Array

Java Arrays

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

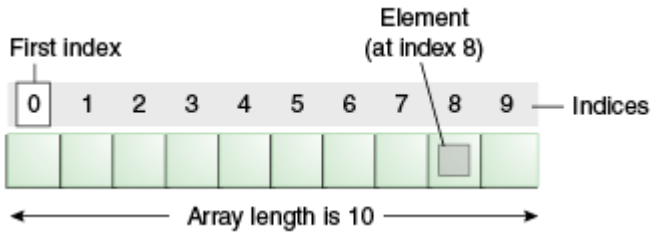
Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

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

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

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

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



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

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=**new** datatype[size];

Example of Java Array

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

```
1.//Java Program to illustrate how to declare, instantiate, initialize
2.//and traverse the Java array.
3.class Testarray{
4.public static void main(String args[]){
5.int a[]=new int[5];//declaration and instantiation
6.a[0]=10;//initialization
7.a[1]=20;
8.a[2]=70;
9.a[3]=40;
10.a[4]=50;
11.//traversing array
12.for(int i=0;i<a.length;i++)//length is the property of array
13.System.out.println(a[i]);
14.}}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1.int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
3. class Testarray1{
4. public static void main(String args[]){
5. int a[]={33,3,4,5};//declaration, instantiation and initialization
6. //printing array
7. for(int i=0;i<a.length;i++)//length is the property of array
8. System.out.println(a[i]);
9. }}
```

For-each Loop for Java Array

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

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

```
1. for(data_type variable:array){
2. //body of the loop
3. }
```

Let us see the example of print the elements of Java array using the for-each loop.

```
1. //Java Program to print the array elements using for-each loop
2. class Testarray1{
3. public static void main(String args[]){
4. int arr[]={33,3,4,5};
5. //printing array using for-each loop
6. for(int i:arr)
7. System.out.println(i);
8. }}
```

Output:

3

4

5

Passing Array to a Method in Java

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

Let's see the simple example to get the minimum number of an array using a method.

```
1.//Java Program to demonstrate the way of passing an array
1.class Testarray2{
2.//creating a method which receives an array as a parameter
3.static void min(int arr[]){
4.int min=arr[0];
5.for(int i=1;i<arr.length;i++)
6. if(min>arr[i])
7. min=arr[i];
8.
9.System.out.println(min);
10.}
11.
12.public static void main(String args[]){
13.int a[]={33,3,4,5};//declaring and initializing an array
14.min(a);//passing array to method
15.}}
```

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

2.//to method.
3.public class TestAnonymousArray{
4.//creating a method which receives an array as a parameter
5.static void printArray(int arr[]){
6.for(int i=0;i<arr.length;i++)
7.System.out.println(arr[i]);
8.}
9.
10.public static void main(String args[]){
```

```
11.printArray(new int[]{10,22,44,66});//passing anonymous array to metho
d
12.}}
```

```
1. //Java Program to demonstrate the case of
2. //ArrayIndexOutOfBoundsException in a Java Array.
3. public class TestArrayException{
4. public static void main(String args[]){
5. int arr[]={50,60,70,80};
6. for(int i=0;i<=arr.length;i++){
7. System.out.println(arr[i]);
8. }
9. }}
```

Test it Now

Output:

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

Multidimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java

```
1. dataType[][] arrayRefVar; (or)
2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

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

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
1. //Java Program to illustrate the use of multidimensional array
2. class Testarray3{
3. public static void main(String args[]){
4. //declaring and initializing 2D array
5. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
6. //printing 2D array
7. for(int i=0;i<3;i++){
8.  for(int j=0;j<3;j++){
9.   System.out.print(arr[i][j]+" ");
10. }
11. System.out.println();
12.}
13.}}
```

Test it Now

Output:

```
1 2 3
2 4 5
4 4 5
```

Addition of 2 Matrices in Java

Let's see a simple example that adds two matrices.

