Unit III: Java as Object Oriented Programming Language- Overview

Fundamentals of JAVA, Arrays: one dimensional array, multi-dimensional array, alternative array declaration statements,

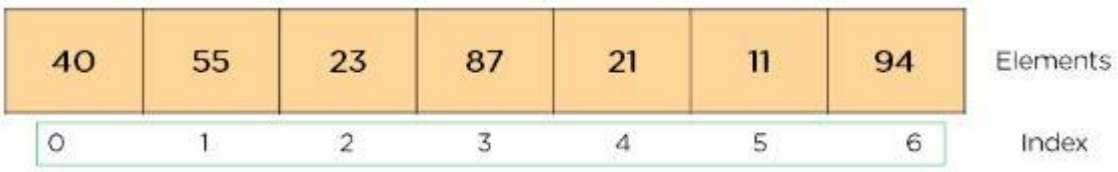
String Handling: String class methods,

Classes and Methods: class fundamentals, declaring objects, assigning object reference variables, adding methods to a class, returning a value, constructors, this keyword, garbage collection, finalize() method, overloading methods, argument passing, object as parameter, returning objects, access control, static, final, nested and inner classes, command line arguments, variable - length arguments.

Arrays are a straightforward yet essential concept of Java programming. Whether you are an experienced programmer or a beginner, you will inevitably use arrays in almost all aspects of Java programming.

What is an Array in Java?

An array refers to a data structure that contains homogeneous elements. This means that all the elements in the array are of the same data type. Let's take an example:



This is an array of seven elements. All the elements are integers and homogeneous. The green box below the array is called the index, which always starts from zero and goes up to n-1 elements. In this case, as there are seven elements, the index is from zero to six. There are three main features of an array:

1. Dynamic allocation: In arrays, the memory is created dynamically, which reduces the amount of storage required for the code.
2. Elements stored under a single name: All the elements are stored under one name. This name is used any time we use an array.
3. Occupies contiguous location: The elements in the arrays are stored at adjacent positions. This makes it easy for the user to find the locations of its elements.

Advantages of Arrays in Java

- Java arrays enable you to access any element randomly with the help of indexes
- It is easy to store and manipulate large data sets

Disadvantages of Arrays in Java

- The size of the array cannot be increased or decreased once it is declared—arrays have a fixed size
- Java cannot store heterogeneous data. It can only store a single type of primitives

Now that we understand what Java arrays are- let us look at how arrays in Java are declared and defined.

Define an Array in Java

Arrays in Java are easy to define and declare. First, we have to define the array. The syntax for it is:

```
type var-name[];
OR
type[] var-name;
```

Here, the type is int, String, double, or long. **var-name** is the variable name of the array.

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with **square brackets**:

```
String[] cars;
```

We have now declared a variable that holds an array of strings. To insert values to it, you can place the values in a comma-separated list, inside curly braces:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

To create an array of integers, you could write:

```
int[] myNum = {10, 20, 30, 40};
```

Access the Elements of an Array

You can access an array element by referring to the index number.
This statement accesses the value of the first element in cars:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars[0]);

// Outputs Volvo
```

Note: Array indexes start with 0: [0] is the first element. [1] is the second element, etc.

Change an Array Element

To change the value of a specific element, refer to the index number:

Example

```
cars[0] = "Opel";
```

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

cars[0] = "Opel";

System.out.println(cars[0]);

// Now outputs Opel instead of Volvo
```

Array Length

To find out how many elements an array has, use the length property:

Example

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};

System.out.println(cars.length);

// Outputs 4
```

```
// Java program to illustrate creating an array of integers, puts some values in the array,
// and prints each value to standard output.

class Demo {
        public static void main(String[] args)
        {
                // declares an Array of integers.
                int[] arr;

                // allocating memory for 5 integers.
                arr = new int[5];
                arr[0] = 10;
                arr[1] = 20;

                // so on...
                arr[2] = 30;
                arr[3] = 40;
                arr[4] = 50;

