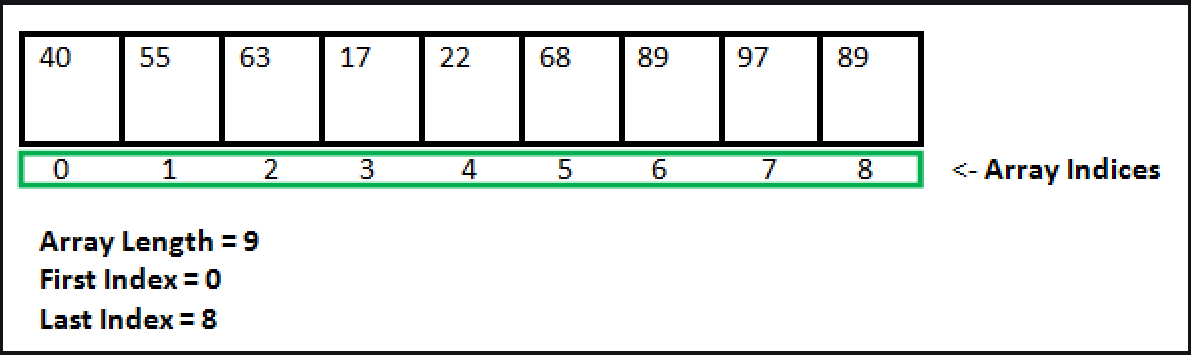
Arrays in Java

Array in java is a group of like-typed variables referred to by a common name. Arrays in Java work differently than they do in C/C++. Following are some important points about Java arrays.

- In Java, all arrays are dynamically allocated. (discussed below)
- 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.



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., or user-defined data types (objects of a class). Thus, the element type for the array determines what type of data the array will hold.

Example:

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

byte byteArray[];
short shortsArray[];
boolean booleanArray[];
long longArray[];
float floatArray[];
double doubleArray[];
char charArray[];

// an array of references to objects of
// the class MyClass (a class created by
// user)
MyClass myClassArray[];

Object[]  ao,        // array of Object
Collection[] ca;  // array of Collection
                     // of unknown type
```

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
```

Note:

*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). Do refer to default array values in Java.*

*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.***

Array Literal

In a situation where the size of the array and variables of the array are already known, array literals can be used.

```
 int[] intArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 // Declaring array literal
```

- The length of this array determines the length of the created array.
- There is no need to write the new int[] part in the latest versions of Java.

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]);
```

Implementation:

- Java

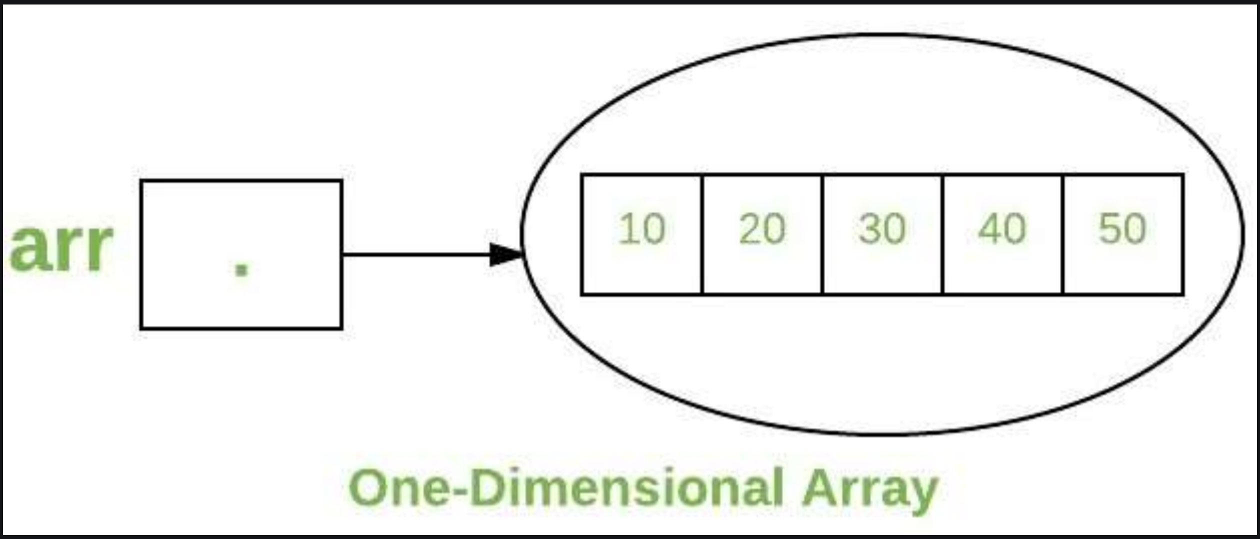
```java
// Java program to illustrate creating an array

// of integers,  puts some values in the array,

// and prints each value to standard output.


class GFG {

    public static void main(String[] args)

    {

        // declares an Array of integers.

        int[] arr;


        // allocating memory for 5 integers.

