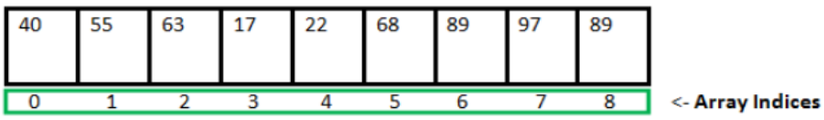
MODULE 9

ARRAYS IN JAVA

An array is a collection (sequence) of a fixed number of variables called elements or components, wherein all the elements are of the same data type.



Array Length = 9
First Index = 0
Last Index = 8

Creating, initializing, and accessing an Array **One-Dimensional Arrays:**

The general form of a one-dimensional array declaration is:

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

An array declaration has two components: the type and the name. ***type*** declares the element type of the array. The element type determines the data type of each element that comprises the array. Like an array of integers, we can also create an array of other primitive data types like char, float, double, etc.,

Example:

```
// both are valid declarations
int intArray[];
or int[] intArray;

byte byteArray[];
short shortsArray[];
boolean booleanArray[];
long longArray[];
```

float floatArray[];

double doubleArray[];

char charArray[];

Although the first declaration establishes that int Array is an array variable, **no actual array exists**. It merely tells the compiler that this variable (int Array) will hold an array of the integer type. To link int Array with an actual, physical array of integers, you must allocate one using **new** and assign it to int Array.

Instantiating an Array in Java

When an array is declared, only a reference of an array is created. To create or give memory to the array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:
var-name = new type [size];

Here, *type* specifies the type of data being allocated, *size* determines the number of elements in the array, and *var-name* is the name of the array variable that is linked to the array. To use *new* to allocate an array, **you must specify the type and number of elements to allocate.**

Example:
int intArray[]; //declaring array

intArray = new int[20]; // allocating memory to array

OR

int[] intArray = new int[20]; // combining both statements in one

The elements in the array allocated by new will automatically be initialized to ***zero*** *(for numeric types),* ***false*** *(for boolean), or* ***null*** *(for reference types).*

Obtaining an array is a two-step process. First, you must declare a variable of the desired array type. Second, you must allocate the memory to hold the array, using new, and assign it to the array variable. Thus, ***in Java****,* ***all arrays are dynamically allocated.***

Accessing Java Array Elements using for Loop

Each element in the array is accessed via its index. The index begins with 0 and ends at (total array size)-1. All the elements of array can be accessed using Java for Loop.

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

EXAMPLE PROGRAM

```
public class GFG {
    public static void main(String[] args)
    {
        // declares an Array of integers.
        int[] arr = new int[5];

        // initialize the first elements of the array
        arr[0] = 10;

        // initialize the second elements of the array
        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
```

Two Dimensional Array in Java

The Two Dimensional Array in Java programming language is nothing but an Array of Arrays. In Java Two Dimensional Array, data stored in row and columns, and we can access the record using both the row index and column index (like an Excel File).

Two Dimensional Array Declaration in Java

The following code snippet shows the two dimensional array declaration in Java Programming Language:

Data_Type[][] Array_Name;

- Data_type: It decides the type of elements it will accept. For example, If we want to store integer values, then the Data Type will be declared as int. If we want to store Float values, then the Data Type will be float.
- Array_Name: This is the name to give it to this Java two dimensional array. For example, Car, students, age, marks, department, employees, etc.

Similarly, one can declare the remaining type of two-dimensional arrays:

```
int [][] anIntegerArray; // declaring an two dimensional array of Integers
byte[][] anByteArray; // declaring an two dimensional array of Bytes
short[][] anShortArray; // declaring an two dimensional array of Shorts
long[][] anLongArray; // declaring an two dimensional array of Longs
float[][] anFloatArray; // declaring an two dimensional array of Floats
double[][] anDoubleArray; // declaring an two dimensional array of Doubles
boolean[][] anBooleanArray; // declaring an two dimensional array of Booleans
```

char[][] anCharArray; // declaring an two dimensional array of Chars
String[][] anStringArray; // declaring an two dimensional array of Strings

Create Two dimensional Array in Java

In order to create a two dimensional array in Java, we have to use the New operator as we shown below:

Data_Type[][] Array_Name = new int[Row_Size][Column_Size];

- Row_Size: Number of Row elements an array can store. For example, Row_Size = 5, then the array will have five rows.
- Column_Size: Number of Column elements an array can store. For example, Column_Size = 6, then the array will have 6 Columns.