```
1. //Java Program to demonstrate the addition of two matrices in Java
2. class Testarray5{
3. public static void main(String args[]){
4. //creating two matrices
5. int a[][]={{1,3,4},{3,4,5}};
```

```
6. int b[][]={{1,3,4},{3,4,5}};
7.
8. //creating another matrix to store the sum of two matrices
9. int c[][]=new int[2][3];
10.
11.//adding and printing addition of 2 matrices
12.for(int i=0;i<2;i++){
13.for(int j=0;j<3;j++){
14.c[i][j]=a[i][j]+b[i][j];
15.System.out.print(c[i][j]+" ");
16.}
17.System.out.println();//new line
18.}
19.
20.}}
```

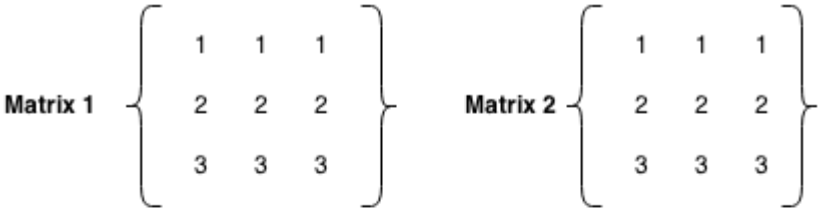
Test it Now

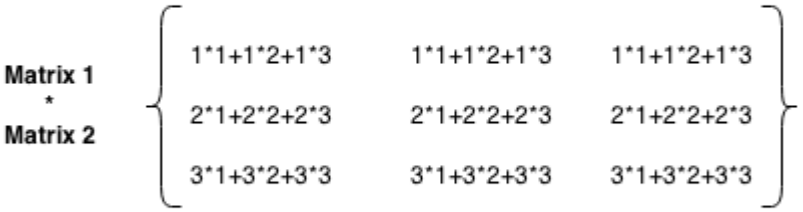
Output:

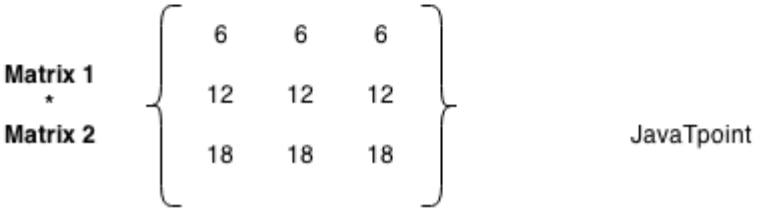
```
2 6 8
6 8 10
```

Multiplication of 2 Matrices in Java

In the case of matrix multiplication, a one-row element of the first matrix is multiplied by all the columns of the second matrix which can be understood by the image given below.







Let's see a simple example to multiply two matrices of 3 rows and 3 columns.

```
1. //Java Program to multiply two matrices
2. public class MatrixMultiplicationExample{
3. public static void main(String args[]){
4. //creating two matrices
5. int a[][]={{1,1,1},{2,2,2},{3,3,3}};
6. int b[][]={{1,1,1},{2,2,2},{3,3,3}};
7.
8. //creating another matrix to store the multiplication of two matrices
9. int c[][]=new int[3][3];  //3 rows and 3 columns
10.
11.//multiplying and printing multiplication of 2 matrices
12.for(int i=0;i<3;i++){
13.for(int j=0;j<3;j++){
14.c[i][j]=0;
15.for(int k=0;k<3;k++)
16.{
17.c[i][j]+=a[i][k]*b[k][j];
18.}//end of k loop
19.System.out.print(c[i][j]+" ");  //printing matrix element
20.}//end of j loop
21.System.out.println();//new line
22.}
23.}}
```

Test it Now

Output:

```
6 6 6
```

```
12 12 12
```

```
18 18 18
```

****

Top 10 Methods for Java Arrays

Category: Basics >> Top 10 September 12, 2013

The following are top 10 methods for Java Array. They are the most voted questions from stackoverflow.

0. Declare an array

```
String[] aArray = new String[5];
String[] bArray = {"a","b","c", "d", "e"};
String[] cArray = new String[]
{"a","b","c","d","e"};
```

1. Print an array in Java