        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
```

Time Complexity: O(n)
Auxiliary Space: O(1)

You can also access java arrays using for each loops.



Arrays of Objects

An array of objects is created like an array of primitive-type data items in the following way.

```
Student[] arr = new Student[5]; //student is a user-defined class
```

Syntax:

1) data type[] arrName;
2) datatype arrName[];

3) datatype [] arrName;

- Java

```java
import java.io.*;


class GFG {

    public static void main (String[] args) {


      int [] arr=new int [4];

      // 4 is the size of arr

    }

}
```

Output

The student Array contains five memory spaces each of the size of student class in which the address of five Student objects can be stored. The Student objects have to be instantiated using the constructor of the Student class, and their references should be assigned to the array elements in the following way.

Example

- Java

```java
// Java program to illustrate creating

//  an array of objects


class Student {

    public int roll_no;

    public String name;

    Student(int roll_no, String name)

    {

        this.roll_no = roll_no;

        this.name = name;

    }

}


// Elements of the array are objects of a class Student.
```

```java
public class GFG {

    public static void main(String[] args)

    {

        // declares an Array of integers.

        Student[] arr;



        // allocating memory for 5 objects of type Student.

        arr = new Student[5];



        // initialize the first elements of the array

        arr[0] = new Student(1, "aman");



        // initialize the second elements of the array

        arr[1] = new Student(2, "vaibhav");



        // so on...

        arr[2] = new Student(3, "shikar");

        arr[3] = new Student(4, "dharmesh");

        arr[4] = new Student(5, "mohit");



        // accessing the elements of the specified array

        for (int i = 0; i < arr.length; i++)

            System.out.println("Element at " + i + " : "

                               + arr[i].roll_no + " "

                               + arr[i].name);

    }

}
```

Output

```
Element at 0 : 1 aman

Element at 1 : 2 vaibhav

Element at 2 : 3 shikar

Element at 3 : 4 dharmesh

Element at 4 : 5 mohit
```

Time Complexity: O(n)
Auxiliary Space : O(1)

Example

An array of objects is also created like :

- Java

```java
// Java program to illustrate creating

//  an array of objects


class Student

{


    public String name;

    Student(String name)

    {

        this.name = name;

    }

      @Override

    public String toString(){

        return name;

    }

}


// Elements of the array are objects of a class Student.

public class GFG

{

    public static void main (String[] args)

    {

           // declares an Array and initializing  the elements of the array

        Student[] myStudents = new Student[]{new Student("Dharma"),new Student("sanvi"),new Student


        // accessing the elements of the specified array

        for(Student m:myStudents){

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

Output

```
Dharma
sanvi
Rupa
Ajay
```

What happens if we try to access elements outside the array size?

JVM throws **ArrayIndexOutOfBoundsException** to indicate that the array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of an array.

Below code shows what happens if we try to access elements outside the array size.

- Java

```java
// Code for showing error "ArrayIndexOutOfBoundsException"


public class GFG {
    public static void main(String[] args)
    {
        int[] arr = new int[4];
        arr[0] = 10;
        arr[1] = 20;
        arr[2] = 30;
        arr[3] = 40;

        System.out.println(
            "Trying to access element outside the size of array");
        System.out.println(arr[5]);
    }
}
```

Output

```
Trying to access element outside the size of array
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5
```

```
out of bounds for length 4
at GFG.main(GFG.java:13)
```

- Java

```java
public class GFG {

    public static void main(String[] args)

    {

        int[] arr = new int[2];

        arr[0] = 10;

        arr[1] = 20;


        for (int i = 0; i < arr.length; i++)

            System.out.println(arr[i]);

    }

}
```

Output

```
10

20
```

Time Complexity: O(n),here n is size of array.
Auxiliary Space: O(1), since no extra space required.

Multidimensional Arrays:

Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other arrays. These are also known as Jagged Arrays. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

Syntax :

