a binary number. For example, the preceding matrices correspond to the numbers

```
000010000 101001100 110100001 101110100 100111110
```

There are a total of 512 possibilities, so you can use decimal numbers 0, 1, 2, 3, . . . , and 511 to represent all states of the matrix. Write a program that prompts the user to enter a number between 0 and 511 and displays the corresponding matrix with the characters **H** and **T**. In the following sample run, the user entered **7**, which corresponds to **000000111**. Since **0** stands for **H** and 1 for **T**, the output is correct.

```
Enter a number between 0 and 511: 7 Enter
H H H
H H H
T T T
```

**8.12 (*Financial application: compute tax*) Rewrite Listing 3.5, ComputeTax.java, using arrays. For each filing status, there are six tax rates. Each rate is applied to a certain amount of taxable income. For example, from the taxable income of $400,000 for a single filer, $8,350 is taxed at 10%, (33,950 − 8,350) at 15%,

(82,250 − 33,950) at 25%, (171,550 − 82,550) at 28%, (372,550 − 82,250) at 33%, and (400,000 − 372,950) at 36%. The six rates are the same for all filing statuses, which can be represented in the following array:

```
double[] rates = {0.10, 0.15, 0.25, 0.28, 0.33, 0.35};
```

The brackets for each rate for all the filing statuses can be represented in a two-dimensional array as follows:

```
int[][] brackets = {
  {8350, 33950, 82250, 171550, 372950},   // Single filer
  {16700, 67900, 137050, 20885, 372950},  // Married jointly
                                          // -or qualifying widow(er)
  {8350, 33950, 68525, 104425, 186475},   // Married separately
  {11950, 45500, 117450, 190200, 372950}  // Head of household
};
```

Suppose the taxable income is $400,000 for single filers. The tax can be computed as follows:

```
tax = brackets[0][0] * rates[0] +
  (brackets[0][1] - brackets[0][0]) * rates[1] +
  (brackets[0][2] - brackets[0][1]) * rates[2] +
  (brackets[0][3] - brackets[0][2]) * rates[3] +
  (brackets[0][4] - brackets[0][3]) * rates[4] +
  (400000 - brackets[0][4]) * rates[5];
```

***8.13** (*Locate the smallest element*) Write the following method that returns the location of the smallest element in a two-dimensional array.

```
public static int[] locateSmallest(double[][] a)
```

The return value is a one-dimensional array that contains two elements. These two elements indicate the row and column indices of the smallest element in the two-dimensional array. Write a test program that prompts the user to enter a two dimensional array and displays the location of the smallest element in the array. Here is a sample run:

```
Enter the number of rows and columns of the array: 3 4 Enter
Enter the array:
23.5 35 2 10 Enter
4.5 3 45 3.5 Enter
35 44 5.5 9.6 Enter
The location of the smallest element is at (0, 2)
```

****8.14** (*Explore matrix*) Write a program that prompts the user to enter the length of a square matrix, randomly fills in 0s and 1s into the matrix, prints the matrix, and finds the rows, columns, and diagonals with all 0s or 1s. Here is a sample run of the program:

```
Enter the size for the matrix: 4 Enter
0111
0000
0100
1111
All 0s on row 2
All 1s on row 4
No same numbers on a column
No same numbers on the major diagonal
No same numbers on the sub-diagonal
```

***8.15** (*Geometry: same line?*) Programming Exercise 6.39 gives a method for testing whether three points are on the same line.

Write the following method to test whether all the points in the array `points` are on the same line:

```
public static boolean sameLine(double[][] points)
```

Write a program that prompts the user to enter five points and displays whether they are on the same line. Here are sample runs:

```
Enter five points: 3.4 2 6.5 9.5 2.3 2.3 5.5 5 -5 4 Enter
The five points are not on the same line
```

```
Enter five points: 1 1 2 2 3 3 4 4 5 5 Enter
The five points are on the same line
```

***8.16** (*Sort two-dimensional array*) Write a method to sort a two-dimensional array using the following header:

```
public static void sort(int m[][])
```

The method performs a primary sort on rows, and a secondary sort on columns. For example, the following array

```
{{4, 2},{1, 7},{4, 5},{1, 2},{1, 1},{4, 1}}
```

will be sorted to

```
{{4, 5},{4, 2},{4, 1},{1, 7},{1, 2},{1, 1}}.
```

*****8.17** (*Financial tsunami*) Banks lend money to each other. In tough economic times, if a bank goes bankrupt, it may not be able to pay back the loan. A bank's total assets are its current balance plus its loans to other banks. The diagram in Figure 8.8 shows five banks. The banks' current balances are 25, 125, 175, 75, and 181 million dollars, respectively. The directed edge from node 1 to node 2 indicates that bank 1 lends 40 million dollars to bank 2.

