**Arrays**

**Java Arrays:**

An array is a collection of similar type of elements which have a 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.
* So we can represent the formula for Array index as ***0 to (n-1).***
* we can get the length of the array using the ***length*** member.
* We can store primitive values or objects in an array in Java.
* we can also create single dimensional or multidimensional arrays in Java.
* Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.
* A Java array variable can also be declared like other variables with **[]** after the data type.
* Java array can be also be used as a static field, a local variable or a method parameter.
* The **size** of an array must be specified by an int value and not long or short.
* The direct superclass of an array type is [Object](https://www.geeksforgeeks.org/object-class-in-java/).
* Every array type implements the interfaces [***Cloneable***](https://www.geeksforgeeks.org/marker-interface-java/) and [***java.io.Serializable***](https://www.geeksforgeeks.org/serialization-in-java/)***.***

Array can contains primitives data types as well as objects of a class depending on the definition of array. In case of primitives data types, the actual values are stored in contiguous memory locations. In case of objects of a class, [the actual objects are stored in heap segment](https://www.geeksforgeeks.org/g-fact-46/).

In Java, when we only declare a variable of a class type, only a reference is created (memory is not allocated for the object). To allocate memory to an object, we must use new(). So the object is always allocated memory on heap.



**Advantages:**

* ***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:**

* ***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:**

* Single Dimensional Array
* Multidimensional Array

**Single Dimensional Array:**

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 array of int type, we can also create an array of other primitive data types like char, float, double..etc or user defined data type(objects of a class).Thus, the element type for the array determines what type of data the array will hold.

**Syntax to declare an array:**

type var-name[];

OR

type[] var-name;

**Example:**

int intArray[];

or int[] intArray;

// 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

**Instantiating an Array in Java:**

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:The general form of new as it applies to one-dimensional arrays appears as follows:

**Syntax to instantiate an array:**

var-name = new type [size];

Here, type specifies the type of data being allocated, size specifies the number of elements in the array, and var-name is the name of array variable that is linked to the array. That is, 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).
* 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 that will hold the array, using new, and assign it to the array variable. Thus, in Java all arrays are dynamically allocated.

**Default array values in Java:**

If we don’t assign values to array elements, and try to access them, compiler does not produce error [as in case of simple variables](https://www.geeksforgeeks.org/g-fact-50/). Instead it assigns values which aren’t garbage.

Below are the default assigned values.

* boolean : false
* int : 0
* double : 0.0
* String : null
* User Defined Type : null

|  |
| --- |
| **Example:**  **Java program to demonstrate default values of array elements**  class ArrayDemo  {  public static void main(String[] args)  {  System.out.println("String array default values:");  String str[] = new String[5];  for (String s : str)  System.out.print(s + " ");  System.out.println("\n\nInteger array default values:");  int num[] = new int[5];  for (int val : num)  System.out.print(val + " ");  System.out.println("\n\nDouble array default values:");  double dnum[] = new double[5];  for (double val : dnum)  System.out.print(val + " ");  System.out.println("\n\nBoolean array default values:");  boolean bnum[] = new boolean[5];  for (boolean val : bnum)  System.out.print(val + " ");  System.out.println("\n\nReference Array default values:");  ArrayDemo ademo[] = new ArrayDemo[5];  for (ArrayDemo val : ademo)  System.out.print(val + " ");  }  } |

**Output:**

String array default values:

null null null null null

Integer array default values:

0 0 0 0 0

Double array default values:

0.0 0.0 0.0 0.0 0.0

Boolean array default values:

false false false false false

Reference Array default values:

null null null null null

**Array Literal:**

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

**Syntax:**

type[] var-name = new type[]{l1,l2,l3........};

or

type var-name[]={l1,l2,l3........}; // from java version 8

**Example:**

// Declaring array literal

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

or

int intArray[]={1,2,3,4,5,6,7,8,9,10 };

**Array Length:**

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

**Syntax:**

arrayname.length

**Example:**

class Cars

{

public static void main(String args[])

{

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

System.out.println(cars.length);

}

}

**Output**:

4

**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 of Java Array**:

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse or access an 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

**Example to declare, instantiate and initialize the java array together by:**

class Testarray1

{

public static void main(String args[])

{

int a[]={33,3,4,5};//declaration, instantiation and initialization in a single line.

//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:**

There is also a "for-each" loop, which is used exclusively to loop through elements in arrays:

**Syntax:**

for (type variable : arrayname) {

body of the loop;

}

If you compare the for loop and **for-each** loop, you will see that the for-each method is easier to write, it does not require a counter (using the length property), and it is more readable.

**Example:**

//Java Program to print the array elements using for-each loop

class Testarray1

{

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

**Limitations:**

* For-each loops are not appropriate when you want to modify the array:

eg.

for (int num : marks)

{

// only changes num, not the array element

num = num\*2;

}

* For-each loops do not keep track of index. So we can not obtain array index using For-Each loop

eg.

for (int num : numbers)

{

if (num == target)

{

return ???; // do not know the index of num

}

}

* For-each only iterates forward over the array in single steps

// cannot be converted to a for-each loop

eg.

for (int i=numbers.length-1; i>0; i--)

{

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

}

* For-each cannot process two decision making statements at once

// cannot be easily converted to a for-each loop

eg.

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

{

if (numbers[i] == arr[i])

{ ...

}}

**Arrays of Objects:**

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

Student[] arr = new Student[7]; //student is a user-defined class

**Example:**

/ /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 array are objects of a class Student.