                // accessing the elements of the specified array
                for (int i = 0; i < arr.length; i++)
                        System.out.println("Element at index " + i
                                                        + " : " + arr[i]);
        }
}
```

Output

Element at index 0 : 10
Element at index 1 : 20
Element at index 2 : 30
Element at index 3 : 40
Element at index 4 : 50

Types of Array in java

There are two types of array:

1. One-dimensional array
2. Multi-dimensional array

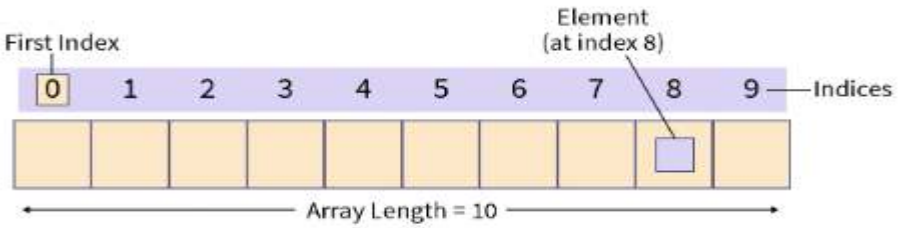
One-Dimensional Array in Java

It is clear from the name that a one-dimensional array in java must deal with only one parameter. Entities of similar types can be stored together using one-dimensional arrays. It can store primitive data types (int, float, char, etc.) or objects.

Creating a one-dimension array in Java

A one-dimensional array can be visualized as a single row or a column of array elements that are represented by a variable name and whose elements are accessed by index values.

For example, the score of a series of football matches can be stored in a one-dimensional array.



Declaration of one-dimensional array

So, let's see how we can declare a one-dimensional array in Java.

General form:

```
data-type var-name[];
    OR
data-type[] var-name;
    OR
data-type []var-name;
```

An array declaration has two components :

a) **data-type:** The data type determines the data type of each element present in the array-like char, int, float, objects etc.
b) **var-name:** It is the name of the reference variable which points to the array object stored in the heap memory.
c) **[ ] :** It is called subscript.

Example:

```
int Array[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];
```

The first statement just informs the compiler that this variable (Array) will hold an array of the integer type. To link Array with an actual, physical array of integers, we must allocate one using the **new** operator and assign it to Array.

Construction of one-dimensional array in Java

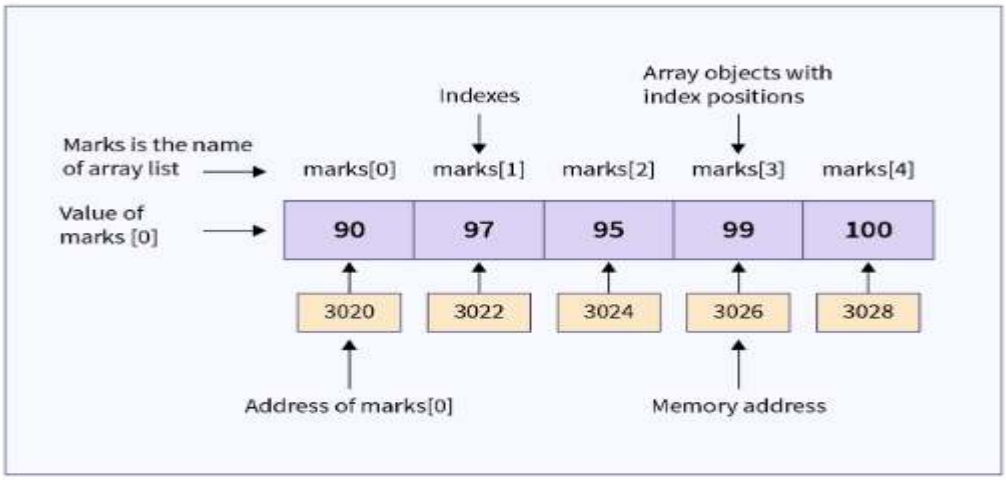
There are mainly two ways to create an array in java:

1. We can declare and store the values directly at the time of declaration :

