Chapter 8

Multi-Dimensional Arrays

Objectives

- To represent data using two-dimensional arrays
- To declare variables for two-dimensional arrays, create arrays, and access array elements in a two-dimensional array using row and column indexes
- To pass two-dimensional arrays to methods
- To use multidimensional arrays

Two-dimensional array: introduction

- *Data in a table or a matrix can be represented using a two-dimensional array.*
- For example, the following table that lists the distances between cities can be stored using a two-dimensional array named **distances**.

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

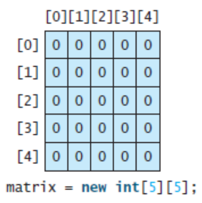
```
double[][] distances = {
  {0, 983, 787, 714, 1375, 967, 1087},
  {983, 0, 214, 1102, 1763, 1723, 1842},
  {787, 214, 0, 888, 1549, 1548, 1627},
  {714, 1102, 888, 0, 661, 781, 810},
  {1375, 1763, 1549, 661, 0, 1426, 1187},
  {967, 1723, 1548, 781, 1426, 0, 239},
  {1087, 1842, 1627, 810, 1187,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

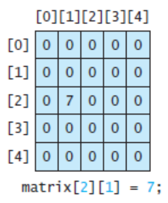
- An element in a two-dimensional array is accessed through a row and column index.

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

- The syntax for declaring a two-dimensional array is:
  - elementType[][] arrayRefVar;

  or
  - elementType arrayRefVar[][]; // Allowed, but not preferred
- As an example, here is how you would declare a two-dimensional array variable **matrix** of **int** values: **int**[][] matrix;
- 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**];
- To assign the value **7** to a specific element at row **2** and column **1**, you can use the following syntax: matrix[**2**][**1**] = **7**;





- 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.

Array initializer

- 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, which 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)

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

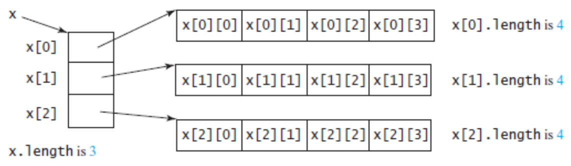
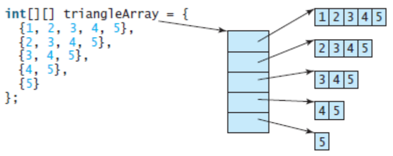

x
x[0]
x[1]
x[2]
x.length is 3
x[0][0] x[0][1] x[0][2] x[0][3]
x[0].length is 4
x[1][0] x[1][1] x[1][2] x[1][3]
x[1].length is 4
x[2][0] x[2][1] x[2][2] x[2][3]
x[2].length is 4

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

Ragged Arrays

- 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*.
  The figure is an example of creating such a ragged array.



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

- 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];
```

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

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

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

Processing Two-Dimensional Arrays

- *Nested* **for** *loops are often used to process a two-dimensional array.*
- Suppose an array **matrix** is created as follows:

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

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

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

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

Processing Two-Dimensional Arrays (cont'd)

2. *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];
  }
}
```

3. 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.

Attempt to implement this on your own before looking into the textbook for suggested solution (pg. 291).

Case Study: Grading a Multiple-Choice Test

- *The problem is to write a program that will grade 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, and the key is stored in a one-dimensional array.

Students' Answers to the Questions:

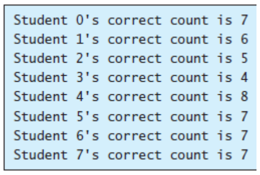
	0	1	2	3	4	5	6	7	8	9
Student 0	A	B	A	C	C	D	E	E	A	D
Student 1	D	B	A	B	C	A	E	E	A	D
Student 2	E	D	D	A	C	B	E	E	A	D
Student 3	C	B	A	E	D	C	E	E	A	D
Student 4	A	B	D	C	C	D	E	E	A	D
Student 5	B	B	E	C	C	D	E	E	A	D
Student 6	B	B	A	C	C	D	E	E	A	D
Student 7	E	B	E	C	C	D	E	E	A	D

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

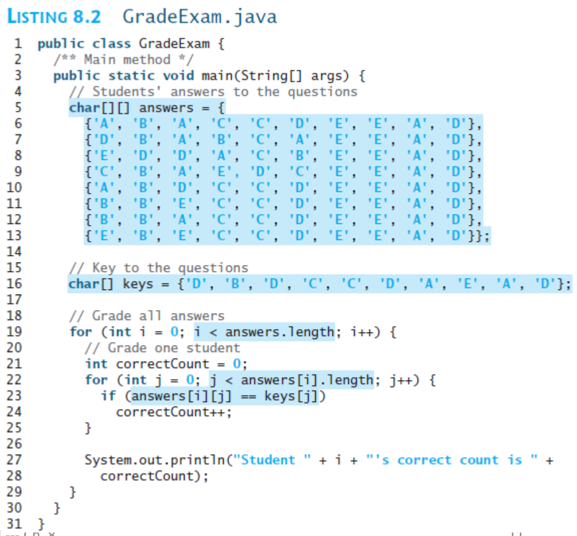
Case Study: Grading a Multiple-Choice Test

- 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.



```
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
```

- Read the case study on "Finding the Closest Pair" and Sudoku for more examples on two-dimensional arrays.



LISTING 8.2 GradeExam.java

```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'},
      {'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);
    }
  }
}
```

Multidimensional Arrays

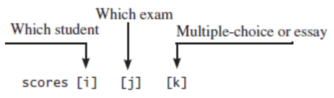
- In Java, you can create *n*-dimensional arrays for any integer *n*.
- A multidimensional array is actually an array in which each element is another array. A three-dimensional array consists of an array of two-dimensional arrays.
- A two-dimensional array consists of an array of one-dimensional arrays.
- For example, suppose **x = new int[2][2][5]**,
  - **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**.

Multidimensional Arrays (cont'd)

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

  double[][][] scores = **new double**[**6**][**5**][**2**];
- You can also use the short-hand notation to create and initialize the array as follows:

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



  - **scores[0][1][0]** refers to the multiple-choice score for the first student's second exam, which is **9.0**.
  - **scores[0][1][1]** refers to the essay score for the first student's second exam, which is **22.5**.

Chapter Summary

- A two-dimensional array can be used to store a table.
- A variable for two-dimensional arrays can be declared using the syntax: **elementType[][] arrayVar**.
- A two-dimensional array can be created using the syntax: **new elementType [ROW_SIZE][COLUMN_SIZE]**
- Each element in a two-dimensional array is represented using the syntax: **arrayVar[rowIndex][columnIndex]**.
- You can create and initialize a two-dimensional array using an array initializer with the syntax: **elementType[][] arrayVar = {{row values}, . . . , {row values}}**
- You can use arrays of arrays to form multidimensional arrays. For example, a variablefor three-dimensional arrays can be declared as **elementType[][][] arrayVar**, and a three-dimensional array can be created using **new elementType[size1][size2] [size3]**.

Ideas for further practice

- (*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

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