```
int[] intArray = { 1, 2, 3, 4, 5 };
String intArrayString = Arrays.toString(intArray);

// print directly will print reference value
System.out.println(intArray);
// [I@7150bd4d

System.out.println(intArrayString);
// [1, 2, 3, 4, 5]
```

2. Create an ArrayList from an array

```
String[] stringArray = { "a", "b", "c", "d", "e" };
ArrayList<String> arrayList = new ArrayList<String>(Arrays.asList(stringArray));
System.out.println(arrayList);
// [a, b, c, d, e]
```

3. Check if an array contains a certain value

```
String[] stringArray = { "a", "b", "c", "d", "e" };
boolean b = Arrays.asList(stringArray).contains("a");
System.out.println(b);
// true
```

4. Concatenate two arrays

```
int[] intArray = { 1, 2, 3, 4, 5 };
int[] intArray2 = { 6, 7, 8, 9, 10 };
// Apache Commons Lang library
int[] combinedIntArray = ArrayUtils.addAll(intArray, intArray2);
```

5. Declare an array inline

```
method(new String[]{"a", "b", "c", "d", "e"});
```

6. Joins the elements of the provided array into a single String

```
// containing the provided list of elements
// Apache common lang
String j = StringUtils.join(new String[] { "a", "b", "c" }, ",
");
System.out.println(j);
// a, b, c
```

7. Covnert an ArrayList to an array

```
String[] stringArray = { "a", "b", "c", "d", "e" };
ArrayList<String> arrayList = new ArrayList<String>(Arrays.asList(stringArray));
String[] stringArr = new String[arrayList.size()];
arrayList.toArray(stringArr);
for (String s : stringArr)
        System.out.println(s);
```

8. Convert an array to a set

```
Set<String> set = new HashSet<String>(Arrays.asList(stringArray));
System.out.println(set);
//[d, e, b, c, a]
```

9. Reverse an array

```
int[] intArray = { 1, 2, 3, 4, 5 };
ArrayUtils.reverse(intArray);
System.out.println(Arrays.toString(intArray));
//[5, 4, 3, 2, 1]
```

10. Remove element of an array

```
int[] intArray = { 1, 2, 3, 4, 5 };
int[] removed = ArrayUtils.removeElement(intArray, 3);//create a new
array
System.out.println(Arrays.toString(removed));
```

One more - convert int to byte array

```
byte[] bytes = ByteBuffer.allocate(4).putInt(8).array();

for (byte t : bytes) {
   System.out.format("0x%x ", t);
```

Methods in Java Array:

The Arrays class of the **java.util package** contains several static methods that can be used to fill, sort, search, etc in arrays. These are:

1. **static <T> List<T> asList(T... a)**: This method returns a fixed-size list backed by the specified Arrays.

```
// 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]
```

2.**static int binarySearch(elementToBeSearched):** These methods searches for the specified element in the array with the help of 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, intKey));
    }
}
```

Output:

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

3.**static <T> int binarySearch(T[] a, int fromIndex, int toIndex, T key, Comparator<T> c)**: 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
```

4.**compare(array 1, array 2)**: This method compares two arrays passed as parameters lexicographically.

```
// 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
```

5.**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
```

6.**copyOf(originalArray, newLength)**: 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]
```

7.**copyOfRange(originalArray, fromIndex, endIndex)**: 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]
```

8.**static boolean deepEquals(Object[] a1, Object[] a2)**: 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
```

9.**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
```

10.**static String deepToString(Object[] a)**: 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]]
```

11.**equals(array1, array2)**: 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
```

12.**fill(originalArray, fillValue)**: 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]
```

13.**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
```

14.**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
```

15.**parallelPrefix(originalArray, fromIndex, endIndex, functionalOperator)**: This method performs parallelPrefix for the given range of the array with the specified functional operator.

16.**parallelPrefix(originalArray, operator)**: This method performs parallelPrefix for complete array with the specified functional operator.

17.**parallelSetAll(originalArray, functionalGenerator)**: This method set all the elements of this array in parallel, using the provided generator function.

18.**parallelSort(originalArray)**: 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]
```

19.**setAll(originalArray, functionalGenerator)**: This method sets all the element of the specified array using the generator function provided.

