

Lecture 61

Java – Arrays [2-D Array]

Mostafa Elhosseini

Professor at College of Computer Science and
Engineering

Taibah University

<https://youtube.com/drmelhosseini>

| Displaying Arrays

- You can use `println` to display an array, but it probably doesn't do what you would like. For example, say you print an array like this:
`int[] a = {1, 2, 3, 4};`
`System.out.println(a);`
- The output is something like this:
 - `[I@626b2d4a`
 - The **bracket** indicates that the **value is an array**, **I stands for integer**, and the **rest represents the address** of the array in memory
 - The **Java library** includes a class, **`java.util.Arrays`**, that provides **methods for working with arrays**. One of them, **`toString`**, returns a **string representation of an array**

| 2-D Array

- **Represent tables** of values with data arranged in **rows and columns**.
- To identify a particular table element, you specify **two indices**.
- By convention, the **first** identifies the element's **row** and the **second** its **column**.
- Java **does not support** multidimensional arrays **directly**, but it allows you to specify **one-dimensional arrays** whose **elements** are also **one-dimensional arrays**, thus achieving the same effect

| 2-D Array

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Diagram illustrating the structure of a 2-D array 'a'. The array is organized into rows and columns. The elements are identified by their row and column indices. The diagram shows the first three rows (Row 0, Row 1, Row 2) and the first three columns (Column 0, Column 1, Column 2). The elements are represented as a[row][column].

Labels and arrows pointing to the first row, first column element (a[0][0]):

- Column index (points to the second index, 0)
- Row index (points to the first index, 0)
- Array name (points to 'a')

- Every element in array `a` is identified by an array-access expression of the form `a[row][column]`;

| 2-D Array Initialization

- A **two-dimensional** array **b** with **two rows and two columns** could be declared and initialized with nested array initializers
`int[][] b = {{1, 2}, {3, 4}};`
- The **compiler** counts the number of **nested array initializers** (represented by sets of braces within the outer braces) to determine the number of rows in array **b**. The **compiler counts the initializer values in the nested array initializer** for a row to determine the number of columns in that row.

| 2-D Array Initialization

- **Multidimensional arrays are maintained as arrays of one-dimensional arrays**
- Therefore, **array b** in the preceding declaration is actually **composed of two separate one-dimensional arrays**—one containing the values in the **first** nested initializer list **{1, 2}** and one containing the values in the **second** nested initializer list **{3, 4}**. Thus, array **b** itself is an **array of two elements**, each a one-dimensional array of int values

| 2-D Array with Rows of Different Lengths

- The **manner** in which multidimensional arrays are represented makes them **quite flexible**.
- In fact, the lengths of the rows in array **b** are not required to be the same

```
int[][] b = {{1, 2}, {3, 4, 5}};
```
- **creates integer array b** with two elements (determined by the number of nested array initializers) that represent the rows of the two-dimensional array. **Each element of b** is a **reference to a one-dimensional array of int** variables. The int array for **row 0** is a one-dimensional array with two elements (**1 and 2**), and the int array for **row 1** is a one-dimensional array with three elements (**3, 4 and 5**).

| Creating 2-D Array

- A multidimensional array with the same number of columns in every row can be created with an array-creation expression
`int[][] b = new int[3][4];`
- **3 and 4** to specify the number of **rows and number of columns**, respectively, **but this is not required**.
- Programs can also **use variables to specify array dimensions**, because **new creates arrays at execution time—not at compile time**. The elements of a multidimensional array are initialized when the array object is created

| Creating 2-D Array

- A multidimensional array in which each **row** has a **different number of columns** can be created as follows:

```
int[][] b = new int[2][]; // create 2 rows  
b[0] = new int[5]; // create 5 columns for row 0  
b[1] = new int[3]; // create 3 columns for row 1
```

| Example - 01

- Initializing two-dimensional arrays with array initializers and using **nested for loops** to traverse the arrays to display it

| Example - 01

```
import java.util.Scanner;
import java.util.Arrays;
public class sekoseko {
    public static void main(String[] args) {
        //Scanner sc = new Scanner(System.in);
        int[][] c1 = {{1, 2, 3}, {4, 5, 6}};
        int[][] c2 = {{1, 2}, {3}, {4, 5, 6}};
        int i,j;
        for(i= 0;i<c2.length;i++)
        {
            for(j = 0;j < c2[i].length;j++)
                System.out.print(c2[i][j]);
            System.out.println();
        }
    }
}
```

| References

- Java: How to Program, 9th Edition (Deitel) 9th Edition by Paul Deitel , Harvey M. Deitel
- Head First Java, 2nd Edition 2nd Edition by Kathy Sierra, Bert Bates
- Think Java: How to Think Like a Computer Scientist 1st Edition by Allen B. Downey, Chris Mayfield