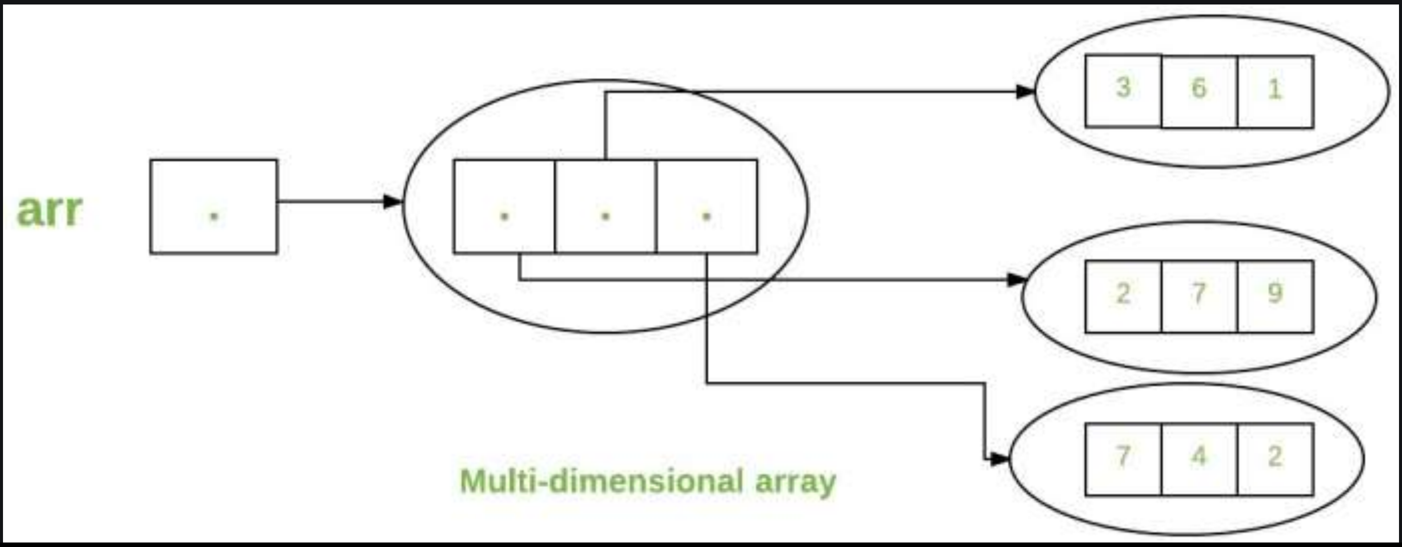
```
datatype [][] arrayrefvariable;
or
datatype arrayrefvariable[][];
```

Example:

- Java

```java
import java.io.*;


class GFG {

    public static void main (String[] args) {

      // Syntax

       int [][] arr= new int[3][3];
      // 3 row and 3 column

    }

}
```

Output



```
int[][] intArray = new int[10][20]; //a 2D array or matrix
int[][][] intArray = new int[10][20][10]; //a 3D array
```

Example

- Java

```
public class multiDimensional {

    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 i = 0; i < 3; i++) {

            for (int j = 0; j < 3; j++)

                System.out.print(arr[i][j] + " ");


            System.out.println();

        }
    }
}
```

Output

```
2 7 9
3 6 1
7 4 2
```

Passing Arrays to Methods

Like variables, we can also pass arrays to methods. For example, the below program passes the array to method *sum* to calculate the sum of the array's values.

- Java

```java
// Java program to demonstrate
// passing of array to method

public class Test {
    // Driver method
    public static void main(String args[])
    {
        int arr[] = { 3, 1, 2, 5, 4 };

        // passing array to method m1
        sum(arr);
    }

    public static void sum(int[] arr)
    {
        // getting sum of array values
        int sum = 0;

        for (int i = 0; i < arr.length; i++)
            sum += arr[i];

        System.out.println("sum of array values : " + sum);
    }
}
```

Output

```
sum of array values : 15
```

Time Complexity: *O(n)*
Auxiliary Space : *O(1)*

Returning Arrays from Methods

As usual, a method can also return an array. For example, the below program returns an array from method *m1*.

- Java

```java
// Java program to demonstrate

// return of array from method


class Test {

    // Driver method

    public static void main(String args[])

    {

        int arr[] = m1();


        for (int i = 0; i < arr.length; i++)

            System.out.print(arr[i] + " ");

    }


    public static int[] m1()

    {

        // returning  array

        return new int[] { 1, 2, 3 };

    }

}
```

Output

```
1 2 3
```

Time Complexity: O(n)
Auxiliary Space : O(1)

Class Objects for Arrays

Every array has an associated Class object, shared with all other arrays with the same component type.

- Java

```java
// Java program to demonstrate

// Class Objects for Arrays


class Test {

    public static void main(String args[])

    {

        int intArray[] = new int[3];

        byte byteArray[] = new byte[3];

        short shortsArray[] = new short[3];


        // array of Strings

        String[] strArray = new String[3];


        System.out.println(intArray.getClass());

        System.out.println(

            intArray.getClass().getSuperclass());

        System.out.println(byteArray.getClass());

        System.out.println(shortsArray.getClass());

        System.out.println(strArray.getClass());

    }

}
```

Output

```
class [I

class java.lang.Object

class [B

class [S

class [Ljava.lang.String;
```

Explanation:

1. The string “[I” is the run-time type signature for the class object “array with component type *int*.”

2. The only direct superclass of an array type is java.lang.Object.
3. The string “[B” is the run-time type signature for the class object “array with component type *byte*.”
4. The string “[S” is the run-time type signature for the class object “array with component type *short*.”
5. The string “[L” is the run-time type signature for the class object “array with component type of a Class.” The Class name is then followed.

