**CHAPTER 8: MULTIDIMENSIONAL ARRAYS**

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.

* 1. **TWO-DIMENSIONAL ARRAY BASICS**

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

Here am going to discuss

* How to declare a variable for two-dimensional arrays.
* How to create a twodimensional array.
* How to access elements in a two-dimensional array.
  + 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;

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];

* + 1. **OBTAINING THE LENGTH OF TWO-DIMENSIONAL ARRAYS**

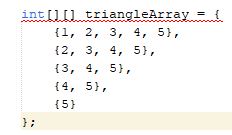
A two-dimensional array is actually an array in which each element is a one-dimensional array.

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**.

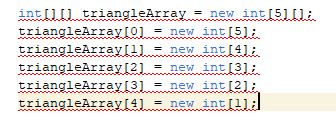
* + 1. **RAGGED ARRAYS**

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**.

Here is an example of creating a ragged array:

****

Or you can create a ragged array using the following syntax:



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

The syntax new **int[][]** would be wrong.

* 1. **PROCESSING TWO-DIMENSIONAL ARRAYS**

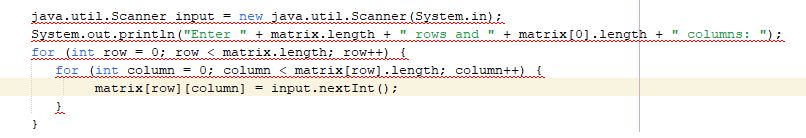
Nested for loops are often used to process a two-dimensional array.

Suppose an array matrix is created as follows:

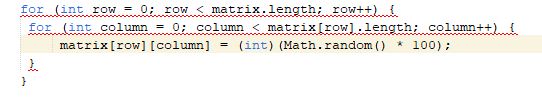
**int[][] matrix = new int[10][10];**

Some examples of processing two-dimensional arrays.

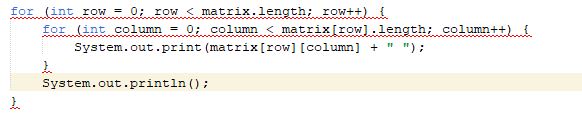
1. Initializing arrays with input values. The following loop initializes the array with user input values:



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



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

****

* 1. **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.

EXAMPLE:

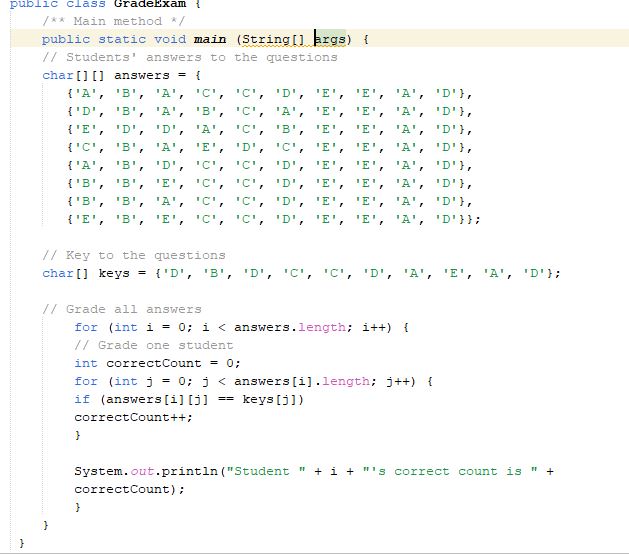
Suppose you need to write a program that grades multiple-choice tests. Assume there are eight students and ten questions, and the answers are stored in a two-dimensional array. Each row records a student’s answers to the questions, as shown in the following array:

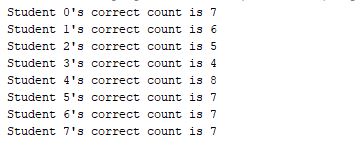
Students’ Answers to the Questions:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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 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.

Solution:

****

****

* 1. **MULTIDIMENSIONAL ARRAYS**

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

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.

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}}};

**THE END!**