20.**sort(originalArray)**: 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]
```

21.**sort(originalArray, fromIndex, endIndex)**: 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]
```

22.**static <T> void sort(T[] a, int fromIndex, int toIndex, Comparator< super T> c)**: 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
```

23.**static <T> void sort(T[] a, Comparator< super T> c)**: This method sorts the specified array of objects according to the order induced by the specified comparator.

```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
```

24.**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
```

25.**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
```

26.**stream(originalArray)**: 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
```

27.**toString(originalArray)**: 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]
```

1.2. Using **compare()** and **compareUnsigned()** methods

The group of methods ***compare()*** and ***compareUnsigned()*** compares two arrays lexicographically. There are two different version of different overloads for `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short` and `Object` arrays. Version one for equality check for two arrays and version two for equality

check for slices of two arrays. The ***compareUnsigned()*** method treats the integer values as unsigned and this overloaded method applicable only for data types which can be signed.

1. A `null` array is lexicographically less than a non-null array.
2. Two arrays are considered equal if both are `null`.
3. If first array and second array are equal then `compare()` returns zero;
4. If the first array (or slice) is lexicographically less than the second array (or slice) returns -ve (negative value).
5. If the first array (or slice) is lexicographically greater than the second array (or slice) returns +ve (positive value).

```
int compare(int[] a, int[] b)
int compare(int[] a, int aFromIndex, int aToIndex, int[] b,
int bFromIndex, int bToIndex)
int compare(int[] a, int[] b)
int compare(int[] a, int aFromIndex, int aToIndex, int[] b,
int bFromIndex, int bToIndex)
```

Example :

```
public class ComparingArraysCompareDemo {
public static void main(String[] args) {
int[] i1 = {2, 4, 6, 8, 10};
int[] i2 = {2, 4, 6, 8, 10};
int[] i3 = {2, 4, 12, 8, 10};
System.out.println("i1: "+ Arrays.toString(i1));
System.out.println("i2: "+ Arrays.toString(i2));
System.out.println("i3: "+ Arrays.toString(i3));
// Comparing arrays lexicographically
System.out.println("\nArrays.compare(i1, i2): " + Arrays.compare(i1, i2));
System.out.println("Arrays.compare(i1, i3): " + Arrays.compare(i1, i3));
System.out.println("Arrays.compare(i3, i1): " + Arrays.compare(i3, i1));
// Comparing slicces of arrays lexicographically
System.out.println("\nArrays.compare(i1, 0, 3, i3, 0, 3): " +
Arrays.compare(i1, 0, 3, i3, 0, 3));
System.out.println("Arrays.compare(i1, 0, 3, i2, 0, 3): " +
Arrays.compare(i1, 0, 3, i2, 0, 3));
System.out.println("Arrays.compare(i3, 0, 3, i1, 0, 3): " +
Arrays.compare(i3, 0, 3, i1, 0, 3));
}
}
```

Output :

```
i1: [2, 4, 6, 8, 10]
i2: [2, 4, 6, 8, 10]
i3: [2, 4, 12, 8, 10]
```

```
Arrays.compare(i1, i2): 0
Arrays.compare(i1, i3): -1
Arrays.compare(i3, i1): 1
Arrays.compare(i1, 0, 3, i3, 0, 3): -1
Arrays.compare(i1, 0, 3, i2, 0, 3): 0
Arrays.compare(i3, 0, 3, i1, 0, 3): 1
```

1.3 Comparing arrays using **mismatch()** method

The group of methods **mismatch()** finds and returns the index of the first mismatch between two arrays. There are different overloads for `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short` and `Object` arrays. We can also specify the start and end index in both arrays for checking the mismatch.

1. `mismatch()` returns -1 if there is no mismatch.
2. If either array is `null`, it throws a `NullPointerException`.