Array Members

Now, as you know that arrays are objects of a class, and a direct superclass of arrays is a class Object. The members of an array type are all of the following:

- The public final field *length* contains the number of components of the array. Length may be positive or zero.
- All the members are inherited from class Object; the only method of Object that is not inherited is its clone method.
- The public method *clone()* overrides the clone method in class Object and throws no checked exceptions.

Arrays Types and Their Allowed Element Types

Array Types	**Allowed Element Types**
Primitive Type Arrays	Any type which can be implicitly promoted to declared type.
Object Type Arrays	Either declared type objects or it’s child class objects.
Abstract Class Type Arrays	Its child-class objects are allowed.
Interface Type Arrays	Its implementation class objects are allowed.

Cloning of Arrays

When you clone a single-dimensional array, such as Object[], a “deep copy” is performed with the new array containing copies of the original array’s elements as opposed to references.

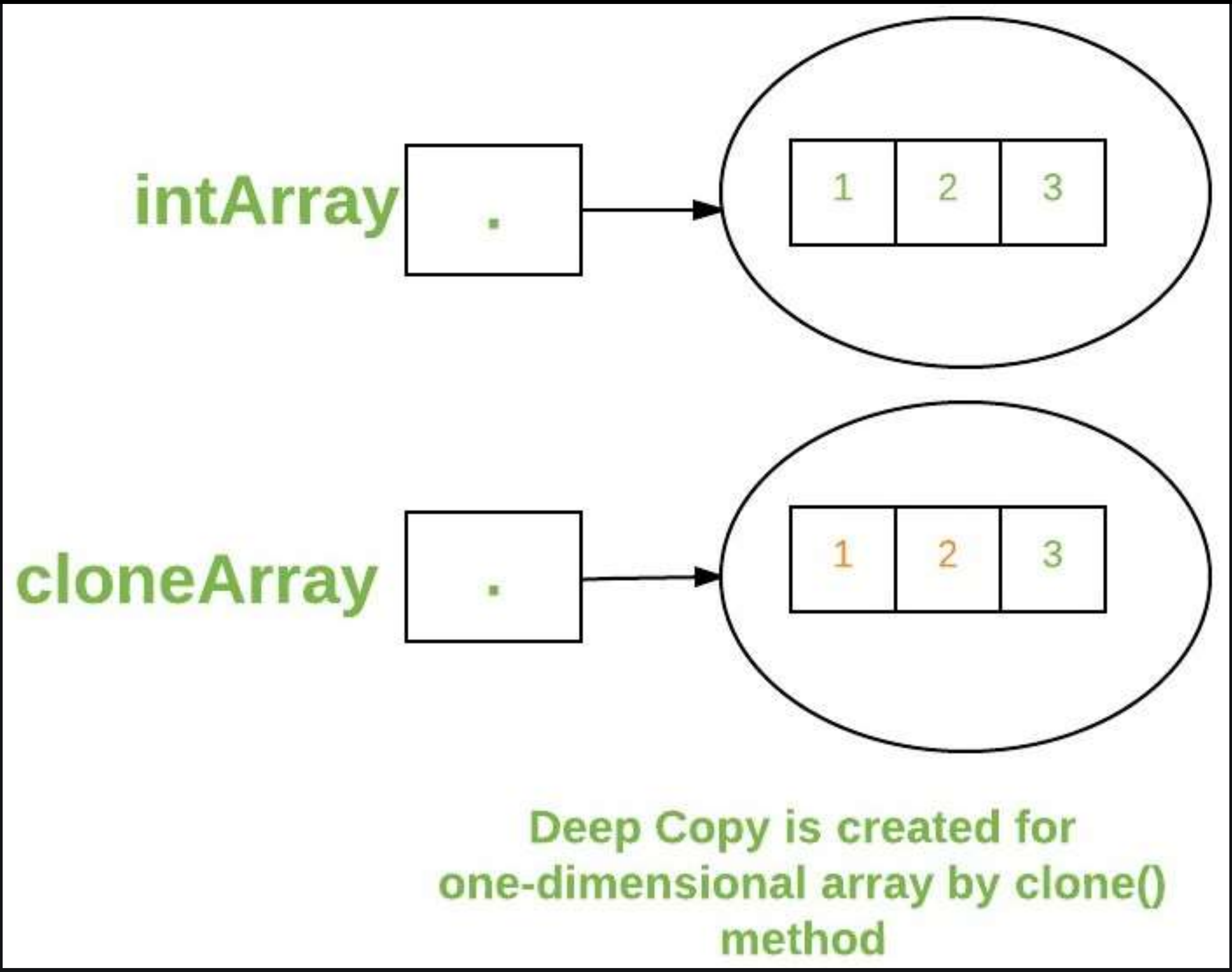
- Java

```
// Java program to demonstrate

// cloning of one-dimensional arrays
```

```java
class Test {

    public static void main(String args[])

    {

        int intArray[] = { 1, 2, 3 };


        int cloneArray[] = intArray.clone();


        // will print false as deep copy is created

        // for one-dimensional array

        System.out.println(intArray == cloneArray);


        for (int i = 0; i < cloneArray.length; i++) {

            System.out.print(cloneArray[i] + " ");

        }

    }

}
```

