Chapter 8
Multidimensional Arrays

8.1 Introduction 290
8.2 Two-Dimensional Array Basics 290
8.3 Processing Two-Dimensional Arrays 293
8.4 Passing Two-Dimensional Arrays to Methods 295
8.5 Case Study: Grading a Multiple-Choice Test 296
8.6 Case Study: Finding the Closest Pair 298
8.7 Case Study: Sudoku 300
8.8 Multidimensional Arrays 303
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).
Chapter 8
Multidimensional Arrays

8.1 Introduction 290

- Thus far, you have used one-dimensional arrays to model linear collections of elements. You can use a two-dimensional array to represent a matrix or a table. For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

<table>
<thead>
<tr>
<th></th>
<th>Chicago</th>
<th>Boston</th>
<th>New York</th>
<th>Atlanta</th>
<th>Miami</th>
<th>Dallas</th>
<th>Houston</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>0</td>
<td>983</td>
<td>787</td>
<td>714</td>
<td>1375</td>
<td>967</td>
<td>1087</td>
</tr>
<tr>
<td>Boston</td>
<td>983</td>
<td>0</td>
<td>214</td>
<td>1102</td>
<td>1763</td>
<td>1723</td>
<td>1842</td>
</tr>
<tr>
<td>New York</td>
<td>787</td>
<td>214</td>
<td>0</td>
<td>888</td>
<td>1549</td>
<td>1548</td>
<td>1627</td>
</tr>
<tr>
<td>Atlanta</td>
<td>714</td>
<td>1102</td>
<td>888</td>
<td>0</td>
<td>661</td>
<td>781</td>
<td>810</td>
</tr>
<tr>
<td>Miami</td>
<td>1375</td>
<td>1763</td>
<td>1549</td>
<td>661</td>
<td>0</td>
<td>1426</td>
<td>1187</td>
</tr>
<tr>
<td>Dallas</td>
<td>967</td>
<td>1723</td>
<td>1548</td>
<td>781</td>
<td>1426</td>
<td>0</td>
<td>239</td>
</tr>
<tr>
<td>Houston</td>
<td>1087</td>
<td>1842</td>
<td>1627</td>
<td>810</td>
<td>1187</td>
<td>239</td>
<td>0</td>
</tr>
</tbody>
</table>

```cpp
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 290

- You can use 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.

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

- Here is the syntax for declaring a two-dimensional array:

  ```java
dataType [][] arrayRefVar;
```

  or

  ```java
dataType arrayRefVar[][]; // This style is correct, but not preferred
```

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

  ```java
  int [][] matrix;
  or
  int matrix[][]; // This style is correct, but not preferred
  ```

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

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

![Matrix Diagram](image)

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

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

Equivalent

```java
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;
```
8.2.2 Obtaining the Lengths of Two-Dimensional Arrays

```java
int[ ][ ] x = new int[3][4];
```

x.length is 3
x[0].length is 4, x[1].length is 4, x[2].length is 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

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as a ragged array.

```java
int[][] triangleArray = {
    {1, 2, 3, 4, 5},
    {2, 3, 4, 5},
    {3, 4, 5},
    {4, 5},
    {5}
};
```

- If you don’t know the values in a raged array in advance, but know the sizes, say the same as before, you can create a ragged array using the syntax that follows:

```java
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,

```java
triangleArray[0][3] = 4;
triangleArray[4][0] = 5;
```
8.3 Processing Two-Dimensional Arrays 293

- Suppose an array matrix is declared as follows:

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

- Here are some examples of processing two-dimensional arrays:
  - (Initializing arrays with input values) The following loop initializes the array with user input values:
    ```java
    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();
      }
    }
    ```
  - (Initializing arrays with random values) You can now assign random values to the array using the following loop:
    ```java
    for (int row = 0; row < triangleArray.length; row++)
    for (int column = 0; column < triangleArray[row].length; column++)
      triangleArray[row][column] = (int) (Math.random() * 1000);
    ```
  - (Printing arrays)
    ```java
    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();
    }
    ```
  - (Summing all elements)
  - (Summing elements by column)
  - (Which row has the largest sum?)
  - (Random shuffling)
8.4 Passing Two-Dimensional Arrays to Methods 295

- You can pass a two-dimensional array to a method just as you pass a one-dimensional array.
- Listing 8.1 gives an example with a method that returns the sum of all the elements in a matrix.