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

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

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

**Example:**

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]);

}

}

Runtime error:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2

at GFG.main(File.java:12)

**Example:**

Here is a complete example showing how to create, initialize, and process arrays −

class TestArray3 {

public static void main(String[] args) {

double[] myList = {1.9, 2.9, 3.4, 3.5};

// Print all the array elements

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

System.out.println(myList[i] + " ");

}

// Summing all elements

double total = 0;

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

total += myList[i];

}

System.out.println("Total is " + total);

// Finding the largest element

double max = myList[0];

for (int i = 1; i < myList.length; i++) {

if (myList[i] > max) max = myList[i];

}

System.out.println("Max is " + max);

}

}

**Output:**

1.9

2.9

3.4

3.5

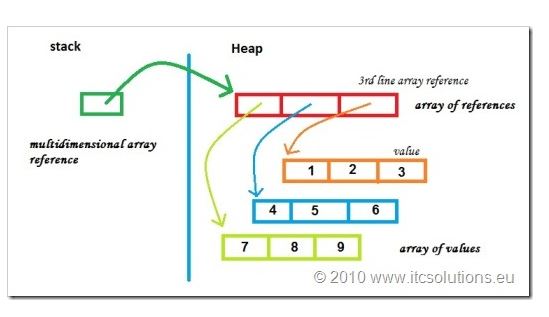
Total is 11.7

Max is 3.5

**Multidimensional Arrays:**

A multidimensional array is an array containing one or more arrays. Multidimensional arrays are **arrays of arrays** with each element of the array holding the reference of other array.

These are also known as [Jagged Arrays](https://www.geeksforgeeks.org/jagged-array-in-java/). A multidimensional array is created by appending one set of square brackets ([]) per dimension.

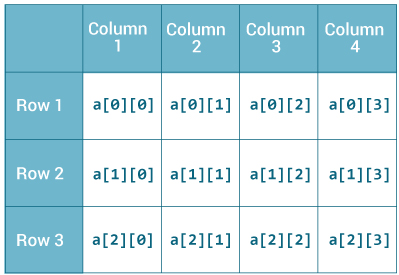


**Syntax:**

type[][] arrname //a 2D array or matrix

type[][][] arrname //a 3D array

Here, a is a two-dimensional (2d) array.



**How to initialize a 2d array in Java?**

**Example** to initialize a 2d array in Java.

int[][] a = {

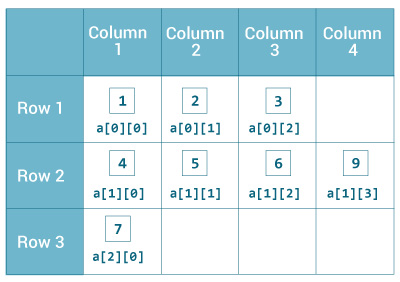
{1, 2, 3},

{4, 5, 6, 9},

{7},

};

each component of array a is an array in itself, and length of each rows is also different.



**Example**:

To find length of each row

class MultidimentionalArray

{

public static void main(String[] args)

{

int[][] a = {

{1, 2, 3},

{4, 5, 6, 9},

{7},

};

System.out.println("Length of row 1: " + a[0].length);

System.out.println("Length of row 2: " + a[1].length);

System.out.println("Length of row 3: " + a[2].length);

}

}

**Output:**

Length of row 1: 3

Length of row 2: 4

Length of row 3: 1

Since each component of a multidimensional array is also an array(a[0],a[1] and a[2] are also arrays),you

can use length attribute to find the length of each rows.

**Example:**

Print all elements of 2d array Using for Loop

class MultidimentionalArray

{

public static void main(String[] args)

{

int[][] a = {

{1, -2, 3},

{-4, -5, 6, 9},

{7},

};

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

for(int j = 0; j < a[i].length; j++) {

System.out.println(a[i][j]);

}

}

}

}

**Output:**

1

-2

3

-4

-5

6

9

7

**Example**:

Print all elements of 2d array Using for each Loop

class MultidimentionalArray

{

public static void main(String[] args)

{

int[][] a = {

{1, -2, 3},

{-4, -5, 6, 9},

{7},

};

for (int[] innerArray : a){

for(int data : innerArray){

System.out.println(data);

}

}

}

}

**Output:**

1

-2

3

-4

-5

6

9

7

**How to initialize a 3d array in Java?**

**Example** : to initialize a 3d array in Java:

int[][][] test = {

{

{1, -2, 3},

{2, 3, 4}

},

{

{-4, -5, 6, 9},

{1},

{2, 3}

}

};

Basically, 3d array is an array of 2d arrays.

Similar like 2d arrays, rows of 3d arrays can vary in length.

**Example**: Print all elements of 3d array Using for Loop

class ThreeArray {

public static void main(String[] args) {

// test is a 3d array

int[][][] test = {

{

{1, -2, 3},

{2, 3, 4}

},

{

{-4, -5, 6, 9},

{1},

{2, 3}

}

};

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

for(int j=0;j<test[i].lenth;j++){

for(int k=0;k<test[i][j].length;k++){

System.out.println(test[i][j][k]);

}

}

}

}

}

**Output:**

1

-2

3

2

3

4

-4

-5

6

9

1

2

3

**Example**: Print all elements of 3d array Using for each Loop

class ThreeArray {

public static void main(String[] args) {

// test is a 3d array

int[][][] test = {

{

{1, -2, 3},

{2, 3, 4}

},

{

{-4, -5, 6, 9},

{1},

{2, 3}

}

};

for(int[][] arr2d: test){

for(int[] arr1d: arr2d){

for(int item: arr1d){

System.out.println(item);

}

}

}

}

}

**Output:**

1

-2

3

2

3

4

-4

-5

6

9

1

2

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.