Output

```
false

1 2 3
```


intArray
.
1
2
3
cloneArray
.
1
2
3
Deep Copy is created for one-dimensional array by clone() method

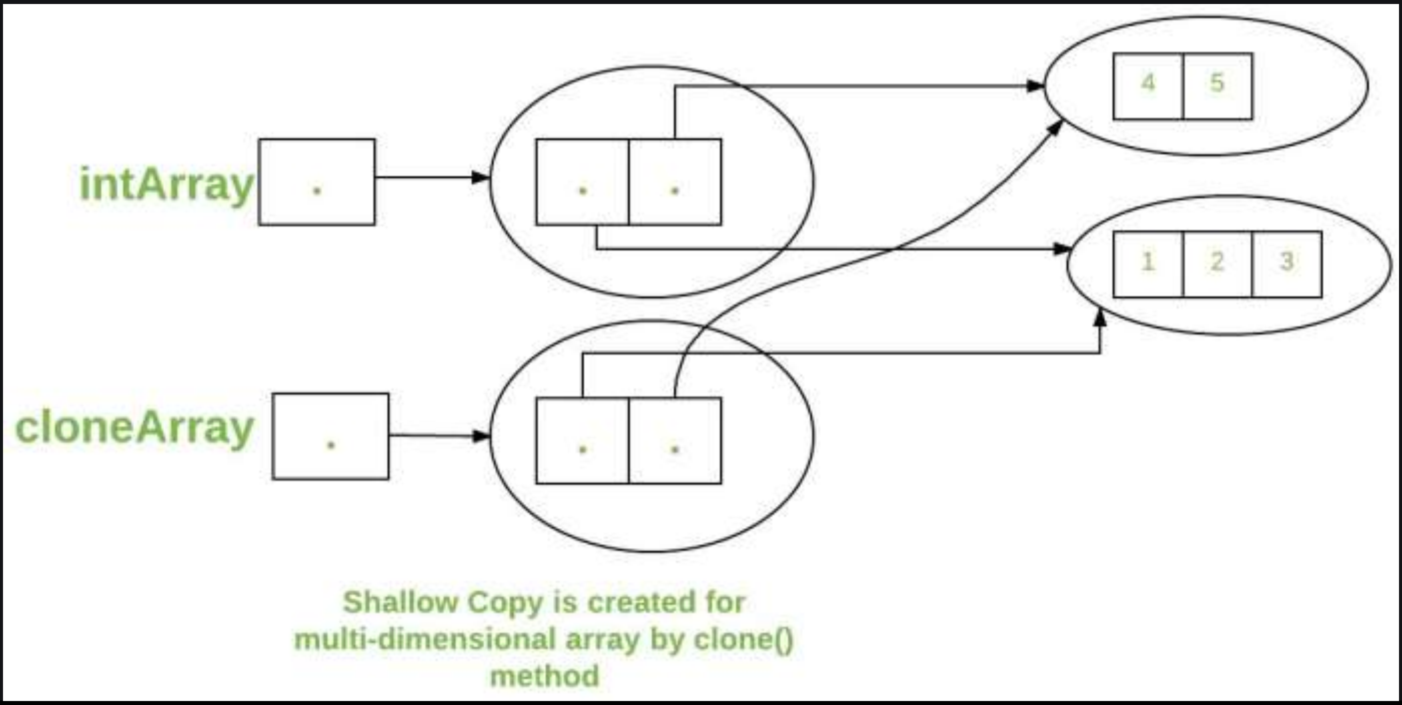
A clone of a multi-dimensional array (like Object[][]) is a “shallow copy,” however, which is to say that it creates only a single new array with each element array a reference to an original element array, but subarrays are shared.