**LISTING 8.1 PassTwoDimensionalArray.java**

```java
import java.util.Scanner;

public class PassTwoDimensionalArray {
    public static void main(String[] args) {
        int[][] m = getArray(); // Get an array

        // Display sum of elements
        System.out.println("Sum of all elements is " + sum(m));
    }

    public static int[][] getArray() { // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Enter array values
        int[][] m = new int[3][4];
        System.out.println("Enter "+m.length + " rows and "+m[0].length + " columns: ");
        for (int i = 0; i < m.length; i++)
            for (int j = 0; j < m[i].length; j++)
                m[i][j] = input.nextInt();

        return m;
    }

    public static int sum(int[][] m) { // Enter array values
        int total = 0;
        for (int row = 0; row < m.length; row++) {
            for (int column = 0; column < m[row].length; column++) {
                total += m[row][column];
            }
        }

        return total;
    }
}
```

Enter 3 rows and 4 columns:
```
1 2 3 4
5 6 7 8
9 10 11 12
```
Sum of all elements is 78
8.5 Case Study: Grading a Multiple-Choice Test

- Objective: write a program that grades multiple-choice test.
- Suppose 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:

```
   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 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 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
```

### Students' Answers to the Questions:

```
A B A C C D E E A D
D B A B C A E E A D
E D D A C B E E A D
C B A E D C E E A D
A B D C C D E E A D
B B E C C D E E A D
B B A C C D E E A D
E B E C C D E E A D
```

### 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
```

### LISTING 8.2 GradeExam.java

```
public class GradeExam {
   /** Main method */
   public static void main(String args[]) {
      // Students' answers to the questions
      char[][] answers = {
         {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
         {'E', 'D', 'D', 'A', 'C', 'B', 'E', 'E', 'A', 'D'},
         {'C', 'B', 'A', 'E', 'D', 'C', 'E', 'E', 'A', 'D'},
         {'A', 'B', 'D', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
         {'B', 'B', 'E', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
         {'B', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      // Key to the questions

      // Grade all answers
      for (int i = 0; i < answers.length; i++) {
         // Grade one student
         int correctCount = 0;
         for (int j = 0; j < answers[i].length; j++) {
            if (answers[i][j] == keys[j])
               correctCount++;
         }
         System.out.println("Student " + i + "'s correct count is " +
               correctCount);
      }
   }
}
```
<table>
<thead>
<tr>
<th>Student</th>
<th>Correct Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
8.6 Case Study: Finding the Closest Pair 298

• The GPS navigation system is becoming increasingly popular.
• The system uses the graph and geometric algorithms to calculate distances and map a route.
• The section presents a geometric problem for finding a closest pair of point.

![Graph of points](image)

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

**LISTING 8.3** FinNearestPoints.java

• Given a set of points, the closest-pair problem is to find the two points that are nearest to each other.

```java
import java.util.Scanner;

public class FinNearestPoints {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the number of points: ");
        int numberOfPoints = input.nextInt();

        // Create an array to store points
        double[][] points = new double[numberOfPoints][2];
        System.out.print("Enter " + numberOfPoints + " points: ");
        for (int i = 0; i < points.length; i++) {
            points[i][0] = input.nextDouble();
            points[i][1] = input.nextDouble();
        }

        // p1 and p2 are the indices in the points array
        int p1 = 0, p2 = 1; // Initial two points
        double shortestDistance = distance(points[p1][0], points[p1][1],
                                           points[p2][0], points[p2][1]); // Initialize shortestDistance

        // Compute distance for every two points
        for (int i = 0; i < points.length; i++) {
            for (int j = i + 1; j < points.length; j++) {
                double distance = distance(points[i][0], points[i][1],
                                             points[j][0], points[j][1]); // Find distance
            }
        }
    }
}
```

**LISTING 8.3 FinNearestPoints.java**

• Given a set of points, the closest-pair problem is to find the two points that are nearest to each other.

```java
import java.util.Scanner;

public class FindNearestPoints {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter the number of points: ");
        int numberOfPoints = input.nextInt();

        // Create an array to store points
        double[][] points = new double[numberOfPoints][2];
        System.out.print("Enter " + numberOfPoints + " points: ");
        for (int i = 0; i < points.length; i++) {
            points[i][0] = input.nextDouble();
            points[i][1] = input.nextDouble();
        }

        // p1 and p2 are the indices in the points array
        int p1 = 0, p2 = 1; // Initial two points
        double shortestDistance = distance(points[p1][0], points[p1][1],
                                           points[p2][0], points[p2][1]); // Initialize shortestDistance

        // Compute distance for every two points
        for (int i = 0; i < points.length; i++) {
            for (int j = i + 1; j < points.length; j++) {
                double distance = distance(points[i][0], points[i][1],
                                             points[j][0], points[j][1]); // Find distance
            }
        }
    }
}
```
if (shortestDistance > distance) {
    p1 = i; // Update p1
    p2 = j; // Update p2
    shortestDistance = distance; // Update shortestDistance
}
}

// Display result
System.out.println("The closest two points are "+
    "(" + points[p1][0] + ", " + points[p1][1] + ") and (" +
    points[p2][0] + ", " + points[p2][1] + ")");

/** Compute the distance between two points (x1, y1) and (x2, y2)*/
public static double distance(
    double x1, double y1, double x2, double y2) {
    return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}

Enter the number of points: 8
Enter 8 points: -1 3 -1 -1 1 1 2 0.5 2 -1 3 3 4 2 4 -0.5
The closest two points are (1.0, 1.0) and (2.0, 0.5)
8.7 Case Study: Sudoku 300

- This section presents an interesting problem of a sort that appears in the newspaper every day: Sudoku
- Sudoku is a 9 X 9 grid divided into smaller 3 X 3 boxes (also called regions or blocks) as shown in Figure 8.4(a).
- 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 numbers 1 to 9 so that every row, every column, and every 3 X 3 box contains the numbers 1 to 9 as shown in Figure 8.4 (b).

![Sudoku Puzzle and Solution](image)

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

LISTING 8.4 CheckSudokuSolution.java

