## How to override compareTo Method in Java?

As we know, there are basically two types of sorting technique in Java:

* First is internal sorting i.e that uses predefined sorting method ascending order Arrays.sort() for Primitive class arrays and wrapper class arrays and  Collections.sort() for collections both methods sort the elements in ascending order.
* The second technique is for sorting the elements is using the comparator or comparable interface in a class.
  + Comparator Interface: Implement the comparator interface in the class and override compare() method or pass the new comparator as the second argument in the sorting methods and change the sorting order according to the requirements. Comparator only works for wrapper type arrays and for collections like vector, ArrayList, etc.
  + Comparable Interface: This interface implements a single sorting technique, and it affects the whole class. The comparable interface provides a compareTo() method to sort the elements.

To summarize, in Java, if sorting of objects needs to be based on natural order then use the compareTo() method of Comparable Interface. For Integers default natural sorting order is ascending and for Strings it is alphabetical. Whereas, if you’re sorting needs to be done on attributes of different objects, or customized sorting then use compare() of Comparator Interface.

To summarize, in Java, if sorting of objects needs to be based on natural order then use the compareTo() method of Comparable Interface. For Integers default natural sorting order is ascending and for Strings it is alphabetical. Whereas, if you’re sorting needs to be done on attributes of different objects, or customized sorting then use compare() of Comparator Interface.

## Overriding of the compareTo () Method

In order to change the sorting of the objects according to the need of operation first, we have to implement a Comparable interface in the class and override the compareTo() method.

Since we have to sort the array of objects, traditional array.sort() method will not work, as it used to work on primitive types,.

So when we call the ***Arrays.sort()*** method and pass the object array, it will search, whether we have overridden the compareTo() method or not.

Since we have overridden the compareTo() method, so objects will be compared by using this compareTo() methods, based on the age.

The question was this “How you will sort Employee object based on his EmployeeID and his name” and this involves the use of both Comparable as well as Comparator interface in Java  
  
**Comparators and comparable** in Java are two interfaces that are used to implement sorting in Java.   
  
It’s often required to sort objects stored in any collection classes like ArrayList, HashSet, or in Array, and at that time we need to use either compare() or compareTo() method defined in java.util.Comparator and java.lang.Comparable. 

## Comparator vs Comparable in Java

1) Comparator in Java is defined in java.util package while Comparable interface in Java is defined in java.lang package, which very much says that Comparator should be used as a utility to sort objects which Comparable should be provided by default.

2) Comparator interface in Java has method public int compare (Object o1, Object o2) which returns a negative integer, zero or a positive integer as the first argument is less than, equal to, or greater than the second. While Comparable interface has method public int compareTo(Object o) which returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

3) Comparable in Java is used to implement the **natural ordering of objects**. In Java API String, Date and wrapper classes implement a Comparable interface.Its always good practice to override compareTo() for value objects.

4) If any class implements Comparable interface in Java then collection of that object either [List](http://javarevisited.blogspot.sg/2012/04/difference-between-list-and-set-in-java.html) or Array can be sorted automatically by using  Collections.sort() or Arrays.sort() method and object will be sorted based on there natural order defined by CompareTo method.

5) Objects which implement *Comparable in Java*  can be used as keys in a SortedMap like TreeMap or elements in a SortedSet  for example TreeSet, without specifying any Comparator.

These were combination of some theoretical and practical differences between Comparator and Comparator interface in Java. It does help you to decide when to use Comparator vs Comparable but things will be more clear when we some best practices around using both of these interfaces. Now let’s see an example of Comparator in Java:

## When to use Comparator and Comparable in Java

At last let’s see some best practices and recommendation on when to use Comparator or Comparable in Java:

1) If there is a natural or default way of sorting Object already exist during the development of Class then use Comparable. This is intuitive and you are given the class name people should be able to guess it correctly like Strings are sorted chronically, Employee can be sorted by their Id etc.

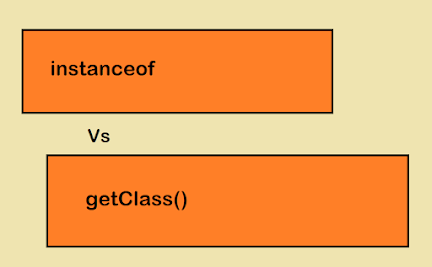
On the other hand, if an Object can be sorted on multiple ways and client is specifying on which parameter sorting should take place then use the Comparator interface. for example, Employee can again be sorted on name, salary or department and clients needs an API to do that. Comparator implementation can sort out this problem.

2) Some time you write code to sort objects of a class for which you are not the original author, or you don't have access to code. In these cases you can not implement Comparable and Comparator is only way to sort those objects.

3) Beware with the fact that How those objects will behave if stored in SorteSet or SortedMap like TreeSet and TreeMap. If an object doesn't implement Comparable then while putting them into SortedMap, always provided a corresponding Comparator which can provide sorting logic.

4) Order of comparison is very important while implementing the Comparable or Comparator interface. for example, if you are sorting objects based upon name then you can compare first name or last name on any order, so decide it judiciously.

## Java - Difference between getClass() and instanceof in equals() method? The key difference between them is that getClass() only returns true if the object is actually an instance of the specified class but an instanceof operator can return true even if the object is a subclass of a specified class or interface in Java.

**boolean** result = (loader.getClass() == Thread.class); // true result = (loader.getClass() == Runnable.class); // false because we are testing against Runnable result = loader **instanceof** Thread; // true because the loader is an object of Thread class result = loader **instanceof** Runnable; // true because the loader is an instance of Thread // and it implements Runnable  
  
So, you can see that getClass() put a restriction that objects are only equal to other objects of the same class, the same runtime type, but instanceof operator returns true for subclass as well.  
  
Due to this restriction, many Java developers including the great **Joshua Bloch have recommended using the instanceof operator in equals() method**.  
  


The [instanceof operator](https://javarevisited.blogspot.com/2015/12/10-points-about-instanceof-operator-in-java-example.html) lets you implement equality between super class and sub class. This is very important from the Collections framework perspective which uses the equals() method to find values. If you use the instanceof operator in equals() method then you can retrieve values even with the object of the subclass as a key provided they have the same content, but this is not possible when you use the getClass() method.