# Chapter 7

# Multidimensional Arrays

## 7.1 Introduction

- 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.

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 |

## 7.2 Two-Dimension Array Basics

- 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.

### 7.2.1 Declaring Variables of Two-Dimensional Arrays and Creating Two-Dimensional Arrays

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

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

or

```
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

```
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:

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

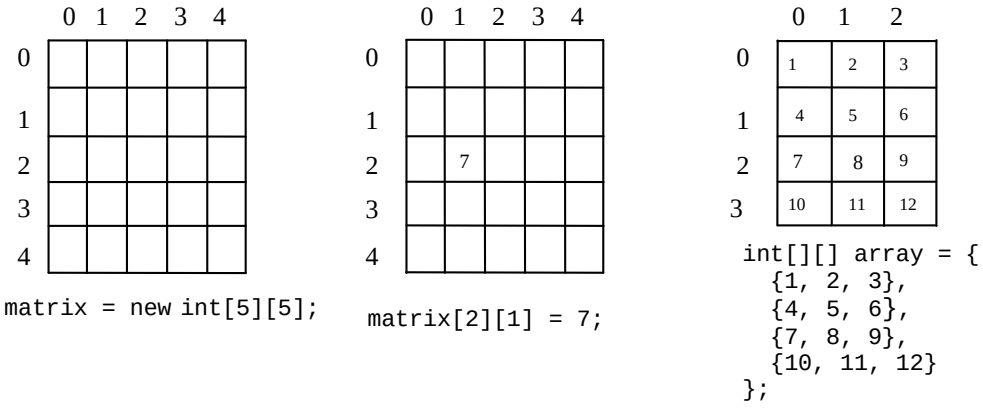



FIGURE 7.1 The index of each subscript of a multidimensional array is an int value starting from 0.

## *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,

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

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;
```

## 7.2.2 Obtaining the Lengths of Two-Dimensional Arrays

**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

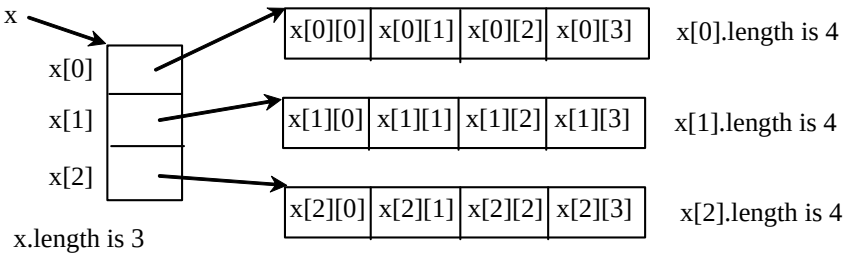
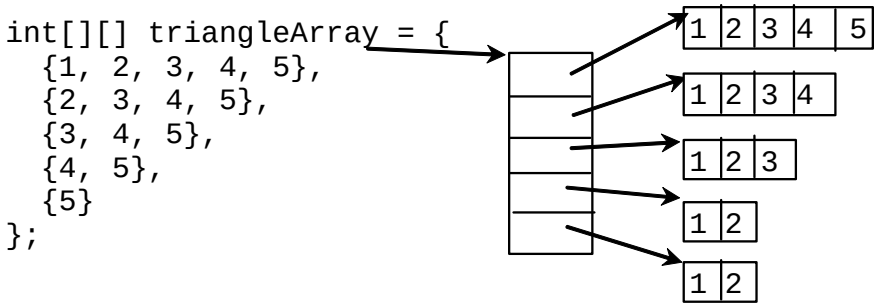



FIGURE 7.2 A two-dimensional array is a one-dimensional array in which each element is another one-dimensional array.

## 7.2.3 Ragged Arrays

- Each row in a two-dimensional array is itself an array. Thus, the rows can have different lengths.




- 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:

```
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];
```

## 7.3 Processing Two-Dimensional Arrays

- Suppose an array matrix is declared as follows:

  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.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:

```
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)

```
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?)

## 7.4 Passing Two-Dimensional Arrays to Methods

- You can pass a two-dimensional array to a method just as you pass a one-dimensional array.
- Listing 7.1 gives an example with a method that returns the sum of all the elements in a matrix.

## LISTING 7.1 PassTwoDimensionalArray.java (Page 240)

```
import java.util.Scanner;

public class PassTwoDimensionalArray {
  public static void main(String[] args) {
    // 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();