```java
import java.util.Scanner;

public class CheckSudokuSolution {
    public static void main(String[] args) {
        // Read a Sudoku solution
        int[][] grid = readASolution();

        System.out.println(isValid(grid) ? "Valid solution" : "Invalid solution");
    }

    /** Read a Sudoku solution from the console */
    public static int[][] readASolution() {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
```
```java
System.out.println("Enter a Sudoku puzzle solution:");
int[][] grid = new int[9][9];
for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
        grid[i][j] = input.nextInt();

return grid;

/** Check whether a solution is valid */
public static boolean isValid(int[][] grid) {
    // Check whether each row has numbers 1 to 9
    for (int i = 0; i < 9; i++)
        if (!is1To9(grid[i])) // If grid[i] does not contain 1 to 9
            return false;

    // Check whether each column has numbers 1 to 9
    for (int j = 0; j < 9; j++) {
        // Obtain a column in the one-dimensional array
        int[] column = new int[9];
        for (int i = 0; i < 9; i++)
            column[i] = grid[i][j];

        if (!is1To9(column)) // If column does not contain 1 to 9
            return false;
    }

    // Check whether each 3 by 3 box has numbers 1 to 9
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            // The starting element in a small 3 by 3 box
            int k = 0;
            int[] list = new int[9]; // Get all number in the box to list
            for (int row = i * 3; row < i * 3 + 3; row++)
                for (int column = j * 3; column < j * 3 + 3; column++)
                    list[k++] = grid[row][column];

            if (!is1To9(list)) // If list does not contain 1 to 9
                return false;
        }
    }

    return true; // The fixed cells are valid
}

/** Check whether the one-dimensional array contains 1 to 9 */
public static boolean is1To9(int[] list) {
    // Make a copy of the array
    int[] temp = new int[list.length];
    System.arraycopy(list, 0, temp, 0, list.length);

    // Sort the array
    java.util.Arrays.sort(temp);

    // Check if list contains 1, 2, 3, ..., 9
    for (int i = 0; i < 9; i++)
```
if (temp[i] != i + 1)
    return false;

return true; // The list contains exactly 1 to 9
}

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

Valid solution
8.8 Multidimensional Arrays 303

- Occasionally, you will need to represent n-dimensional data structures. In Java, you can create n-dimensional arrays for any integer n.
- The following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores:

```java
double[][][] x = new double[2][3][4];
```

- For example, you may use a three-dimensional array to store exam scores for a class of 6 students with 5 exams, and each exam has 2 parts (multiple-choice and essay).

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

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

```java
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}}};
```

```
Which student | Which exam | Multiple-choice or essay
```
```
scores[i][j][k]
```

scores[i][j][k] Which student Which exam Multiple-choice or essay
8.8.2 Case Study: Guessing Birthdays

- The program can guess your birth date.

Note: 19 is 10011 in binary. 7 is 111 in binary. 23 is 11101 in binary

\[ 10000 + 10011 \]
\[ 1001 \]
\[ 00111 \]
\[ 11101 \]
\[ 19 \]
\[ 7 \]
\[ 23 \]

LISTING 8.6 GuessBirthdayUsingArray.java

```java
import java.util.Scanner;

public class GuessBirthdayUsingArray {
    public static void main(String[] args) {
        int day = 0; // Day to be determined
        int answer;

        int[][][] dates = {
            {{1, 3, 5, 7},
             {9, 11, 13, 15},
             {17, 19, 21, 23},
             {25, 27, 29, 31}},
            {{2, 3, 6, 7},
             {10, 11, 14, 15},
             {18, 19, 22, 23},
             {26, 27, 30, 31}},
            {{4, 5, 6, 7},
             {12, 13, 14, 15},
             {20, 21, 22, 23},
             {28, 29, 30, 31}},
            {{8, 9, 10, 11},
             {12, 13, 14, 15},
             {24, 25, 26, 27},
             {28, 29, 30, 31}}
        };

        // Create a Scanner
Scanner input = new Scanner(System.in);

for (int i = 0; i < 5; i++) {
    System.out.println("Is your birth day in Set" + (i + 1) + "?" + (i + 1) + "?");
    for (int j = 0; j < 4; j++) {
        for (int k = 0; k < 4; k++)
            System.out.printf("%4d", dates[i][j][k]);
        System.out.println();
    }
    System.out.print("Enter 0 for No and 1 for Yes: ");
    answer = input.nextInt();

    if (answer == 1)
        day += dates[i][0][0];
}

System.out.println("Your birth day is " + day);