**Example:**

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[]){

printArray(new int[]{10,22,44,66});//passing anonymous array to method

}}

**Output:**

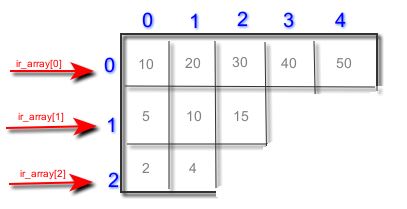
10

22

44

66

**Jagged Array in Java:**



[Jagged array](https://en.wikipedia.org/wiki/Jagged_array) is array of arrays such that member arrays can be of different sizes, i.e., we can create a 2-D arrays but with variable number of columns in each row.

If we are creating odd number of columns in a 2D array, it is known as a jagged array. In other words, it is an array of arrays with different number of columns.

**Example** to illustrate the jagged array :

class TestJaggedArray

{

public static void main(String[] args){

 //declaring a 2D array with 3 rows

 int arr[][] = new int[3][];

//First row has 3 columns

arr[0] = new int[3];

//second row has 4 columns

arr[1] = new int[4];

//third row has 2 columns

arr[2] = new int[2];

 //initializing a jagged array

 int count = 0;

 for (int i=0; i<arr.length; i++)

 for(int j=0; j<arr[i].length; j++)

 arr[i][j] = count++;

 //printing the data of a jagged array

for (int i=0; i<arr.length; i++){

for (int j=0; j<arr[i].length; j++){

System.out.print(arr[i][j]+" ");

}

 System.out.println();//new line

}

}

}

**Output:**

0 1 2

3 4 5 6

7 8

**Example 2**:for jagged array:

|  |
| --- |
| class Main  {  public static void main(String[] args)  {  int r = 5;  // Declaring 2-D array with 5 rows  int arr[][] = new int[r][];  // Creating a 2D array such that first row  // has 1 element, second row has two  // elements and so on.  for (int i=0; i<arr.length; i++)  arr[i] = new int[i+1];  // Initializing array  int count = 0;  for (int i=0; i<arr.length; i++)  for(int j=0; j<arr[i].length; j++)  arr[i][j] = count++;  // Displaying the values of 2D Jagged array  System.out.println("Contents of 2D Jagged Array");  for (int i=0; i<arr.length; i++)  {  for (int j=0; j<arr[i].length; j++)  System.out.print(arr[i][j] + " ");  System.out.println();  }  }  } |

**Output:**

Contents of 2D Jagged Array

0

1 2

3 4 5

6 7 8 9

10 11 12 13 14

**Passing Arrays to Methods:**

Like variables, we can also pass arrays to methods. We can pass the java array to method so that we can reuse the same logic on any array.

**Example:**

sum for calculating sum of array’s values:

Java program to demonstrate passing of array to method

class Test

{

// Driver method

public static void main(String args[])

{

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

// passing array to method sum

sum(arr);

}

public static void sum(int[] arr)  //user defined method

{

// 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

**Example**:

simple example to get the minimum number of an array using a method.

Java Program to demonstrate the way of passing an array

to method.

class Testarray2{

//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

**Returning Arrays from Methods:**

As usual, a method can also return an array.

**Example**:

program returns an array from method m1.

|  |
| --- |
| 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

[**Class**](https://www.geeksforgeeks.org/java-lang-class-class-java-set-1/)**Objects for Arrays:**

Every array has an associated [Class](https://www.geeksforgeeks.org/java-lang-class-class-java-set-1/) object, shared with all other arrays with the same component type.

In Java, an array is an object. For array object, a proxy class is created whose name can be obtained by ***getClass().getName()*** method on the object.

**Example:**

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().getName());

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

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

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

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

}

}

Output:

[I

java.lang.Object

[B

[S

[Ljava.lang.String;

**Explanation :**

* The string “[I” is the run-time type signature for the class object “array with component type int“.
* The only direct superclass of any array type is [java.lang.Object](https://www.geeksforgeeks.org/object-class-in-java/).
* The string “[B” is the run-time type signature for the class object “array with component type byte“.
* The string “[S” is the run-time type signature for the class object “array with component type short“.
* 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.

**Copying a Java Array:**

There are several techniques you can use to copy arrays in Java.

1. Copying Arrays Using Assignment Operator
2. Using Looping Construct to Copy Arrays
3. Copying Arrays Using arraycopy() method
4. Copying Arrays Using copyOfRange() method

**1. Copying Arrays Using Assignment Operator**

**Example:**

class AssignmentOp

{

public static void main(String[] args)

{

int [] numbers = {1, 2, 3, 4, 5, 6};

int [] positiveNumbers = numbers; // copying arrays

for (int number: positiveNumbers) {

System.out.print(number + ",");

}

}

}

**Output:**

1,2,3,4,5,6,

Though this technique to copy arrays seem to work perfectly, there is a problem with it.

If you change elements of one array in the above example, corresponding elements of the other array is also changed.

class AssignmentOp

{

public static void main(String[] args)

{

int [] numbers = {1, 2, 3, 4, 5, 6};

int [] positiveNumbers = numbers; // copying arrays

numbers[0] = -1;

for (int number: positiveNumbers){

System.out.print(number + ", ");

}

}

}

**Output:**

-1,2,3,4,5,6,

When the first element of numbers array is changed to -1, the first element of positive numbers array also becomes -1. It's because both arrays refers to the same array object.

This is called [shallow copy](http://stackoverflow.com/questions/1175620/in-java-what-is-a-shallow-copy).

However, most often, we need deep copy rather than shallow copy. A deep copy copies the values creating the new array object.