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

  public static int sum(int[][] m) {
    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
```

## 7.5 Example: 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:

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 |

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 7.2 GradeExam.java: Grading a Multiple-Choice Test

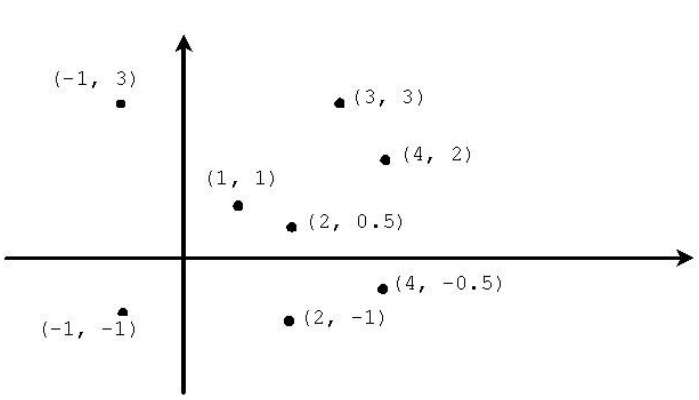
```
public class GradeExam {
  /** Main method */
  public static void main(String args[]) {
    // Students' answers to the questions
    char[][] answers = {
      {'A', 'B', 'A', 'C', 'C', 'D', 'E', 'E', 'A', 'D'},
      {'D', 'B', 'A', 'B', 'C', 'A', 'E', 'E', 'A', 'D'},
      {'E', 'D', 'D', 'A',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

    // 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);
```

```
    }
  }
}
```

```
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
```

## 7.6 Problem: Finding a Closest Pair

- 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.




| | 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 7.3 Points can be represented in a two-dimensional array.

## LISTING 7.3 FinNearestPoints.java (Page 243)

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

```
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)
```

## 7.7 Problem: Sudoku

- 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 7.4(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 7.4 (b).

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 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 7.4 The Sudoku puzzle in (a) is solved in (b).

## LISTING 7.3 CheckSudokuSolution.java (Page 245)

```
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);

    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
```

## 7.8 Multidimensional Arrays

- The following syntax declares a three-dimensional array variable scores, creates an array, and assigns its reference to scores:

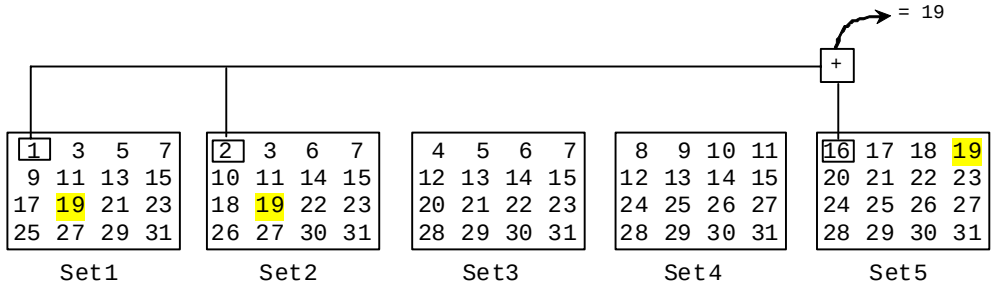
```
double [ ] [ ] [ ] x = new double[2][3][4];
```

**double[ ][ ] [ ] x = new double[2][3][4];**

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

## 7.8.2 Problem: 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        00110        10000
     10           10         1000
+     1    +       1          100
  10011        00111    +       1
                           11101

  19           7            23
```

## LISTING 7.6 GuessBirthdayUsingArray.java (Page 250)

```
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}},
      {{16, 17, 18, 19},
       {20, 21, 22, 23},
       {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) + "?");
       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("\nEnter 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);
  }
}
```

```
Is your birth day in Set1?
    1   3   5   7
    9  11  13  15
   17  19  21  23
   25  27  29  31

Enter 0 for No and 1 for Yes: 1
Is your birth day in Set2?
   2   3   6   7
  10  11  14  15
  18  19  22  23
  26  27  30  31

Enter 0 for No and 1 for Yes: 1
Is your birth day in Set3?
   4   5   6   7
  12  13  14  15
  20  21  22  23
  28  29  30  31

Enter 0 for No and 1 for Yes: 0
Is your birth day in Set4?
   8   9  10  11
  12  13  14  15
  24  25  26  27
  28  29  30  31

Enter 0 for No and 1 for Yes: 0
Is your birth day in Set5?
  16  17  18  19
  20  21  22  23
  24  25  26  27
  28  29  30  31

Enter 0 for No and 1 for Yes: 1
Your birth day is 19
```