- Java

```
// Java program to demonstrate

// cloning of multi-dimensional arrays


class Test {

    public static void main(String args[])

    {

        int intArray[][] = { { 1, 2, 3 }, { 4, 5 } };


        int cloneArray[][] = intArray.clone();


        // will print false
        System.out.println(intArray == cloneArray);


        // will print true as shallow copy is created

        // i.e. sub-arrays are shared

        System.out.println(intArray[0] == cloneArray[0]);

        System.out.println(intArray[1] == cloneArray[1]);

    }

}
```

Output

```
false

true

true
```


intArray
cloneArray
4
5
1
2
3
Shallow Copy is created for multi-dimensional array by clone() method

Arrays class in Java

The **Arrays** class in **java.util package** is a part of the **Java Collection Framework**. This class provides static methods to dynamically create and access **Java arrays**. It consists of only static methods and the methods of Object class. The methods of this class can be used by the class name itself.
The class hierarchy is as follows:

```
java.lang.Object

 ? java.util.Arrays
```

Geek, now you must be wondering why do we need java Arrays class when we are able to declare, initialize and compute operations over arrays. The answer to this though lies within the methods of this class which we are going to discuss further as practically these functions help programmers expanding horizons with arrays for instance there are often times when loops are used to do some tasks on an array like:

- Fill an array with a particular value.
- Sort an Arrays.
- Search in an Arrays.
- And many more.

Here Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops, hence forth making our code super short and optimized.

Syntax: Class declaration

```
public class Arrays

    extends Object
```

Syntax: In order to use Arrays

```
Arrays.<function name>;
```

Methods in Java Array Class

The Arrays class of the java.util package contains several static methods that can be used to fill, sort, search, etc in arrays. Now let us discuss the methods of this class which are shown below in a tabular format as follows:

Methods	Action Performed
asList()	Returns a fixed-size list backed by the specified Arrays
binarySearch()	Searches for the specified element in the array with the help of the Binary Search Algorithm
binarySearch(array, fromIndex, toIndex, key, Comparator)	Searches a range of the specified array for the specified object using the Binary Search Algorithm
compare(array 1, array 2)	Compares two arrays passed as parameters lexicographically.
copyOf(originalArray, newLength)	Copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.
copyOfRange(originalArray, fromIndex, endIndex)	Copies the specified range of the specified array into a new Arrays.
deepEquals(Object[] a1, Object[] a2)	Returns true if the two specified arrays are deeply equal to one another.

Methods	Action Performed
deepHashCode(Object[] a)	Returns a hash code based on the “deep contents” of the specified Arrays.
deepToString(Object[] a)	Returns a string representation of the “deep contents” of the specified Arrays.
equals(array1, array2)	Checks if both the arrays are equal or not.
fill(originalArray, fillValue)	Assigns this fill value to each index of this arrays.
hashCode(originalArray)	Returns an integer hashCode of this array instance.
mismatch(array1, array2)	Finds and returns the index of the first unmatched element between the two specified arrays.
parallelPrefix(originalArray, fromIndex, endIndex, functionalOperator)	Performs parallelPrefix for the given range of the array with the specified functional operator.
parallelPrefix(originalArray, operator)	Performs parallelPrefix for complete array with the specified functional operator.
parallelSetAll(originalArray, functionalGenerator)	Sets all the elements of this array in parallel, using the provided generator function.
parallelSort(originalArray)	Sorts the specified array using parallel sort.
setAll(originalArray, functionalGenerator)	Sets all the elements of the specified array using the generator function provided.
sort(originalArray)	Sorts the complete array in ascending order.
sort(originalArray, fromIndex, endIndex)	Sorts the specified range of array in ascending order.
sort(T[] a, int fromIndex, int toIndex, Comparator< super T> c)	Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.
sort(T[] a, Comparator< super T> c)	Sorts the specified array of objects according to the order induced by the specified comparator.
spliterator(originalArray)	Returns a Spliterator covering all of the specified Arrays.

Methods	Action Performed
spliterator(originalArray, fromIndex, endIndex)	Returns a Spliterator of the type of the array covering the specified range of the specified arrays.
stream(originalArray)	Returns a sequential stream with the specified array as its source.
toString(originalArray)	It returns a string representation of the contents of this array. The string representation consists of a list of the array's elements, enclosed in square brackets ("[]"). Adjacent elements are separated by the characters a comma followed by a space. Elements are converted to strings as by String.valueOf() function.

Implementation:

Example 1: asList() Method

- Java

```
// Java Program to Demonstrate Arrays Class
// Via asList() method

// Importing Arrays utility class
// from java.util package
import java.util.Arrays;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To convert the elements as List
        System.out.println("Integer Array as List: "
                           + Arrays.asList(intArr));
    }
}
```

Output

```
Integer Array as List: [[I@2f4d3709]
```

Example 2: binarySearch() Method

This methods search for the specified element in the array with the help of the binary search algorithm.