**2. Using Looping Construct to Copy Arrays**

**Example:**

import java.util.Arrays;

class ArraysCopy

{

public static void main(String[] args) {

int [] source = {1, 2, 3, 4, 5, 6};

int [] destination = new int[6];

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

destination[i] = source[i];

}

// converting array to string

System.out.println(Arrays.toString(destination));

}

}

**Output:**

[1, 2, 3, 4, 5, 6]

**3.Copying Arrays Using arraycopy() method**

The [System class](https://docs.oracle.com/javase/8/docs/api/java/lang/System.html) contains arraycopy() method that allows you to copy data from one array to another.

The arraycopy() method is efficient as well as flexible. The method allows you to copy a specified portion of the source array to the destination array.

public static void arraycopy(Object src, int srcPos,

Object dest, int destPos, int length)

Here,

* src-array you want to copy
* srcPos- starting position (index) in the src array
* dest - elements of src array will be copied to this array
* desPos - starting position (index) in the dest array
* length - number of elements to copy

**Syntax :**

System.arraycopy(Object src, int srcPos,Object dest, int destPos, int length

);

**Example** of Copying an Array in Java

We need to use import java.util.Arrays;

to use arraycopy()

Java Program to copy a source array into a destination array in Java

import java.util.Arrays;

class TestArrayCopyDemo

{

public static void main(String[] args) {

//declaring a source array

char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',  'i', 'n', 'a', 't', 'e', 'd' };

 //declaring a destination array

char[] copyTo = new char[7];

//copying array using System.arraycopy() method

System.arraycopy(copyFrom, 2, copyTo, 0, 7);

//printing the destination array

System.out.println(String.valueOf(copyTo));

}

}

**Output:**

caffein

**Example:**

Using arraycopy() & Arrays.toString()

import java.util.Arrays;

class ArraysCopy

{

public static void main(String[] args) {

int[] n1 = {2, 3, 12, 4, 12, -2};

int[] n3 = new int[5];

// Creating n2 array of having length of n1 array

int[] n2 = new int[n1.length];

// copying entire n1 array to n2

System.arraycopy(n1, 0, n2, 0, n1.length);

System.out.println("n2 = " + Arrays.toString(n2));

// copying elements from index 2 on n1 array

// copying element to index 1 of n3 array

// 2 elements will be copie

System.arraycopy(n1, 2, n3, 1, 2);

System.out.println("n3 = " + Arrays.toString(n3));

}

}

**Output**:

n2 = [2, 3, 12, 4, 12, -2]

n3 = [0, 12, 4, 0, 0]

**4.Copying Arrays Using copyOfRange() method**

you can use copyOfRange() method defined in [java.util.Arrays](https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html) class to copy arrays. You do not need to create the destination array before this method is called.

**copyOfRange() for different datatypes:**

**Syntax:**

public static  int[] copyOfRange(int[] original,

int from,

int to)

public static  long[] copyOfRange(long[] original,

int from,

int to)

public static char[] copyOfRange(char[] original,

int from,

int to)

public static float[] copyOfRange(float[] original,

int from,

int to)

public static double[] copyOfRange(double[] original,

int from,

int to)

public static boolean[] copyOfRange(boolean[] original,

int from,

int to)

**Parameters:**

original - the array from which a range is to be copied

from - the initial index of the range to be copied, inclusive

to - the final index of the range to be copied, exclusive. (This index may lie outside the array.)

**Returns:**

a new array containing the specified range from the original array, truncated or padded with zeros to obtain the required length

**Throws:**

[ArrayIndexOutOfBoundsException](https://docs.oracle.com/javase/8/docs/api/java/lang/ArrayIndexOutOfBoundsException.html) - if from < 0 or from > original.length

[IllegalArgumentException](https://docs.oracle.com/javase/8/docs/api/java/lang/IllegalArgumentException.html) - if from > to

[NullPointerException](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) - if original is null

**Example:**

To use toString() and copyOfRange() method

import java.util.Arrays;\*/

class ArraysCopy

{

public static void main(String[] args) {

int[] source = {2, 3, 12, 4, 12, -2};

// copying entire source array to destination

int[] destination1 = Arrays.copyOfRange(source, 0, source.length);

System.out.println("destination1 = " + Arrays.toString(destination1));

// copying from index 2 to 5 (5 is not included)

int[] destination2 = Arrays.copyOfRange(source, 2, 5);

System.out.println("destination2 = " + Arrays.toString(destination2));

}

}

**Output:**

destination1 = [2, 3, 12, 4, 12, -2]

destination2 = [12, 4, 12]

**Copying 2d Arrays Using Loop**

**Example:**

import java.util.Arrays;

class ArraysCopy

{

public static void main(String[] args)

{

int[][] source = {

{1, 2, 3, 4},

{5, 6},

{0, 2, 42, -4, 5}

};

int[][] destination = new int[source.length][];

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

// allocating space for each row of destination array

destination[i] = new int[source[i].length];

for (int j = 0; j < destination[i].length;j++) {

destination[i][j] = source[i][j];

}

}

// displaying destination array

System.out.println(Arrays.deepToString(destination));

}

}

**Output:**

[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]

**Example:**

Using arraycopy()

import java.util.Arrays;

class AssignmentOperator

{

public static void main(String[] args) {

int[][] source = {

{1, 2, 3, 4},

{5, 6},

{0, 2, 42, -4, 5}

};

int[][] destination = new int[source.length][];

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

// allocating space for each row of destination array

destination[i] = new int[source[i].length];

System.arraycopy(source[i], 0, destination[i], 0, destination[i].length);

}

// displaying destination array

System.out.println(Arrays.deepToString(destination));

}

}

**Output:**

[[1, 2, 3, 4], [5, 6], [0, 2, 42, -4, 5]]

**Cloning an Array in Java:**

Since, Java array implements the Cloneable interface, we can create the clone of the Java array. If we create the clone of a single-dimensional array, it creates the deep copy of the Java array. It means, it will copy the actual value. But, if we create the clone of a multidimensional array, it creates the shallow copy of the Java array which means it copies the references.