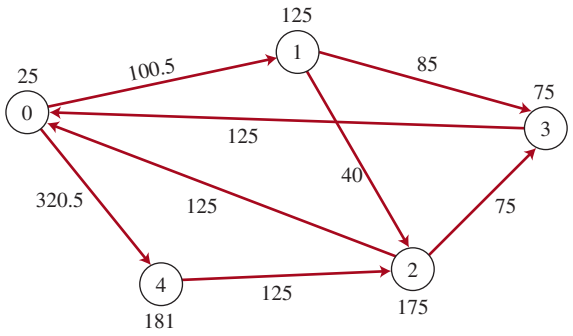


FIGURE 8.8 Banks lend money to each other.

If a bank's total assets are under a certain limit, the bank is unsafe. The money it borrowed cannot be returned to the lender, and the lender cannot count the loan in its total assets. Consequently, the lender may also be unsafe, if its total assets are under the limit. Write a program to find all the unsafe banks. Your program reads the input as follows. It first reads two integers `n` and `limit`, where `n` indicates the number of banks and `limit` is the minimum total assets for keeping a bank safe. It then reads `n` lines that describe the information for `n` banks with IDs from `0` to `n-1`.

The first number in the line is the bank's balance, the second number indicates the number of banks that borrowed money from the bank, and the rest are pairs of two numbers. Each pair describes a borrower. The first number in the pair is the borrower's ID and the second is the amount borrowed. For example, the input for the five banks in Figure 8.8 is as follows (note the limit is 201):

```
5 201
25 2 1 100.5 4 320.5
125 2 2 40 3 85
175 2 0 125 3 75
75 1 0 125
181 1 2 125
```

The total assets of bank 3 are (75 + 125), which is under 201, so bank 3 is unsafe. After bank 3 becomes unsafe, the total assets of bank 1 fall below (125 + 40). Thus, bank 1 is also unsafe. The output of the program should be

```
Unsafe banks are 3 1
```

(*Hint*: Use a two-dimensional array `borrowers` to represent loans. `borrowers[i][j]` indicates the loan that bank `i` provides to bank `j`. Once bank `j` becomes unsafe, `borrowers[i][j]` should be set to `0`.)

***8.18** (*Shuffle rows*) Write a method that shuffles the rows in a two-dimensional `int` array using the following header:

```
public static void shuffle(int[][] m)
```

Write a test program that shuffles the following matrix:

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

****8.19** (*Pattern recognition: four consecutive equal numbers*) Write the following method that tests whether a two-dimensional array has four consecutive numbers of the same value, either horizontally, vertically, or diagonally:

```
public static boolean isConsecutiveFour(int[][] values)
```

Write a test program that prompts the user to enter the number of rows and columns of a two-dimensional array then the values in the array, and displays true if the array contains four consecutive numbers with the same value. Otherwise, the program displays false. Here are some examples of the true cases:

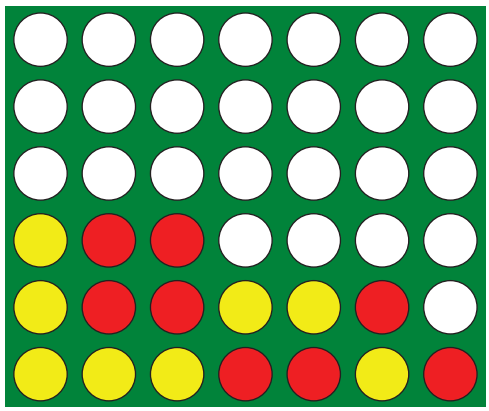
0	1	0	3	1	6	1
0	1	6	8	6	0	1
5	6	2	1	8	2	9
6	5	6	1	1	9	1
1	3	6	1	4	0	7
3	3	3	3	4	0	7

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

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

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

*****8.20** (*Game: connect four*) Connect four is a two-player board game in which the players alternately drop colored disks into a seven-column, six-row vertically suspended grid, as shown below.



