***Arrays.sort() in Java***

Arrays.sort() works for arrays of both primitive data type and non-primitive datatype. It is used to sort the elements present in the specified array in a natural order of ascending order.  
  
**Type 1:** Arrays.sort(arr) for primitive types.  
**Example:** Working of Arrays.sort in a primitive data type.  
java

// Java program to sort an array

// using Arrays.sort()

import java.util.Arrays;

public class Test

{

public static void main(String[] args)

{

// Our arr contains 8 elements

int[] arr1 = {5, 20, 12, 30};

char[] arr2 = {'B', 'B', 'A', 'C', 'A'};

// Sorting the first array

Arrays.sort(arr1);

// Displaying the first array

System.out.println(Arrays.toString(arr1));

// Sorting the second array

Arrays.sort(arr2);

// Displaying the second array

System.out.println(Arrays.toString(arr2));

}

}

**Output:**

[5, 12, 20, 30]

[A, A, B, B, C]

**Note:** The primitive types cannot take a comparator and follows the natural non-decreasing order of sorting but for non-primitive types, a comparator can be used.  
  
**Type 2:**We can also use sort() to sort a subarray of arr[].  
**Syntax:**

public static void sort(int[] arr, int from\_Index, int to\_Index)

**Parameters:**

* arr - the array to be sorted
* from\_Index - the index of the first element, inclusive, to be sorted
* to\_Index - the index of the last element, exclusive, to be sorted

**Example:**  
java

// A sample Java program to sort a subarray

// using Arrays.sort().

import java.util.Arrays;

public class SortExample

{

public static void main(String[] args)

{

int[] arr = {5, 30, 20, 10, 8};

// Sort subarray from index 1 to 3, i.e.,

// only sort subarray {30, 20, 10} and

// keep other elements as it is.

Arrays.sort(arr, 1, 4);

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

}

}

**Output:**

[5, 10, 20, 30, 8]

**Sorting an array of Non-Primitive types by implementing Comparable interface.**  
  
**Example:** Sorting the arrays in increasing order of x-coordinate.  
java

// A sample Java program to implementing

// Comparable alongside Arrays.sort().

import java.util.\*;

import java.lang.\*;

import java.io.\*;

// A user-defined Point class implementing

// Comparable interface

class Point implements Comparable<Point>

{

int x, y;

// Costructor intialising x & y

Point(int x, int y)

{

this.x = x;

this.y = y;

}

// compareTo() function defining the

// nature of sorting i.e., according to

// x-coordinate

public int compareTo(Point P)

{

// Comparing two objects by

// Subtracting the passed object

// from current object

return this.x - P.x;

}

}

// Main class

class Test

{

public static void main(String[] args)

{

// Array of 3 objects

Point arr[] = {

new Point(10, 20),

new Point(3, 12),

new Point(5, 7) };

// Sorting the object containing array

Arrays.sort(arr);

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

System.out.println(

arr[i].x + " " + arr[i].y);

}

}

**Output:**

3 12

5 7

10 20

**Note:** If instead of Arrays.sort(arr); we write Arrays.sort(arr, Collections.reverse(arr)); then we get the arrays sorted in reverse order.  
  
**Sorting an array of Non-Primitive types by implementing Comparator interface.**  
**Example:** Sorting the arrays in increasing order of x-coordinate.  
java

// A sample Java program to implementing

// Comparator alongside Arrays.sort().

import java.util.\*;

import java.lang.\*;

import java.io.\*;

// Point class which does not implement

// Comparable interface. Thus the objects

// of this class are not comparable.

class Point

{

int x, y;

Point(int x, int y)

{

this.x = x;

this.y = y;

}

}

// This class implements

// Comparator interface to compare

class MyCmp implements Comparator<Point>

{

// Sorts the Point objects according

// to x-coordinates in natural order

public int compare(Point p1, Point p2)

{

return p1.x - p2.x;

}

}

// Main class

class Test

{

public static void main(String[] args)

{

// Array of 3 objects

Point arr[] = {

new Point(10, 20),

new Point(3, 12),

new Point(5, 7) };

// Sorting the object containing the array

// by passing the array and object MyCmp

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

// Displaying the sorted array

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

System.out.println(

arr[i].x + " " + arr[i].y);

}

}

**Output:**

3 12

5 7

10 20

**Reversing sorting an array using a Wrapper Class which are of Non-Primitive types.**  
**Example:**  
java

// A sample Java program to sort a subarray

// in descending order using Arrays.sort().

import java.util.Arrays;

import java.util.Collections;

public class SortExample

{

public static void main(String[] args)

{

Integer[] arr = {5, 20, 10, 12};

// Note that we have Integer here instead of

// int[] as Collections.reverseOrder doesn't

// work for primitive types.

Arrays.sort(arr, Collections.reverseOrder());

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

}

}

**Output:**

[20, 12, 10, 5]

**Problem:**Sort an Integer array in such a way that all even numbers come first followed by all the odd numbers.  
java

// A sample Java program to sort a subarray

// using Arrays.sort().

import java.util.\*;

import java.lang.\*;

import java.io.\*;

// This class implements

// Comparator interface to compare

class MyCmp implements Comparator<Integer>

{

// Sorts the Integers

public int compare(Integer a, Integer b)

{

return a%2 - b%2;

}

}

// Main class

class Test

{

public static void main(String[] args)

{

// Integer array

Integer[] arr = {5, 20, 10, 3, 12};

// Sorting arrays by passing

// arr and MyCmp object

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

// Displaying the sorted array

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

}

}

**Output:**

[20, 10, 12, 5, 3]

**Note:** The order is maintained as non-primitive types(Integer) guarantee the stability of the sorting algorithm which is based on MergeSort.