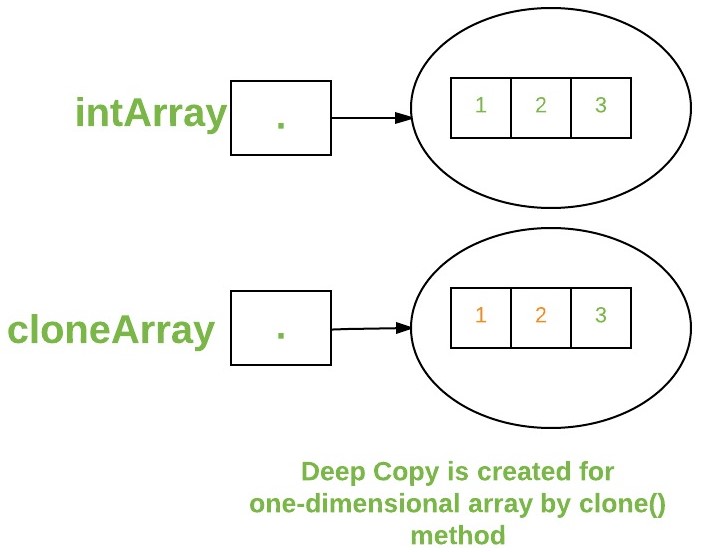
**Syntax:**

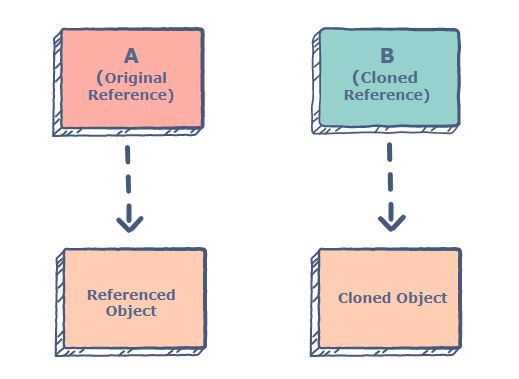
type cloneARR[]=ArrName.clone();

**Cloning of 1D array:**

**Deep copy:**

deep copy of the Java array. It means, it will copy the actual value.





**Example:** cloning of 1D array:

|  |
| --- |
| Java program to demonstrate cloning of one-dimensional arrays    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:**

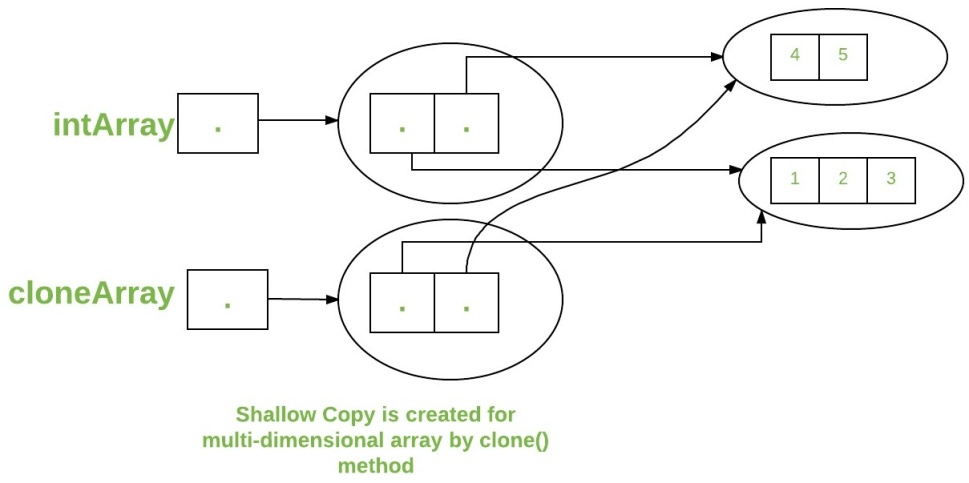
false

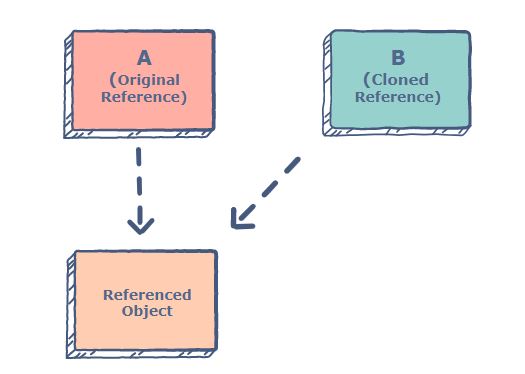
1 2 3

**cloning of mltidimentional array:**

**Shallow copy:**

A clone of a multidimensional 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.