- Java

```java
// Java Program to Demonstrate Arrays Class
// Via binarySearch() method

// Importing Arrays utility class
// from java.util package
import java.util.Arrays;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        Arrays.sort(intArr);

        int intKey = 22;

        // Print the key and corresponding index
        System.out.println(
            intKey + " found at index = "
            + Arrays.binarySearch(intArr, intKey));
    }
}
```

Output

```
22 found at index = 3
```

Example 3: binarySearch(array, fromIndex, toIndex, key, Comparator) Method
This method searches a range of the specified array for the specified object using the binary search algorithm.

- Java

```java
// Java program to demonstrate
// Arrays.binarySearch() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        Arrays.sort(intArr);
        int intKey = 22;

        System.out.println(
            intKey
            + " found at index = "
            + Arrays
                  .binarySearch(intArr, 1, 3, intKey));
    }
}
```

Output

```
22 found at index = -4
```

Example 4: compare(array 1, array 2) Method

- Java

```java
// Java program to demonstrate
// Arrays.compare() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Array
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.compare(intArr, intArr1));
    }
}
```

Output

```
Integer Arrays on comparison: 1
```

Example 5: compareUnsigned(array 1, array 2) Method

- Java

```java
// Java program to demonstrate
// Arrays.compareUnsigned() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.compareUnsigned(intArr, intArr1));
    }
}
```

Output

```
Integer Arrays on comparison: 1
```

Example 6: copyOf(originalArray, newLength) Method

- Java

```
// Java program to demonstrate
// Arrays.copyOf() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));

        System.out.println("\nNew Arrays by copyOf:\n");

        System.out.println("Integer Array: "
                           + Arrays.toString(
                                 Arrays.copyOf(intArr, 10)));
    }
}
```

Output

```
Integer Array: [10, 20, 15, 22, 35]


New Arrays by copyOf:

```

```
Integer Array: [10, 20, 15, 22, 35, 0, 0, 0, 0, 0]
```

Example 7: copyOfRange(originalArray, fromIndex, endIndex) Method

- Java

```java
// Java program to demonstrate
// Arrays.copyOfRange() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));

        System.out.println("\nNew Arrays by copyOfRange:\n");

        // To copy the array into an array of new length
        System.out.println("Integer Array: "
                           + Arrays.toString(
                                 Arrays.copyOfRange(intArr, 1, 3)));
    }
}
```

Output

```
Integer Array: [10, 20, 15, 22, 35]


New Arrays by copyOfRange:


Integer Array: [20, 15]
```

Example 8: deepEquals(Object[] a1, Object[] a2) Method

- Java

```java
// Java program to demonstrate
// Arrays.deepEquals() method

import java.util.Arrays;

public class Main {
    public static void main(String[] args)
    {
        // Get the Arrays
        int intArr[][] = { { 10, 20, 15, 22, 35 } };

        // Get the second Arrays
        int intArr1[][] = { { 10, 15, 22 } };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.deepEquals(intArr, intArr1));
    }
}
```

Output

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

Example 9: deepHashCode(Object[] a) Method

- Java

```
// Java program to demonstrate
// Arrays.deepHashCode() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[][] = { { 10, 20, 15, 22, 35 } };

        // To get the dep hashCode of the arrays
        System.out.println("Integer Array: "
                           + Arrays.deepHashCode(intArr));
    }
}
```

Output

```
Integer Array: 38475344
```

Example 10: deepToString(Object[] a) Method

- Java

```
// Java program to demonstrate
// Arrays.deepToString() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[][] = { { 10, 20, 15, 22, 35 } };

        // To get the deep String of the arrays
        System.out.println("Integer Array: "
                           + Arrays.deepToString(intArr));
    }
}
```

Output

```
Integer Array: [[10, 20, 15, 22, 35]]
```

Example 11: equals(array1, array2) Method

- Java

```
// Java program to demonstrate
// Arrays.equals() method

import java.util.Arrays;

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("Integer Arrays on comparison: "
                           + Arrays.equals(intArr, intArr1));
    }
}
```

Output

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

Example 12: fill(originalArray, fillValue) Method

- Java

```
// Java program to demonstrate
// Arrays.fill() method

import java.util.Arrays;

public class Main {
```

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        int intKey = 22;

        Arrays.fill(intArr, intKey);

        // To fill the arrays
        System.out.println("Integer Array on filling: "
                           + Arrays.toString(intArr));
    }
}
```

Output

```
Integer Array on filling: [22, 22, 22, 22, 22]
```

Example 13: hashCode(originalArray) Method

- Java

```java
// Java program to demonstrate
// Arrays.hashCode() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To get the hashCode of the arrays
        System.out.println("Integer Array: "
                           + Arrays.hashCode(intArr));
    }
}
```

Output

```
Integer Array: 38475313
```

Example 14: mismatch(array1, array2) Method

- Java

