Difference between Comparable and Comparator

|  |  |
| --- | --- |
| **Comparable** | **Comparator (Remember T in util class )** |
| 1) Comparable provides **single sorting sequence**. In other words, we can sort the collection on the basis of single element such as id or name or price etc. | Comparator provides **multiple sorting sequence**. In other words, we can sort the collection on the basis of multiple elements such as id, name and price etc. |
| 2) Comparable **affects the original class** i.e. actual class is modified. | Comparator **doesn't affect the original class** i.e. actual class is not modified. |
| 3) Comparable provides public int **compareTo(arg1) method** to sort elements. | Comparator provides public int **compare(arg1,arg2) method** to sort elements. |
| 4) Comparable is found in **java.lang** package. | Comparator is found in **java.util** package. |
| 5) We can sort the list elements of Comparable type by **Collections.sort(List)** method. | We can sort the list elements of Comparator type by **Collections.sort(List,Comparator)** method. |

**Interview question in persistent:**

Is it possible other value in return , why its 0 , -1 and 1

|  |  |
| --- | --- |
| **//ascending order age Comparator**  **class** AgeComparator **implements** Comparator{  **public** **int** compare(Object o1,Object o2){  Student s1=(Student)o1;  Student s2=(Student)o2;  **if**(s1.age==s2.age)  **return** 0;  **else** **if**(s1.age>s2.age)  **return** 1;  **// any +ve number 90 or 91**  **else**  **return** -1; **//// any -ve number -90 or** -91  }  } | //can be writtern **class** AgeComparator **implements** Comparator{  **public** **int** compare(Object o1,Object o2){  Student s1=(Student)o1;  Student s2=(Student)o2;  **return** s1.age - s2.age;  }  } |

# How to Sort HashMap in Java based on Keys and Values

HashMap is not meant to keep entries in sorted order, but if you have to sort HashMap based upon keys or values, you can do that in Java. Sorting HashMap on keys is quite easy, all you need to do is to create a TreeMap by copying entries from HashMap. TreeMap is an implementation of SortedMap and keeps keys in their natural order or a custom order specified by Comparator provided while creating TreeMap. This means you can process entries of HashMap in a sorted order but you cannot pass a HashMap containing mappings in a specific order, this is just not possible because [HashMap doesn't guarantee any ordering](http://java67.blogspot.sg/2012/08/difference-between-hashmap-and-LinkedHashMap-Java.html" \t "_blank). On other hand, sorting HashMap by values is rather complex because there is no direct method to support that operation. You need to write code for that. In order to sort HashMap by values you can first create a Comparator, which can compare two entries based on values. Then get the Set of entries from Map, convert Set to List and use Collections.sort(List) method to sort your list of entries by values by passing your customized value comparator. This is similar of [how you sort an ArrayList in Java](http://java67.blogspot.sg/2012/08/how-to-sort-arraylist-in-java-list.html" \t "_blank). Half of the job is done by now. Now create a new LinkedHashMap and add sorted entries into that. Since LinkedHashMap guarantees insertion order of mappings, you will finally have a Map where contents are sorted by values.

### Steps to sort HashMap by values

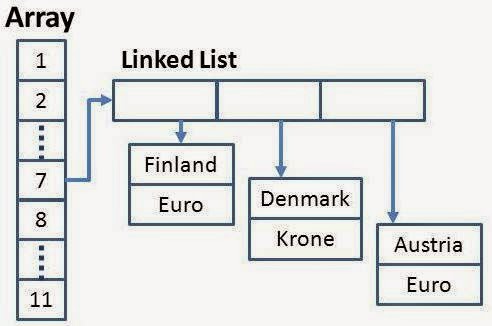
One difference between sorting HashMap by keys and values is that it can contain duplicate values by not duplicate keys. You cannot use TreeMap here because it only sort entries by keys. In this case you need to :

1. Get all entries by calling entrySet() method of Map
2. Create a [custom Comparator](http://java67.blogspot.sg/2014/11/java-8-comparator-example-using-lambda-expression.html" \t "_blank) to sort entries based upon values
3. Convert entry set to list
4. Sort entry list by using Collections.sort() method by passing your value comparator
5. Create a LinkedHashMap by adding entries in sorted order. Bcz it preserve insertion order

### Steps to sort HashMap by keys

There are two ways to *sort HashMap by keys*, first by using TreeMap and second by using LinkedHashMap. If you want to sort using TreeMap then it's simple, just create a TreeMap by copying content of HashMap. On the other hand, if you want to create a LinkedHashMap then you first need to get key set, convert that Set to List, sort that List and then add them into LHM in same order. Remember[HashMap can contain one null key](http://java67.blogspot.sg/2012/08/5-difference-between-hashtable-hashmap-Java-collection.html" \t "_blank) but duplicate keys are not allowed.

## HashMap Sorting by Keys and Values in Java Example

[](http://1.bp.blogspot.com/-2_1Fvl7bp-w/VK08jaVUi4I/AAAAAAAACVU/VFzkP1YLnm8/s1600/hashmap+java+example.jpg)

Here is our sample Java program to sort a HashMap first by keys and then by values. This program is divided into two part, first part sorts HashMap by keys and second part sorts it by values. Second part is more tricky then first part as there is no native Map implementation which supports any order for values. In order to sort a HashMap by values we had to create our own Comparator implementation which compares each entries by values to arrange them in a particular order. You can see that our valueComparator [overrides comapre() method](http://java67.blogspot.sg/2013/04/example-of-overriding-equals-hashcode-compareTo-java-method.html" \t "_blank)and accepts two entries. Later it retrieves values from those entries and compare them and return result. Since there is no method in Java Collection API to sort Map, we need to use Collections.sort() method which accepts a List. This involves creating a temporary ArrayList with entries for sorting purpose and then again copying entries from sorted ArrayList to a new LinkedHashMap to keep them in sorted order. Finally we create a HashMap from that LinkedHashMap, which is what we needed.

Read more: [http://www.java67.com/2015/01/how-to-sort-hashmap-in-java-based-on.html#ixzz4t2qWDQjz](http://www.java67.com/2015/01/how-to-sort-hashmap-in-java-based-on.html" \l "ixzz4t2qWDQjz)