BCA – 401: Java Programming

Rahul Kumar Singh

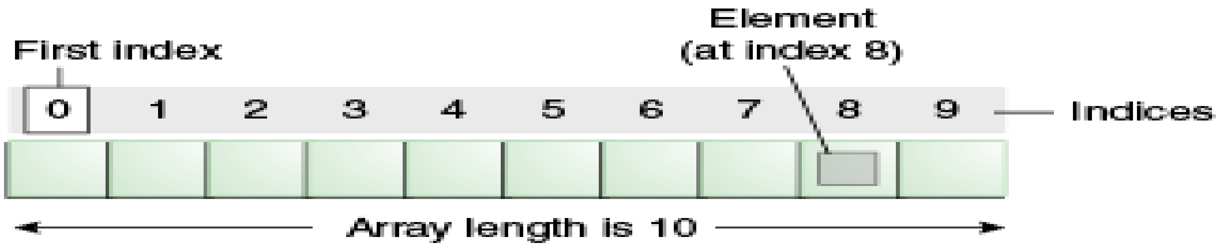
In today's Class we have discussed on Arrays in Java.

Arrays in Java:-

Normally, an array is a collection of similar type of elements which has contiguous memory location.

Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.



Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

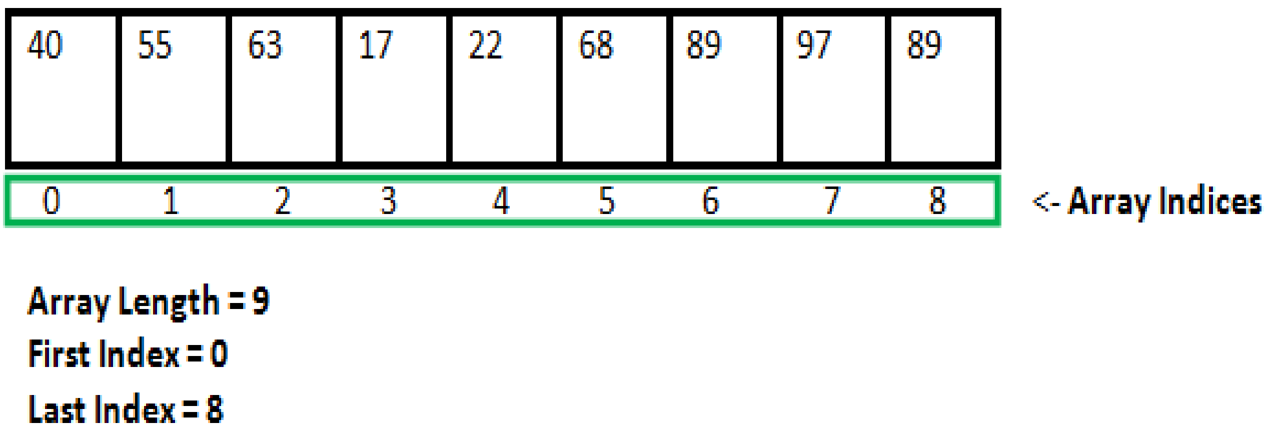
In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. Serialization is a mechanism of converting the state of an object into a byte stream. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.

Following are some important points about Java arrays:-

- In Java, all arrays are dynamically allocated.
- Arrays are stored in contiguous memory [consecutive memory locations].
- Since arrays are objects in Java, we can find their length using the object property length. This is different from C/C++, where we find length using sizeof.
- A Java array variable can also be declared like other variables with [] after the data type.
- The variables in the array are ordered, and each has an index beginning with 0.
- Java array can also be used as a static field, a local variable, or a method parameter.
- The size of an array must be specified by int or short value and not long.
- The direct superclass of an array type is Object.
- Every array type implements the interfaces Cloneable and java.io.Serializable.

- This storage of arrays helps us randomly access the elements of an array [Support Random Access].
- The size of the array cannot be altered(once initialized). However, an array reference can be made to point to another array.
- An array can contain primitives (int, char, etc.) and object (or non-primitive) references of a class depending on the definition of the array. In the case of primitive data types, the actual values are stored in contiguous memory locations. In the case of class objects, the actual objects are stored in a heap segment.