The objective of the game is to connect four same-colored disks in a row, a column, or a diagonal before your opponent can do likewise. The program prompts two players to drop a red or yellow disk alternately. Whenever a disk is dropped, the program redisplays the board on the console and determines the status of the game (win, draw, or continue). Here is a sample run:

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
-----------------------------
Drop a red disk at column (0–6): 0 Enter

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | | | | | |
-----------------------------
```

```
Drop a yellow disk at column (0–6): 3 Enter

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | |Y| | | |

 . . .
 . . .
 . . .

Drop a yellow disk at column (0–6): 6 Enter

| | | | | | | |
| | | | | | | |
| | | |R| | | |
| | | |Y|R|Y| |
| | |R|Y|Y|Y|Y|
|R|Y|R|Y|R|R|R|
-----------------------------
The yellow player won
```

***8.21** (*Central city*) Given a set of cities, the central city is the city that has the shortest total distance to all other cities. Write a program that prompts the user to enter the number of cities and the locations of the cities (coordinates), and finds the central city and its total distance to all other cities.

```
Enter the number of cities: 5 Enter
Enter the coordinates of the cities:
  2.5 5 5.1 3 1 9 5.4 54 5.5 2.1 Enter
The central city is at (2.5, 5.0)
The total distance to all other cities is 60.81
```

VideoNote
Even number of 1s

***8.22** (*Even number of 1s*) Write a program that generates a 6-by-6 two-dimensional matrix filled with 0s and 1s, displays the matrix, and checks if every row and every column have an even number of 1s.

***8.23** (*Game: find the flipped cell*) Suppose you are given a 6-by-6 matrix filled with 0s and 1s. All rows and all columns have an even number of 1s. Let the user flip one cell (i.e., flip from 1 to 0 or from 0 to 1) and write a program to find which cell was flipped. Your program should prompt the user to enter a 6-by-6 array with 0s and 1s and find the first row *r* and first column *c* where the even number of the 1s property is violated (i.e., the number of 1s is not even). The flipped cell is at (*r*, *c*). Here is a sample run:

```
Enter a 6-by-6 matrix row by row:
1 1 1 0 1 1 Enter
1 1 1 1 0 0 Enter
0 1 0 1 1 1 Enter
1 1 1 1 1 1 Enter
0 1 1 1 1 0 Enter
1 0 0 0 0 1 Enter
The flipped cell is at (0, 1)
```

***8.24** (*Check Sudoku solution*) Listing 8.4 checks whether a solution is valid by checking whether every number is valid in the board. Rewrite the program by checking whether every row, every column, and every small box has the numbers 1 to 9.

***8.25** (*Markov matrix*) An $n \times n$ matrix is called a *positive Markov matrix* if each element is positive and the sum of the elements in each column is 1. Write the following method to check whether a matrix is a Markov matrix:

```
public static boolean isMarkovMatrix(double[][] m)
```

Write a test program that prompts the user to enter a 3×3 matrix of double values and tests whether it is a Markov matrix. Here are sample runs:

```
Enter a 3-by-3 matrix row by row:
0.15 0.875 0.375 Enter
0.55 0.005 0.225 Enter
0.30 0.12 0.4 Enter
It is a Markov matrix
```

```
Enter a 3-by-3 matrix row by row:
0.95 -0.875 0.375 Enter
0.65 0.005 0.225 Enter
0.30 0.22 -0.4 Enter
It is not a Markov matrix
```

***8.26** (*Row sorting*) Implement the following method to sort the rows in a two-dimensional array. A new array is returned and the original array is intact.

```
public static double[][] sortRows(double[][] m)
```

Write a test program that prompts the user to enter a 4×4 matrix of double values and displays a new row-sorted matrix. Here is a sample run:

```
Enter a 4-by-4 matrix row by row:
0.15 0.875 0.375 0.225 Enter
0.55 0.005 0.225 0.015 Enter
0.30 0.12 0.4 0.008 Enter
0.07 0.021 0.14 0.2 Enter

The row-sorted array is
0.15 0.225 0.375 0.875
0.005 0.015 0.225 0.55
0.008 0.12 0.30 0.4
0.021 0.07 0.14 0.2
```

***8.27** (*Column sorting*) Implement the following method to sort the columns in a two dimensional array. A new array is returned and the original array is intact.