EXAMPLE

```
public class twoDimensional {
    public static void main(String args[])
    {
        // declaring and initializing 2D array
        int arr[][]= { { 2, 7, 9 }, { 3, 6, 1 }, { 7, 4, 2 } };

        // printing 2D array
        for (int row = 0; row < 3; row++) {
            for (int column = 0; column < 3; column++)
                System.out.print(arr[row][column] + " ");

            System.out.println();
        }
    }
}
```

Three-Dimensional Arrays In Java

Three-dimensional arrays are complex for multi-dimensional arrays. A three dimensional can be defined as an array of two-dimensional arrays.

The general definition of a Three-dimensional array is given below:
data_type [] [] [] array_name = new data_type [d1][d2][d3];

Here,
d1, d2, d3 = sizes of the dimensions
data_type = data type of the elements of the array
array_name = name of the 3d array

Example of 3d array definition is:
int [] [] [] intArray = new int[2][3][4];

The above definition of 3d array can be interpreted as having 2 tables or arrays, 3 rows and 4 columns that totals up to 2x3x4 = 24 elements.

This means that in a 3d array, the three dimensions are interpreted as:

- **The number of Tables/Arrays:** The first dimension indicates how many tables or arrays a 3d array will have.
- **The number of Rows:** The second dimension signifies the total number of rows an array will have.
- **The number of Columns:** The third dimension indicates the total columns in the 3d array.

Initialize 3d Array
The approaches used to initialize a 3d array are the same as the ones used for initializing Two-dimensional arrays.

You can either initialize the array by assigning values to individual array elements or initialize the array during the declaration.

The example below shows the initialization of the 3d array while declaration.

```
public class Main
{
    public static void main(String[] args) {

        //initialize 3-d array
        int[][][] intArray = { { { 1, 2, 3}, { 4, 5, 6 } ,  { 7, 8, 9 } } };
        System.out.println ("3-d array is given below :");
        //print the elements of array
        for (int i = 0; i < 1; i++)
            for (int j = 0; j < 3; j++)
            for (int z = 0; z < 3; z++)
                System.out.println ("intArray [" + i
            + "][" + j + "][" + z + "] = " + intArray [i][j][z]);
    }
}
```

Output:

```
3-
d array is given below:

intArray [0][0][0] = 1

intArray [0][0][1] = 2

intArray [0][0][2] = 3

intArray [0][1][0] = 4

intArray [0][1][1] = 5

intArray [0][1][2] = 6

intArray [0][2][0] = 7

intArray [0][2][1] = 8

intArray [0][2][2] = 9
```

After initializing the 3d array during declaration, we have accessed the individual elements of the array and printed them.

Acces And Print 3d Array

Again, printing and accessing array elements in a three-dimensional array is similar to that in two-dimensional arrays.

The program below uses for loops to access the array elements and print them to the console.

```
public class Main
{
    public static void main(String[] args) {
        //initialize 3-d array
        int[][][] myArray = { { { 1, 2, 3 }, { 4, 5, 6 } },  { { 1, 4, 9 }, { 16, 25, 36 } },
                { { 1, 8, 27 }, { 64, 125, 216 } } };
        System.out.println("3x2x3 array is given below:");
        //print the 3-d array
        for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 3; k++) {
        System.out.print(myArray[i][j][k] + "\t");
        }
        System.out.println();
                    }
        System.out.println();
        }
    }
}
```

Output:

```
3x2x3 array is given below:

1        2        3

4        5        6


1        4        9

16       25       36


1        8        27

64       125      216
```

The above program displays a tabular representation of a three-dimensional array. As shown, it is a 3x2x3 array which means that it has 3 tables, 2 rows and 3 columns and thus 18 elements.