**Example:**2D array

|  |
| --- |
| 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

**How to compare two arrays in Java?**

**Example 1:**

|  |
| --- |
| class Test  {  public static void main (String[] args)  {  int arr1[] = {1, 2, 3};  int arr2[] = {1, 2, 3};  if (arr1 == arr2) // Same as arr1.equals(arr2)  System.out.println("Same");  else  System.out.println("Not same");  }  } |

In Java, [arrays are first class objects](https://www.geeksforgeeks.org/g-fact-65/). In the above program, arr1 and arr2 are two references to two different objects. So when we compare arr1 and arr2, two reference variables are compared, therefore we get the output as “Not Same”

**Output:**

Not Same

**How to compare array contents?**

A simple way is to run a loop and compare elements one by one. Java provides a direct method ***Arrays.equals() to compare two arrays.*** Actually, there is a list of equals() methods in Arrays class for different primitive types (int, char, ..etc) and one for Object type (which is base of all classes in Java).

we need to import ***java.util.Arrays to use Arrays.equals().***

|  |
| --- |
| **Example:**  import java.util.Arrays;  class Test  {  public static void main (String[] args)  {  int arr1[] = {1, 2, 3};  int arr2[] = {1, 2, 3};  if (Arrays.equals(arr1, arr2))  System.out.println("Same");  else  System.out.println("Not same");  }  } |

**Output:**

Same

**How to Deep compare array contents?**

As seen above, the Arrays.equals() works fine and compares arrays contents. Now the questions, what if the arrays contain arrays inside them or some other references which refer to different object but have same values.

**Example:**

|  |
| --- |
| import java.util.Arrays;  class Test  {  public static void main (String[] args)  {  // inarr1 and inarr2 have same values  int inarr1[] = {1, 2, 3};  int inarr2[] = {1, 2, 3};  Object[] arr1 = {inarr1};  // arr1 contains only one element  Object[] arr2 = {inarr2};  // arr2 also contains only one element  if (Arrays.equals(arr1, arr2))  System.out.println("Same");  else  System.out.println("Not same");  }  } |

**Output:**

Not Same

So Arrays.equals() is not able to do deep comparison. Java provides another method for this ***Arrays.deepEquals() which does deep comparison***.

|  |
| --- |
| **Example:**  import java.util.Arrays;  class Test  {  public static void main (String[] args)  {  int inarr1[] = {1, 2, 3};  int inarr2[] = {1, 2, 3};  Object[] arr1 = {inarr1};  // arr1 contains only one element  Object[] arr2 = {inarr2};  // arr2 also contains only one element  if (Arrays.deepEquals(arr1, arr2))  System.out.println("Same");  else  System.out.println("Not same");  }  } |

**Output:**

Same

**Final arrays in Java :**

|  |
| --- |
| When an array variable is declared as final its values cannot be changed or we cannot assign values to that array variable. However, we can change the values of the array elements. [Arrays are objects](https://www.geeksforgeeks.org/g-fact-65/) and [object variables are always references](https://www.geeksforgeeks.org/g-fact-46/) in Java. So, when we declare an object variable as final, it means that the variable cannot be changed to refer to anything else.  **Example:**  class Test  {  public static void main(String args[])  {  final int arr[] = {1, 2, 3, 4, 5};  // Note: arr is final  for (int i = 0; i < arr.length; i++)  {  arr[i] = arr[i]\*10;  System.out.println(arr[i]);  }  }  } |

**Output:**

10

20

30

40

50

The array arr is declared as final, but the elements of array are changed without any problem.

**Example:**

The following program compiles without any error and program fails in compilation.

|  |
| --- |
| class Test  {  int p = 20;  public static void main(String args[])  {  final Test t = new Test();  t.p = 30;  System.out.println(t.p);  }  } |

**Output:**

30

|  |
| --- |
| **Example:**  class Test  {  int p = 20;  public static void main(String args[])  {  final Test t1 = new Test();  Test t2 = new Test();  t1 = t2;  System.out.println(t1.p);  }  } |

**Output:**

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The final local variable t1 cannot be assigned. It must be blank and not using a compound assignment

at arrays.Test.main(Test.java:9)

So a final array means that the array variable which is actually a reference to an object, cannot be changed to refer to anything else, but the members of array can be modified.

**Example:**

|  |
| --- |
| class Test  {  public static void main(String args[])  {  final int arr1[] = {1, 2, 3, 4, 5};  int arr2[] = {10, 20, 30, 40, 50};  arr2 = arr1;  arr1 = arr2;  for (int i = 0; i < arr2.length; i++)  System.out.println(arr2[i]);  }  } |

**Output:**

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The final local variable arr1 cannot be assigned. It must be blank and not using a compound assignment

at arrays.Test.main(Test.java:10)

**Array vs ArrayList in Java**

In Java, following are two different ways to create an array.

**Array:**

Simple fixed sized arrays that we create in Java, like below

int arr[] = new int[10]

[**ArrayList**](https://www.geeksforgeeks.org/arraylist-in-java/)**:**

Dynamic sized arrays in Java that implement List interface.

**Syntax:**

ArrayList<Type> arrL = new ArrayList<Type>();

Here Type is the type of elements in ArrayList to

be created.

**Differences between Array and ArrayList**

1. An array is basic functionality provided by Java. ArrayList is part of collection framework in Java.

Therefore array members are accessed using [], while ArrayList has a set of methods to access

elements and modify them.

**Example:**

|  |
| --- |
| A Java program to demonstrate differences between array and ArrayList  import java.util.ArrayList;  import java.util.Arrays;    class Test  {   public static void main(String args[])  {  /\* ........... Normal Array............. \*/  int[] arr = new int[2];  arr[0] = 1;  arr[1] = 2;  System.out.println(arr[0]);  /\*............ArrayList..............\*/   // Create an arrayList with initial capacity 2  ArrayList<Integer> arrL = new ArrayList<Integer>(2);  // Add elements to ArrayList  arrL.add(1);  arrL.add(2);   // Access elements of ArrayList  System.out.println(arrL.get(0));  }  } |

**Output:**

1

1

.