```
public static double[][] sortColumns(double[][] m)
```

Write a test program that prompts the user to enter a 4 × 4 matrix of double values and displays a new column-sorted matrix. Here is a sample run:

```
Enter a 4-by-4 matrix row by row:
0.15 0.875 0.375 0.225 Enter
0.55 0.005 0.225 0.015 Enter
0.30 0.12 0.4 0.008 Enter
0.07 0.021 0.14 0.2 Enter

The column-sorted array is
0.07 0.005 0.14 0.008
0.15 0.021 0.225 0.015
0.30 0.12 0.375 0.2
0.55 0.875 0.4 0.225
```

8.28 (*Strictly identical arrays*) The two-dimensional arrays **m1** and **m2** are *strictly identical* if their corresponding elements are equal. Write a method that returns **true** if **m1** and **m2** are strictly identical, using the following header:

```
public static boolean equals(int[][] m1, int[][] m2)
```

Write a test program that prompts the user to enter two 3 × 3 arrays of integers and displays whether the two are strictly identical. Here are the sample runs:

```
Enter list1: 51 22 25 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are strictly identical
```

```
Enter list1: 51 25 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are not strictly identical
```

8.29 (*Identical arrays*) The two-dimensional arrays **m1** and **m2** are *identical* if they have the same contents. Write a method that returns **true** if **m1** and **m2** are identical, using the following header:

```
public static boolean equals(int[][] m1, int[][] m2)
```

Write a test program that prompts the user to enter two 3 × 3 arrays of integers and displays whether the two are identical. Here are the sample runs:

```
Enter list1: 51 25 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are identical
```

```
Enter list1: 51 5 22 6 1 4 24 54 6 Enter
Enter list2: 51 22 25 6 1 4 24 54 6 Enter
The two arrays are not identical
```

***8.30** (*Algebra: solve linear equations*) Write a method that solves the following 2 × 2 system of linear equations:

$$a_{00}x + a_{01}y = b_0$$
$$a_{10}x + a_{11}y = b_1$$

$$x = \frac{b_0a_{11} - b_1a_{01}}{a_{00}a_{11} - a_{01}a_{10}} \quad y = \frac{b_1a_{00} - b_0a_{10}}{a_{00}a_{11} - a_{01}a_{10}}$$

The method header is:

```
public static double[] linearEquation(double[][] a, double[] b)
```

The method returns `null` if $a_{00}a_{11} - a_{01}a_{10}$ is `0`. Write a test program that prompts the user to enter a_{00}, a_{01}, a_{10}, a_{11}, b_0, and b_1 and displays the result. If $a_{00}a_{11} - a_{01}a_{10}$ is `0`, report that "The equation has no solution." A sample run is similar to Programming Exercise 3.3.

***8.31** (*Geometry: intersecting point*) Write a method that returns the intersecting point of two lines. The intersecting point of the two lines can be found by using the formula given in Programming Exercise 3.25. Assume that (`x1`, `y1`) and (`x2`, `y2`) are the two points on line 1 and (`x3`, `y3`) and (`x4`, `y4`) are on line 2. The method header is:

```
public static double[] getIntersectingPoint(double[][] points)
```

The points are stored in a 4-by-2 two-dimensional array `points` with (`points[0][0]`, `points[0][1]`) for (`x1`, `y1`). The method returns the intersecting point or null if the two lines are parallel. Write a program that prompts the user to enter four points and displays the intersecting point. See Programming Exercise 3.25 for a sample run.

***8.32** (*Geometry: area of a triangle*) Write a method that returns the area of a triangle using the following header:

```
public static double getTriangleArea(double[][] points)
```

The points are stored in a 3-by-2 two-dimensional array `points` with `points[0][0]` and `points[0][1]` for (`x1`, `y1`). The triangle area can be computed using the formula in Programming Exercise 2.19. The method returns `0` if the three points are on the same line. Write a program that prompts the user to enter three points of a triangle and displays the triangle's area. Here are the sample runs:

```
Enter x1, y1, x2, y2, x3, y3: 2.5 2 5 -1.0 4.0 2.0 Enter
The area of the triangle is 2.25
```

```
Enter x1, y1, x2, y2, x3, y3: 2 2 4.5 4.5 6 6 Enter
The three points are on the same line
```

***8.33** (*Geometry: polygon subareas*) A convex four-vertex polygon is divided into four triangles, as shown in Figure 8.9.

Write a program that prompts the user to enter the coordinates of four vertices and displays the areas of the four triangles in increasing order. Here is a sample run:

```
Enter x1, y1, x2, y2, x3, y3, x4, y4:
  -2.5 2 4 4 3 -2 -2 -3.5 Enter
The areas are 6.17 7.96 8.08 10.42
```

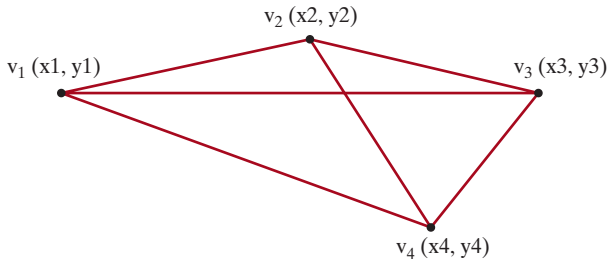


FIGURE 8.9 A four-vertex polygon is defined by four vertices.

***8.34** (*Geometry: rightmost lowest point*) In computational geometry, often you need to find the rightmost lowest point in a set of points. Write the following method that returns the rightmost lowest point in a set of points:

```
public static double[]
  getRightmostLowestPoint(double[][] points)
```

Write a test program that prompts the user to enter the coordinates of six points and displays the rightmost lowest point. Here is a sample run:

```
Enter 6 points: 1.5 2.5 -3 4.5 5.6 -7 6.5 -7 8 1 10 2.5 Enter
The rightmost lowest point is (6.5, -7.0)
```

****8.35** (*Largest block*) Given a square matrix with the elements 0 or 1, write a program to find a maximum square submatrix whose elements are all 1s. Your program should prompt the user to enter the number of rows in the matrix. The program then displays the location of the first element in the maximum square submatrix and the number of rows in the submatrix. Here is a sample run:

```
Enter the number of rows in the matrix: 5 Enter
Enter the matrix row by row:
1 0 1 0 1 Enter
1 1 1 0 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter
1 0 1 1 1 Enter

The maximum square submatrix is at (2, 2) with size 3
```

Your program should implement and use the following method to find the maximum square submatrix:

```
public static int[] findLargestBlock(int[][] m)
```

The return value is an array that consists of three values. The first two values are the row and column indices for the first element in the submatrix, and the third value is the number of the rows in the submatrix. For an animation of this problem, see https://liveexample.pearsoncmg.com/dsanimation/LargestBlockeBook.html.

**8.36 (*Latin square*) A Latin square is an *n*-by-*n* array filled with `n` different Latin letters, each occurring exactly once in each row and once in each column. Write a program that prompts the user to enter the number `n` and the array of characters, as shown in the sample output, and checks if the input array is a Latin square. The characters are the first `n` characters starting from `A`.

```
Enter number n: 4 Enter
Enter 4 rows of letters separated by spaces:
A B C D Enter
B A D C Enter
C D B A Enter
D C A B Enter
The input array is a Latin square
```

```
Enter number n: 3 Enter
Enter 3 rows of letters separated by spaces:
A F D Enter
Wrong input: the letters must be from A to C
```

**8.37 (*Guess the capitals*) Write a program that repeatedly prompts the user to enter a capital for a state. Upon receiving the user input, the program reports whether the answer is correct. Assume that `50` states and their capitals are stored in a two-dimensional array, as shown in Figure 8.10. The program prompts the user to answer all states' capitals and displays the total correct count. The user's answer is not case-sensitive.

Alabama	Montgomery
Alaska	Juneau
Arizona	Phoenix
...	...
...	...

FIGURE 8.10 A two-dimensional array stores states and their capitals.

Here is a sample run:

```
What is the capital of Alabama? Montogomery Enter
The correct answer should be Montgomery
What is the capital of Alaska? Juneau Enter
Your answer is correct
What is the capital of Arizona? ...
...
The correct count is 35
```