```
int marks[ ] = { 90, 97, 95, 99, 100 };
```

Here JVM(**Java Virtual Machine** that drives the Java Code, converts Java bytecode into machine language) will assign five contiguous memory locations to store the values assigned to the array.

Memory address for array marks -



As shown in the diagram, JVM stores the elements in contiguous memory locations. Each element position can be easily accessed through the index number starting from 0 to n-1 where n is equal to the number of elements stored in the array.

2. The second way of creating an array is by first declaring the array and then allocating the memory through the **new** keyword :

```
var-name = new type[size];
```

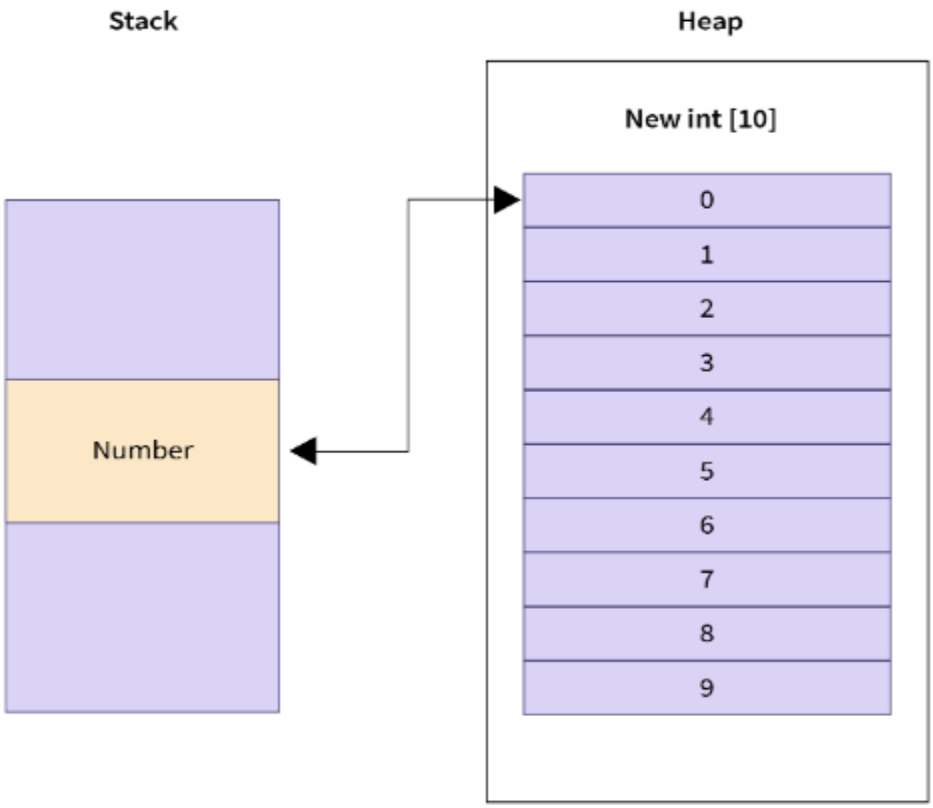
Here **size** determines the max number of elements that can be stored in the array, To allocate memory using **new** we must specify the type and number of elements in the array.

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

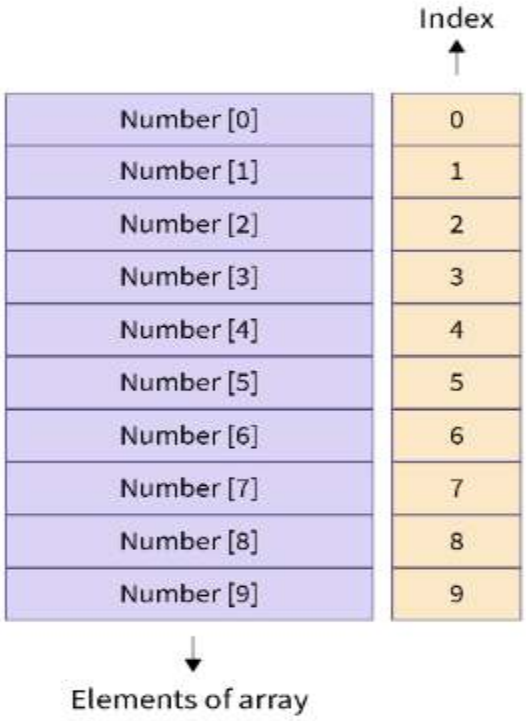
JVM has allocated memory locations to store 10 integers but we have not yet stored actual elements in the array. So, next, we'll look at how do elements get stored in memory.

Memory representation after construction

The **new int[10]** initializes and creates an object referenced as number and assigns memory to the object in heap segment.



By default, the new operator initializes the elements in the array to **zero**(for numeric types), **false**(for Boolean), or **null** (for reference types).



- Array in java is index-based, the first element of the array is stored at 0th position, the second element at 1st position, and so on.
- We can access each value of the array by using numbers with subscript ([]).
- Like we can access the first element through *number* [0], the second element through *number* [1], and similarly the last element through *number* [*length*−1].

Initialization of one-dimensional array

Let's code to add values to the one dimensional array:

```
public class AssignValues {

  public static void main(String args[]) {
    int number[]; // array declared
    number = new int[10]; // allocating memory, initialization
    number[0] = 11;
    number[1] = 22;
    number[2] = 33;
    number[3] = 44;
    number[4] = 55;
    number[5] = 66;
```

```
    number[6] = 77;
    number[7] = 88;
    number[8] = 99;
    number[9] = 100;
  }
}
```

Note:

- Since we chose int data type we can only add integer values to the array. So, the type of variable depends on the **data type** of an array.
- The size of the array depends upon how many values we are providing at the time of initialization.
- The size of one dimensional arrays cannot be changed once created.

Memory representation after initialization

One-Dimensional Array Examples

Example 1: standard method

The standard method includes declaring a variable and initializing it using **new** or with a specialized set of values using **array initializer**.

General Syntax:

```
data_type[] var_name = new data_type[];
            OR
data_type var_name[] = {}; // curly braces encloses specialized set of values
```

For Example:

```
public class Example {

 public static void main(String args[]) {
  int arr[] = { 1, 5, 10, 15, 20 }; // initializing array
  for (int i = 0; i < arr.length; i++) {
   System.out.println(arr[i] + " "); // printing array elements
  }
 }
}
```

Output :

```
1 5 10 15 20
```

In the above example, we created a one-dimensional array of type int and stored 5 elements 1, 5, 10, 15, 20 and set the arr variable to refer to the new array. Each element of the array is accessed through its index value.

Example 2: using the scanner

The Scanner class is used to get user input, and it is found in the java.util package. It was introduced to make it easy to read basic data types from a character input source.

To use the **Scanner class**, first we need to create a Scanner object. The constructor specifies the source(Reader, InputStream, String or File) of the characters that the Scanner will read in our program. The scanner acts as a wrapper for the input source.

To take integer input from the user:

```
Scanner standardInputScanner = new Scanner(System.in);

// reads the next token from the input source and tries to convert it to a value of type int.
int input = scanner.nextInt();
```

Code:

```
import java.util.Scanner;
public class OneDimensionalArrayInput {
  public static void main(String args[]) {
    // creating object of Scanner class
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter length of Array: ");
    int arrLength = scan.nextInt();
    int[] anArray = new int[arrLength];
    System.out.println("Enter the elements of the Array");
    for (int i = 0; i < arrLength; i++) {
      // taking array input
      anArray[i] = scan.nextInt();
    }
    System.out.println("One dimensional array elements are:");
    for (int i = 0; i < arrLength; i++) {
      // printing array elements
      System.out.print(anArray[i] + " ");
    }
  }
}
```

Output:

```
Enter length of Array:
5
Enter the elements of the Array
1
2
3
4
5
One dimensional array elements are:
1 2 3 4 5
```

Explanation:

Here we have used **Scanner class** to take input from the user and the nextInt() method of the Scanner class to take integer input. It created an array of sizes 5 with elements 1, 2, 3, 4,5.

Example 3: Using String

A **String** is a sequence of characters. It is an immutable object, which means its value can not be changed. A String Array is an Array of a fixed number of String values. String arrays are declared, stored and operated in the same way we handle integer arrays.

Syntax:

String[] var_name = {};

Create String Array:

```
public class StringArray {
  public static void main(String args[]) {
    String[] array = { "Learning", "One-dimensional array", "in java" };
    System.out.println("String array elements are:");
    for (int i = 0; i < array.length; i++) {
              System.out.println(array[i]);
    }
  }
}
```

Output:

String array elements are:
Learning
One-dimensional array
in java

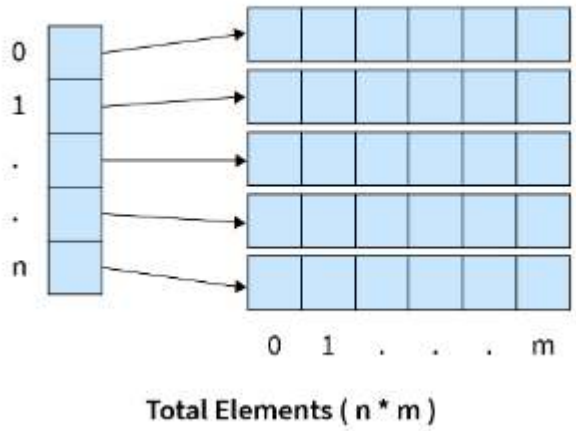
Benefits Of One-Dimensional Array

- Arrays help to maintain enormous data under a single variable name
- They help in accessing elements easily using the index number.
- No chance of extra memory being allocated, which avoids memory overflow or shortage of memory in arrays.
- We can retrieve or sort the data easily using one-dimensional arrays.

Multidimensional Arrays in Java

- An array is a data structure to store a collection of values of the same type.
- The data can be accessed using its integer index.
- Since all the data is stored in contiguous memory locations, an array is always initialized with its size, and it cannot be changed while the program is running.
- Multidimensional arrays, like a 2D array is a bunch of 1D arrays put together in an array.
- A 3D array is like a bunch of 2D arrays put together in a 1D array, and so on.
- Data in multi-dimensional arrays are stored in row-major format, i.e., tabular form. To access array elements in multidimensional arrays, more than one index is used.

Note: Java doesn't have multi-dimensional arrays. They are nested arrays.



Syntax of Multidimensional Array in Java

The general syntax to declare a multi-dimensional array

DataType[1st Dimension][2nd Dimension]....[Nth Dimension] arrayName;

General syntax to initialize the array:

arrayName = new DataTye[length 1][length 2]....[length N];

Or

DataType[1st Dimension][2nd Dimension]....[Nth Dimension] arrayName = new DataTye[length 1][length 2]....[length N];

Here, DataType is the type of data to be stored in the array. The array can be 1 dimensional to N-dimensional. arrayName is the variable name given to the array and length is the size of the array of respective dimensions.

Types of Multidimensional Array in Java

Multidimensional array can be a 2D array, a 3D array, a 4D array, where D stands for Dimension. But the higher the dimension, the more difficult it is to access and store elements.

1. Two-Dimensional Array or 2D Array.

Many computer games, like **simulation games**, **strategy games**, or **first-person conflict** games, involve objects that occupy a two-dimensional space. Applications for such positional games use a two-dimensional array for representing objects using indices, say i and j, which refer to the cells in the array. The first index refers to a row number, and the second index refers to a column number. In Java, a two-dimensional array is nothing but a table or matrix with columns and rows. It is the simplest multi-dimensional array.

a. Indirect Method of Declaration

The indirect method is the method where the size of the array is declared, and array values are filled later. The two square brackets ([] []) can be written either after DataType or after arrayName.

Syntax to initialize and declare a multidimensional array.

A two-dimensional array can be declared as follows:

DataType [][] arrayName; (OR) DataType arrayName[][];

Initialization:
arrayName = new DataType[length 1][length 2];

Java is a statically-typed programming language, specifying **DataType** decides the type of element it will accept. arrayName is the variable name given to the array which will be used while accessing, manipulating, and storing data. length 1 is the number of rows in a two-dimensional array, and length 2 is the number of columns in it.

Here, initially, all the array cells are filled with some base or garbage values.

Note: Size of a two-dimensional array is $length1 * length2$, i.e. number of elements in an array is $length1 * length2$.

Example:

```
public class Main {
   public static void main(String[] args) {
      // 2D array with 3 rows and 4 columns
      int[][] twoD_array = new int[3][4];
      // visiting each row
      for (int row = 0; row < twoD_array.length; row++) {
         // each column for a given row
         for (int col = 0; col < twoD_array[row].length; col++) {
            // storing row number + column number
            twoD_array[row][col] = row + col;
         }
      }
      // printing resultant array
      for (int[] temp : twoD_array) {
         for (int val : temp) {
            System.out.print(val + " ");
         }
         System.out.println();
      }
   }
}
```

Output:

0 1 2 3
1 2 3 4
2 3 4 5

Note that indexing in java starts with 0.

b. Direct Method of Declaration

The direct method is where values to the array are assigned during declaration only.
Syntax to initialize and declare a multidimensional array:

```
DataType[][] arrayName = {
                    {element 1, element 2,....,element n},  //1st row
                    .
                    .
                    {element 1, element 2,...., element n}  //mth row
                  };
```

Note: The two square brackets ([] []) can be written either after DataType or after arrayName while declaring the two-dimensional array.

Example:

```
public class Main {
   public static void main(String[] args){
      int[][] twoD_array = {
                           {1, 3, 4},
                           {9, 8, 6},
                           {2, 0, 5}
                          };
      for(int[] temp : twoD_array){
         for(int val : temp){
            System.out.print(val+" ");
         }
         System.out.println();
      }
   }
}
```

Output:

1 3 4
9 8 6
2 0 5

In the above example, we initialize the element values directly. Here, we don't have to write the size of the array.

There are various approaches to initializing the array:

- In the first example, we initialized the array using for loops.
- In the second example, we directly initialized the values only. Here compiler automatically calculates the rows and columns of the array.
- We can also assign values at various positions separately. This method is difficult to use when there is a large array.

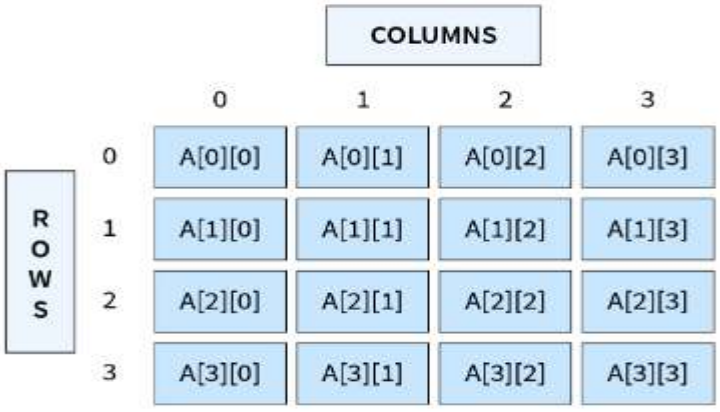
For Example:

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      // 2D array to store characters
      char[][] CharacterArray = new char[2][4];
      // First row stores JAVA in capital letters
      CharacterArray[0][0] = 'J';
      CharacterArray[0][1] = 'A';
      CharacterArray[0][2] = 'V';
      CharacterArray[0][3] = 'A';
      //second-row stores java in small letters
      CharacterArray[1][0] = 'j';
      CharacterArray[1][1] = 'a';
      CharacterArray[1][2] = 'v';
      CharacterArray[1][3] = 'a';
      System.out.println(
            "Printing 2D array using Arrays.deepToString() method: ");
      // deepToString() method converts the 2D array to string as printed in the output
      System.out.println(Arrays.deepToString(CharacterArray));
   }
}
```

Output:

Printing 2D array using Arrays.deepToString() method:
[[J, A, V, A], [j, a, v, a]]

Representation of 2D array



Example: Addition of two matrices

To add two arrays, we create 2 arrays, matrixA and matrixB and initialize them. A third array, matrixC is the array that we obtain by the addition of matrixA and matrixB.