```java
// Java program to demonstrate
// Arrays.mismatch() method

import java.util.Arrays;
```

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

        // Get the Arrays
        int intArr[] = { 10, 20, 15, 22, 35 };

        // Get the second Arrays
        int intArr1[] = { 10, 15, 22 };

        // To compare both arrays
        System.out.println("The element mismatched at index: "
                           + Arrays.mismatch(intArr, intArr1));
    }
}
```

Output

```
The element mismatched at index: 1
```

Example 15: [parallelSort(originalArray)] Method

- Java

```java
// Java program to demonstrate
// Arrays.parallelSort() method

// Importing Arrays class from
// java.util package
import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using parallelSort
        Arrays.parallelSort(intArr);

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

Output

```
Integer Array: [10, 15, 20, 22, 35]
```

Example 16: [sort(originalArray)] Method

- Java

```java
// Java program to demonstrate
// Arrays.sort() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort-
        Arrays.sort(intArr);

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

Output

```
Integer Array: [10, 15, 20, 22, 35]
```

Example 17: sort(originalArray, fromIndex, endIndex) Method

- Java

```java
// Java program to demonstrate
// Arrays.sort() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        Arrays.sort(intArr, 1, 3);

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

Output

```
Integer Array: [10, 15, 20, 22, 35]
```

Example 18: sort(T[] a, int fromIndex, int toIndex, Comparator< super T> c) Method

- Java

```java
// Java program to demonstrate working of Comparator
// interface
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a student.
class Student {
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name,
                   String address)
    {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Used to print student details in main()
    public String toString()
    {
        return this.rollno + " "
            + this.name + " "
            + this.address;
    }
}

class Sortbyroll implements Comparator<Student> {
    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}

// Driver class
class Main {
    public static void main(String[] args)
    {
        Student[] arr = { new Student(111, "bbbb", "london"),
                          new Student(131, "aaaa", "nyc"),
                          new Student(121, "cccc", "jaipur") };

        System.out.println("Unsorted");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr, 1, 2, new Sortbyroll());
        System.out.println("\nSorted by rollno");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Output

```
Unsorted
111 bbbb london
131 aaaa nyc
121 cccc jaipur


Sorted by rollno
111 bbbb london
131 aaaa nyc
121 cccc jaipur
```

Example 19: sort(T[] a, Comparator< super T> c) Method

- Java

```java
// Java program to demonstrate working of Comparator
// interface
import java.util.*;
import java.lang.*;
import java.io.*;

// A class to represent a student.
class Student {
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name,
                   String address)
    {
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Used to print student details in main()
    public String toString()
    {
        return this.rollno + " "
            + this.name + " "
            + this.address;
    }
}

class Sortbyroll implements Comparator<Student> {

    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}

// Driver class
class Main {
    public static void main(String[] args)
    {
        Student[] arr = { new Student(111, "bbbb", "london"),
                          new Student(131, "aaaa", "nyc"),
                          new Student(121, "cccc", "jaipur") };

        System.out.println("Unsorted");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);

        Arrays.sort(arr, new Sortbyroll());

        System.out.println("\nSorted by rollno");
        for (int i = 0; i < arr.length; i++)
            System.out.println(arr[i]);
    }
}
```

Output

```
Unsorted
111 bbbb london
131 aaaa nyc
121 cccc jaipur


Sorted by rollno
111 bbbb london
121 cccc jaipur
131 aaaa nyc
```

Example 20: spliterator(originalArray) Method

- Java

```java
// Java program to demonstrate
// Arrays.spliterator() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr));
    }
}
```

Output

```
Integer Array: java.util.Spliterators$IntArraySpliterator@4e50df2e
```

Example 21: spliterator(originalArray, fromIndex, endIndex) Method

- Java

```java
// Java program to demonstrate
// Arrays.spliterator() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To sort the array using normal sort
        System.out.println("Integer Array: "
                           + Arrays.spliterator(intArr, 1, 3));
    }
}
```

Output

```
Integer Array: java.util.Spliterators$IntArraySpliterator@4e50df2e
```

Example 22: stream(originalArray) Method

- Java

```java
// Java program to demonstrate
// Arrays.stream() method
```

```java
import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To get the Stream from the array
        System.out.println("Integer Array: "
                           + Arrays.stream(intArr));
    }
}
```

Output

```
Integer Array: java.util.stream.IntPipeline$Head@7291c18f
```

Example 23: toString(originalArray) Method

- Java

```java
// Java program to demonstrate
// Arrays.toString() method

import java.util.Arrays;

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

        // Get the Array
        int intArr[] = { 10, 20, 15, 22, 35 };

        // To print the elements in one line
        System.out.println("Integer Array: "
                           + Arrays.toString(intArr));
    }
}
```

Output

```
Integer Array: [10, 20, 15, 22, 35]
```