```
int mismatch(int[] a, int[] b)
int mismatch (int[] a, int aFromIndex, int aToIndex, int[] b,
int bFromIndex, int bToIndex)
```

Example :

```
public class ComparingArraysMismatchDemo {
public static void main(String[] args) {
int[] i1 = {2, 4, 6, 8, 10};
int[] i2 = {2, 4, 6, 8, 10};
int[] i3 = {2, 12, 6, 8, 10};
System.out.println("i1: "+ Arrays.toString(i1));
System.out.println("i2: "+ Arrays.toString(i2));
System.out.println("i3: "+ Arrays.toString(i3));
// Finding first mismatch
System.out.println("\nArrays.mismatch(i1, i2): " + Arrays.mismatch(i1, i2));
System.out.println("Arrays.mismatch(i1, i3): " + Arrays.mismatch(i1, i3));
System.out.println("Arrays.mismatch(i3, i1): " + Arrays.mismatch(i3, i1));
// Finding first mismatch in slices
System.out.println("\nArrays.mismatch(i1, 0, 3, i3, 0, 3): " +
Arrays.mismatch(i1, 0, 3, i3, 0, 3));
System.out.println("Arrays.mismatch(i1, 0, 3, i2, 0, 3): " +
Arrays.mismatch(i1, 0, 3, i2, 0, 3));
System.out.println("Arrays.mismatch(i3, 0, 3, i1, 0, 3): " +
Arrays.mismatch(i3, 0, 3, i1, 0, 3));
}
}
```

Output :

```
i1: [2, 4, 6, 8, 10]
i2: [2, 4, 6, 8, 10]
```

```
i3: [2, 12, 6, 8, 10]
Arrays.mismatch(i1, i2): -1
Arrays.mismatch(i1, i3): 1
Arrays.mismatch(i3, i1): 1
Arrays.mismatch(i1, 0, 3, i3, 0, 3): 1
Arrays.mismatch(i1, 0, 3, i2, 0, 3): -1
Arrays.mismatch(i3, 0, 3, i1, 0, 3): 1
```

Using **equals()** method :

`equals()` returns *true* if two arrays are equal to each other. There are two different version of different overloads for `boolean`, `byte`, `char`, `double`, `float`, `int`, `long`, `short` and `Object` arrays. Version one for equality check for two arrays and version two for equality check for slices of two arrays. **Two arrays are considered equal if both are null**.

Following are overloaded `equals()`s

```
boolean equals(int[] a, int[] b)
boolean equals(int[] a, int aFromIndex, int aToIndex, int[] b,
int bFromIndex, int bToIndex)
```

Example :

```
public class ComparingArraysEqualsDemo {
public static void main(String[] args) {
String[] s1 = {"Peter", "Gerhard", "Philip", "Satheesh", "Mike"};
String[] s2 = {"Peter", "Gerhard", "Philip", "Satheesh", "Mike"};
String[] s3 = {"Gerhard", "Mike", "Peter", "Philip", "Satheesh"};
System.out.println("s1: "+ Arrays.toString(s1));
System.out.println("s2: "+ Arrays.toString(s2));
System.out.println("s3: "+ Arrays.toString(s3));
// Comparing arrays for equalitt
System.out.println("\nArrays.equals(s1, s2): " + Arrays.equals(s1, s2));
System.out.println("Arrays.equals(s1, s3): " + Arrays.equals(s1, s3));
// Comparing slicces of arrays for equality
System.out.println("\nArrays.equals(s1, 0, 3, s3, 0, 3): " +
Arrays.equals(s1, 0, 3, s3, 0, 3));
System.out.println("Arrays.equals(s1, 0, 3, s2, 0, 3): " +
Arrays.equals(s1, 0, 3, s2, 0, 3));
}
```

Output :

```
s1: [Peter, Gerhard, Philip, Satheesh, Mike]
s2: [Peter, Gerhard, Philip, Satheesh, Mike]
s3: [Gerhard, Mike, Peter, Philip, Satheesh]
Arrays.equals(s1, s2): true
Arrays.equals(s1, s3): false
Arrays.equals(s1, 0, 3, s3, 0, 3): false
Arrays.equals(s1, 0, 3, s2, 0, 3): true
```