Note: For addition, all the three matrices should have the same number of rows and columns.

```
public class Main {
   public static void main(String[] args){
      int n = 3;
      int m = 3;
      int[][] matrixA = {
                        {1,6,8},
                        {5,8,0},
                        {2,5,9}
                       };
      int[][] matrixB = {
                        {7,9,2},
                        {1,0,6},
                        {4,3,5}
                       };
      int[][] matrixC = new int[n][m];
      for(int i=0;i<n;i++){
         for(int j=0;j<m;j++){
            matrixC[i][j] = matrixA[i][j]+matrixB[i][j];
         }
      }
      for(int i=0;i<n;i++){
         for(int j=0;j<m;j++){
            System.out.print(matrixC[i][j]+" ");
         }
         System.out.println();
      }
   }
}
```

Output:

8 15 10
6 8 6
6 8 14

Three-Dimensional Array or 3D Array:

- An array is a complex form of a multidimensional array.
- A three-dimensional array adds an extra dimension and exponentially increases the size of a two-dimensional array.
- Effectively it is an array containing multiple two-dimensional arrays, with each element addressed by three indices, the first one for a two-dimensional array` and the rest two for the rows and the columns.

a. Indirect Method of Declaration

Syntax to declare a three-dimensional array:

DataType [][][] arrayName;

Initialization:

arrayName = new DataType [length 1][length 2][length 3];

- It can be of int, String, char, float, etc. DataTypes. Number of elements in a three-dimensional array is "length 1 * length 2 * length 3".
- And memory usage is equal to "number of elements * size of datatype". For instance, the size of int datatype in Java is 4 bytes.
- Thus, memory usage is 4 * (number of elements in an array) bytes.
- In this type of initialization, java fills all the cells of the array with some base or garbage values. For int array, it fills them with zero (0).

Example:

```
int[][][] threeDimensional = new int[2][2][2];
threeDimensional[0][1][1] = 708;
```

Number of elements in this array is 8.

b. Direct Method of Declaration

```
DataType[][][] arrayName = {
                    {
                        {element A1R1C1, element A1R1C2, ....},
                        {element A1R2C1, element A1R2C2, ....}
                    },
```

```
                {
                        {element A2R1C1, element A2R1C2, ....},
                        {element A2R2C1, element A2R2C2, ....}
                }
        };
```

c. Accessing Elements of Three-Dimensional Arrays

- Elements in three-dimensional arrays can be accessed as

  arrayName[arrayIndex][rowIndex][columnIndex];

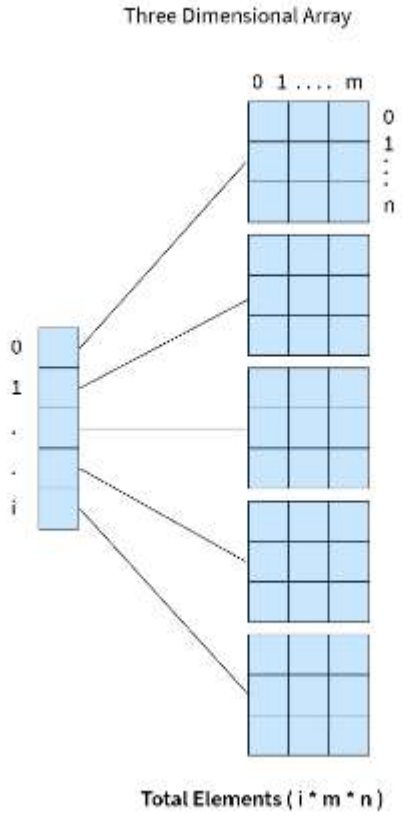
- This is similar to accessing a two-dimensional array. Here, arrayIndex is the outer array index, and rowIndex and columnIndex are rows and columns of inner arrays, respectively.
- Printing elements of three-dimensional Arrays Three-Dimensional arrays can also be printed using for loops, for-each loops, and individually.
- Simplest way to print elements is directly accessing elements
  System.out.print(arrayName[arrayIndex][rowIndex][columnIndex]);
- It can also be accessed using for loop:

```
for (int i = 0; i < arrayName.length; i++) {
   for (int j = 0; j < arrayName[i].length; j++) {
      for (int k = 0; k < arrayName[i][j].length; k++) {
         System.out.print(arrayName[i][j] + " ");
      }
   }
}
```