2. Array is a fixed size data structure while ArrayList is not. One need not to mention the size of Arraylist

while creating its object. Even if we specify some initial capacity, we can add more elements.

|  |
| --- |
| **Example:**  A Java program to demonstrate differences between array and ArrayList  import java.util.ArrayList;  import java.util.Arrays;  class Test  {  public static void main(String args[])  {  /\* ........... Normal Array............. \*/  // Need to specify the size for array  int[] arr = new int[3];  arr[0] = 1;  arr[1] = 2;  arr[2] = 3;  // We cannot add more elements to array arr[]  /\*............ArrayList..............\*/  // Need not to specify size  ArrayList<Integer> arrL = new ArrayList<Integer>();  arrL.add(1);  arrL.add(2);  arrL.add(3);  arrL.add(4);  // We can add more elements to arrL  System.out.println(arrL);  System.out.println(Arrays.toString(arr));  }  } |

**Output:**

[1, 2, 3, 4]

[1, 2, 3]

.

3. Array can contain both primitive data types as well as objects of a class depending on the definition of

the array. However, ArrayList only supports object entries, not the primitive data types.

**Note:** When we do arraylist.add(1); : it converts the primitive int data type into an Integer object.

**Sample Code**:

|  |
| --- |
| import java.util.ArrayList;  class Test  {  public static void main(String args[])  {  // allowed  int[] array = new int[3];  // allowed, however, need to be intialized  Test[] array1 = new Test[3];  // not allowed (Uncommenting below line causes  // compiler error)  // ArrayList<char> arrL = new ArrayList<char>();  // Allowed  ArrayList<Integer> arrL1 = new ArrayList<>();  ArrayList<String> arrL2 = new ArrayList<>();  ArrayList<Object> arrL3 = new ArrayList<>();  }  } |

.

4. Since ArrayList can’t be created for primitive data types, members of ArrayList are always references

to objects at different memory locations. Therefore in ArrayList, the actual objects are never stored at

contiguous locations. References of the actual objects are stored at contiguous locations.In array, it

depends whether the arrays is of primitive type or object type. In case of primitive types,

actual values are contiguous locations, but in case of objects, allocation is similar to ArrayList.

**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.

**Class Hierarchy:**

java.lang.Object

↳ java.util.Arrays

**Class Declaration:**

public class Arrays

extends Object

**Syntax to use Array:**

Arrays.<function name>;

**Need for the Java-Arrays Class:**

There are often times when [loops](https://www.geeksforgeeks.org/loops-in-java/) 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.

Arrays class provides several static methods that can be used to perform these tasks directly without the use of loops.

**Methods in Java Array:**

The Arrays class of the [java.util package](https://www.geeksforgeeks.org/java-util-package-java/) contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

[**static <T> List<T> asList(T… a)**](https://www.geeksforgeeks.org/arrays-aslist-method-in-java-with-examples/)**:**

This method returns a fixed-size list backed by the specified Arrays.

**Example:**

|  |
| --- |
| Java program to demonstrate **Arrays.asList()** 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 convert the elements as List  System.out.println("Integer Array as List: " + Arrays.asList(intArr));  }  } |

**Output**:

Integer Array as List: [[I@232204a1]

[**static int binarySearch(elementToBeSearched)**](https://www.geeksforgeeks.org/arrays-binarysearch-java-examples-set-1/)**:**

This method searches for the specified element in the array with the help of Binary Search algorithm.

|  |
| --- |
| **Example:**  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, intKey));  }  } |

**Output**:

22 found at index = 3

[**static <T> int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<T> c)**](https://www.geeksforgeeks.org/arrays-binarysearch-in-java-with-examples-set-2-search-in-subarray/):

This method searches a range of the specified array for the specified object using the binary search algorithm.

|  |
| --- |
| 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

**compare(array 1, array 2):**

This method compares two arrays passed as parameters lexicographically.

|  |
| --- |
| Java program to demonstrate **Arrays.compare()** method  **Example**:  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

**compareUnsigned(array 1, array 2)**:

This method compares two arrays lexicographically, numerically treating elements as unsigned.

|  |
| --- |
| 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

[**copyOf(originalArray, newLength)**](https://www.geeksforgeeks.org/arrays-copyof-in-java-with-examples/)**:**

This method copies the specified array, truncating or padding with the default value (if necessary) so the copy has the specified length.

|  |
| --- |
| 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]

[**copyOfRange(originalArray, fromIndex, endIndex)**](https://www.geeksforgeeks.org/java-util-arrays-copyofrange-java/)**:**

This method copies the specified range of the specified array into a new Arrays.

|  |
| --- |
| 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]

[**static boolean deepEquals(Object[] a1, Object[] a2)**](https://www.geeksforgeeks.org/java-util-arrays-deepequals-java/)**:**

This method returns true if the two specified arrays are deeply equal to one another.

|  |
| --- |
| 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

**static int deepHashCode(Object[] a):**

This method returns a hash code based on the “deep contents” of the specified Arrays.

|  |
| --- |
| 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

[**static String deepToString(Object[] a)**](https://www.geeksforgeeks.org/arrays-deeptostring-in-java-with-example/)**:**

This method returns a string representation of the “deep contents” of the specified Arrays.

|  |
| --- |
| 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]]

[**equals(array1, array2)**](https://www.geeksforgeeks.org/java-util-arrays-equals-java-examples/)**:**

This method checks if both the arrays are equal or not.

|  |
| --- |
| 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

[**fill(originalArray, fillValue)**](https://www.geeksforgeeks.org/arrays-fill-java-examples/)**:**

This method assigns this fillValue to each index of this Arrays.

|  |
| --- |
| Java program to demonstrate **Arrays.fill()** method    import java.util.Arrays;  public class Main  {  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]

**hashCode(originalArray):**

This method returns an integer hashCode of this array instance.

|  |
| --- |
| 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

**mismatch(array1, array2):**

This method finds and returns the index of the first unmatched element between the two specified arrays.

|  |
| --- |
| Java program to demonstrate **Arrays.mismatch()** 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("The element mismatched at index: "+ Arrays.mismatch(intArr, intArr1));  }  } |

**Output:**

The element mismatched at index: 1

**parallelPrefix(originalArray, fromIndex, endIndex, functionalOperator):**

This method performs parallelPrefix for the given range of the array with the specified functional operator.