Advantages of array:-

Code Optimization: It makes the code optimized, we can retrieve or sort the data efficiently.

Random access: We can get any data located at an index position.

Disadvantages of array:-

Size Limit: We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java:-

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java:-

Declaring Array Variables:-

To use an array in a program, we must declare a variable to reference the array, and we must specify the type of array the variable can reference.

Syntax:- Here is the syntax for declaring an array variable –

```
dataType[] arrayRefVar;   // preferred way.
```

or

```
dataType arrayRefVar[];  // works but not preferred way.
```

Note – The style **dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]** comes from the C/C++

language and was adopted in Java to accommodate C/C++ programmers.

Example:-

```
double[] myList;   // preferred way.
```

or

```
double myList[];   // works but not preferred way.
```

Instantiation of an Array in Java:-

Syntax:-

```
arrayRefVar=new datatype[size];
```

Example of Java Array

In the following example we are going to declare, instantiate, initialize and traverse an array.

//Java Program to illustrate how to declare, instantiate, initialize and traverse the Java array.

```
class Testarray

{

public static void main(String args[])

{

int a[]=new int[5];      //declaration and instantiation
a[0]=10;            //initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)   //length is the property of array
System.out.println(a[i]);
}
}
```

Output:-

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array together.

```
class Testarray
{
public static void main(String args[])
{
int a[]={33,3,4,5};//declaration, instantiation and initialization
//printing array
for(int i=0;i<a.length;i++)      //length is the property of array
System.out.println(a[i]);
}
}
```

Output:-

```
33
3
4
5
```

For-each Loop for Java Array:-

We can also print the Java array using for-each loop. The Java for-each loop prints the array elements one by one. It holds an array element in a variable, then executes the body of the loop.

The syntax of the for-each loop is given below:

```
for(data_type variable:array)
{
```

```
//body of the loop
}
```

Example:-

```
//Java Program to print the array elements using for-each loop
class Testarray
{
public static void main(String args[])
{
int arr[]={33,3,4,5};
//printing array using for-each loop
for(int i:arr)
System.out.println(i);
}
}
```

Output:

```
33
3
4
5
```

Passing Array to a Method in Java:-

We can pass the java array to method so that we can reuse the same logic on any array.

Example:-

In this example we get the minimum number of an array using a method.

//Java Program to demonstrate the way of passing an array to method.

```
class Testarray
{
//creating a method which receives an array as a parameter
static void min(int arr[])
{
int min=arr[0];
for(int i=1;i<arr.length;i++)
 if(min>arr[i])
  min=arr[i];
System.out.println(min);
}
public static void main(String args[])
{
```

```
int a[]={33,3,4,5};        //declaring and initializing an array
min(a);          //passing array to method
}
}
```

Output:-

```
3
```

Anonymous Array in Java:-

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

//Java Program to demonstrate the way of passing an anonymous array to method.

```
public class TestAnonymousArray
{
//creating a method which receives an array as a parameter
static void printArray(int arr[])
{
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}
public static void main(String args[])
{
//passing anonymous array to method
printArray(new int[]{10,22,44,66});
}
}
```

Output:-

```
10
22
44
66
```

Returning Array from the Method:-

We can also return an array from the method in Java.

Example:-

//Java Program to return an array from the method

```
class TestReturnArray
{
//creating method which returns an array
static int[] get()
{
return new int[]{10,30,50,90,60};
}
public static void main(String args[])
{
//calling method which returns an array
int arr[]=get();
//printing the values of an array
for(int i=0;i<arr.length;i++)
System.out.println(arr[i]);
}
}
```

Output:

```
10
30
50
90
60
```

}

ArrayIndexOutOfBoundsException:-

The Java Virtual Machine (JVM) throws an ArrayIndexOutOfBoundsException if length of the array in negative, equal to the array size or greater than the array size while traversing the array.

Example:-

//Java Program to demonstrate the case of ArrayIndexOutOfBoundsException in a Java Array.

```
public class TestArrayException
{
public static void main(String args[])
{
int arr[]={50,60,70,80};
for(int i=0;i<=arr.length;i++)
{
System.out.println(arr[i]);
}
}
}
```

Output:-

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4 at TestArrayException.main(TestArrayException.java:5)
    50
    60
    70
    80
```