- Enhanced for loop or for-each loop can also be used to access elements.

```
for(int[][] twoD : arrayName){
   for(int[] oneD : twoD){
      for(int val : oneD){
         System.out.print(val+" ");
      }
   }
}
```

d. Representation of 3D array



e. Example of Three-Dimensional Array

- In the given example, we have data of 3 students, and their scores of 4 subjects in two semesters.
- As we can see, all three parameters are interrelated and can be stored in a single three-dimensional array.
- The outermost array can store the details student-wise. The inner array is nothing but a 2D array of semesters and subject scores.
- Each student has a unique semester and subject table. This unique table is then stored in our three-dimensional array.
- Similarly, the data can be accessed using for loops.
- The outermost loop will access the student number, the first inner loop will access the semester number, and the innermost loop will print the scores subject-wise.

```
public class Main {
   public static void main(String[] args) {
      int totalStudents = 3;
      int subjects = 4;
      int totalSemesters = 2;
      int[][] Student1 = {
            { 87, 79, 91, 88 },
            { 92, 80, 86, 91 }
      };       // Data of Student 1
      int[][] Student2 = {
            { 80, 75, 66, 74 },
            { 81, 89, 70, 83 }
      };      // Data of Student 2
      int[][] Student3 = {
            { 98, 88, 94, 95 },
            { 99, 93, 96, 97 }
      };      // Data of Student 3

      // array for storing data of all three students
      int[][][] studentArray = new int[totalStudents][totalSemesters][subjects];
      // storing data of students into 3D array
      studentArray[0] = Student1;
      studentArray[1] = Student2;
      studentArray[2] = Student3;
      // an elem of the student Array is a 2D array containing information of each student
      for (int i = 0; i < totalStudents; i++) {
         System.out.println("Student " + (i + 1) + ":-"); // printing student number
         for (int j = 0; j < totalSemesters; j++) {
            // for each student printing all semesters and their scores in it
            System.out.println("Semester " + (j + 1) + ":");
            for (int k = 0; k < subjects; k++) {
               System.out.print(studentArray[i][j][k] + " ");
               // i = student number, j = semester, k = subject number
            }
            System.out.println();
         }
         System.out.println();
      }
   }
}
```

Output:

```
Student 1:-
Semester 1:
87 79 91 88
Semester 2:
92 80 86 91

Student 2:-
Semester 1:
80 75 66 74
Semester 2:
81 89 70 83

Student 3:-
Semester 1:
98 88 94 95
Semester 2:
99 93 96 97
```

Conclusion

- Arrays are homogenous data structures that store the same type of data in them.
- Arrays can be single dimensional or multidimensional.
- Multidimensional arrays are a little complicated, and complexity increases with an increase in dimensions.
- Multidimensional arrays can be two-dimensional, three-dimensional, etc.
- Two-dimensional array can be visualized as an array of 1D arrays. Likewise, a Three-dimensional array can be considered an array of 2D arrays.
- Elements can be added to the array in two ways, directly and indirectly.
- They can be accessed in multiple ways, like using loops or using indexes.
- Two-dimension and Three-dimension arrays have many applications in graphics, games, software, mathematical analysis, etc.