**parallelPrefix(originalArray, operator):**

This method performs parallelPrefix for complete array with the specified functional operator.

**parallelSetAll(originalArray, functionalGenerator):**

This method set all the elements of this array in parallel, using the provided generator function.

[**parallelSort(originalArray)**](https://www.geeksforgeeks.org/java-8-arrays-parallelsort-method-with-examples/)**:**

This method sorts the specified array using parallel sort.

|  |
| --- |
| Java program to demonstrate **Arrays.parallelSort()** 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 parallelSort  Arrays.parallelSort(intArr);  System.out.println("Integer Array: " + Arrays.toString(intArr));  }  } |

**Output**:

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

**setAll(originalArray, functionalGenerator)**:

This method sets all the element of the specified array using the generator function provided.

[**sort(originalArray)**](https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/)**:**

This method sorts the complete array in ascending order.

|  |
| --- |
| 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]

[**sort(originalArray, fromIndex, endIndex)**](https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/):

This method sorts the specified range of array in ascending order.

|  |
| --- |
| 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]

[**static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator< super T> c)**](https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/)**:**

This method sorts the specified range of the specified array of objects according to the order induced by the specified comparator.

|  |
| --- |
| 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

[**static <T> void sort(T[] a, Comparator< super T> c)**](https://www.geeksforgeeks.org/arrays-sort-in-java-with-examples/)**:**

This method sorts the specified array of objects according to the order induced by the specified comparator.

|  |
| --- |
| 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

131 aaaa nyc

121 cccc jaipur

**spliterator(originalArray):**

This method returns a Spliterator covering all of the specified Arrays.

|  |
| --- |
| 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@232204a1

**spliterator(originalArray, fromIndex, endIndex):**

This method returns a Spliterator of the type of the array covering the specified range of the specified Arrays.

|  |
| --- |
| 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@232204a1

[**stream(originalArray)**](https://www.geeksforgeeks.org/arrays-stream-method-in-java/)**:**

This method returns a sequential stream with the specified array as its source.

|  |
| --- |
| Java program to demonstrate **Arrays.stream()** 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 Stream from the array  System.out.println("Integer Array: " + Arrays.stream(intArr));  }  } |

**Output**:

Integer Array: java.util.stream.IntPipeline$Head@4aa298b7

[**toString(originalArray)**](https://www.geeksforgeeks.org/arrays-tostring-in-java-with-examples/)**:**

This method returns a String representation of the contents of this Arrays. 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.

|  |
| --- |
| 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]

**Linked List vs Array**

Both [Arrays](https://www.geeksforgeeks.org/array-data-structure/) and [Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/)can be used to store linear data of similar types, but they both have some advantages and disadvantages over each other.





**Key Differences Between Array and Linked List**

1. An array is the data structure that contains a collection of similar type data elements whereas the Linked list is considered as non-primitive data structure contains a collection of unordered linked elements known as nodes.  
2. In the array the elements belong to indexes, i.e., if you want to get into the fourth element you have to write the variable name with its index or location within the square bracket.  
3. In a linked list though, you have to start from the head and work your way through until you get to the fourth element.  
4. Accessing an element in an array is fast, while Linked list takes linear time, so it is quite a bit slower.  
5. Operations like insertion and deletion in arrays consume a lot of time. On the other hand, the performance of these operations in Linked lists is fast.  
6. Arrays are of fixed size. In contrast, Linked lists are dynamic and flexible and can expand and contract its size.  
7. In an array, memory is assigned during compile time while in a Linked list it is allocated during execution or runtime.  
9. Elements are stored consecutively in arrays whereas it is stored randomly in Linked lists.  
10. The requirement of memory is less due to actual data being stored within the index in the array. As against, there is a need for more memory in Linked Lists due to storage of additional next and previous referencing elements.  
11. In addition memory utilization is inefficient in the array. Conversely, memory utilization is efficient in the linked list.

**Following are the points in favor of Linked Lists.**

(1) The size of the arrays is fixed: So we must know the upper limit on the number of elements in advance. Also, generally, the allocated memory is equal to the upper limit irrespective of the usage, and in practical uses, the upper limit is rarely reached.

(2) Inserting a new element in an array of elements is expensive because a room has to be created for the new elements and to create room existing elements have to be shifted.

**Example**:

suppose we maintain a sorted list of IDs in an array id[].

id[] = [1000, 1010, 1050, 2000, 2040, …..].

And if we want to insert a new ID 1005, then to maintain the sorted order, we have to move all the elements after 1000 (excluding 1000).

Deletion is also expensive with arrays until unless some special techniques are used. For example, to delete 1010 in id[], everything after 1010 has to be moved.

**So Linked list provides the following two advantages over arrays**

1) Dynamic size  
2) Ease of insertion/deletion

**Linked lists have following drawbacks:**

1) Random access is not allowed. We have to access elements sequentially starting from the first node. So we cannot do a binary search with linked lists.  
2) Extra memory space for a pointer is required with each element of the list.  
3) Arrays have better cache locality that can make a pretty big difference in performance.