Implementation of Arrays Class in Java

binarySearch (Array, fromIndex, toIndex, Key, Comparator) Method

Prototype:

```
static int binarySearch (int[] a, int key)
```

Parameters:

- a: The array to be searched
- key: The value to be searched for

Return Value:

- int: index of the search key, if it is contained in the array; otherwise, **(-(insertion point) – 1)**. The insertion point is defined as the point at which the key would be inserted into the array: the index of the first element greater than the key, or a.length if all elements in the array are less than the specified key. Note that this guarantees that the return value will be **>= 0** if and only if the key is found.

Code

```
import java.util.Arrays;

public class Scaler {

	public static void main(String[] args)
	{
		int Arr[] = { 10, 20, 11, 21, 31 };

		Arrays.sort(Arr);

		int Key = 31;

		System.out.println(
			Key + " found at index = "
			+ Arrays.binarySearch(Arr, Key));
	}
}
```

Output

```
31 found at index = 4
```

Explanation Here we sort the input array as we need a sorted array for our binarySearch method. Then we pass the array and the key to the binarySearch method. the index at which we find the key is displayed.

Note:

- If the input list is not sorted, the results are undefined.
- If there are duplicates, there is no guarantee which one will be found

equals (array1, array2) Method

Prototype:

```
static boolean equals (int [] a, int [] a2)
```

Parameters:

- a1: the first array that will be tested for equality
- a2: the second array that will be tested for equality

Return Value:

- Returns true only if both arrays are equal. Two arrays are considered equal if both arrays contain the same number of elements, and all **corresponding pairs** of elements in the two arrays are equal.

Code

```
import java.util.Arrays;

public class Scaler {
   public static void main(String[] args)
   {

      int Arr[] = { 10, 20, 11, 21, 31 };

      int Arr1[] = { 10, 11, 21 };

      System.out.println("Integer Arrays on comparison are : " + Arrays.equals(Arr, Arr1));
   }
}
```

Output

```
Integer Arrays on comparison are: false
```

Explanation Here we use the equals method and we use two arrays and call the equals method to check if they are equal or not. Since they aren't equal we get false as our result.

fill (Original Array, fillValue) Method

Prototype:

```
static void fill(int[] a, int val)
```

Parameters:

- a: array that will be filled.
- val: value that will be filled in all places of the array.

Return Value:

- None

Code

```
import java.util.Arrays;

public class Scaler {
   public static void main(String[] args)
   {
      int Arr[] = { 10, 20, 11, 21, 31 };

      int Key = 23;

      Arrays.fill(Arr, Key);
```

```
        System.out.println("Integer Array on filling is: "
                           + Arrays.toString(Arr));
    }
}
```

Output

```
Integer Array on filling is: [23, 23, 23, 23, 23]
```

Explanation This example shows the basic usage of the 'fill' method. We fill our entire array with a key-value and here the value is 23. Hence the output received is an array with all the values as 23.

sort (Original Array) Method

Prototype:

```
static void sort(int[] a)
```

Parameters: a: array that is to be sorted.

Return Value:

- None

Code

```
import java.util.Arrays;

public class Scaler {
    public static void main(String[] args)
    {
        int Arr[] = { 10, 20, 11, 21, 31 };

        Arrays.sort(Arr);

        System.out.println("Integer Array is: " + Arrays.toString(Arr));
    }
}
```

Output

```
Integer Array is: [10, 11, 20, 21, 31]
```

Explanation Here we use an array and call the sort method to arrange the elements of the array in ascending order i.e. sorting the array. Hence we get the sorted array as our output.

Benefits of Arrays Class in Java

- Arrays class makes it easier to perform common operations on an array.
- No need to use complicated loops to work with an array.
- No need to reinvent the wheel for operations such as binary search and sorting.