Sort Map by Keys Generic Version.

In previous post we saw how to sort [HashMap by keys](http://data-structure-learning.blogspot.com/2015/06/sort-hashmap-by-keys.html) and [values](http://data-structure-learning.blogspot.com/2015/06/sort-hashmap-by-values.html).

Now the reason of writing this post is that when we wrote those methods they were specific to types. Let’s say that we have below method.

/\*\*

\* Method 1

\* Uses sorting to sort the keys and then insert in LinkedHashMap.

\* \*/

**public** **static** Map<String, String> sortMap(Map<String, String> map) {

/\*\*

\* Define new ArrayList<String>

\* \*/

List<String> keys = **new** ArrayList<String>();

/\*\*

\* use addAll(..) method to insert entire keySet in ArrayList.

\* \*/

keys.addAll(map.keySet());

/\*\*

\* Sort the List.

\* \*/

Collections.*sort*(keys, Collections.*reverseOrder*());

/\*\*

\* Define new LinkedHashMap.

\* \*/

Map<String, String> newMap = **new** LinkedHashMap<String, String>();

/\*\*

\* Insert data in LinkedHashMap. The data will be in sorted order because

\* LinkedHashMap maintains the order of insertion.

\* \*/

**for** (String str : keys) {

newMap.put(str, map.get(str));

}

*displayMap*(newMap);

**return** newMap;

}

Now this method cannot be used to sort any other type of Map than Map of String, String as key-value. So what we can do is define a method that takes generic argument and works on generic types.

Let us try to build a method that does this task.

1. First, we will take List of Map.Entry<K, V>. Why K, V? Because we don’t know what type of K and V is.

List<Map.Entry<K, V>> list = **new** LinkedList<Map.Entry<K, V>>(

map.entrySet());

1. Now we write a Comparator of type Comparator<Map.Entry<K, V>>

Collections.*sort*(list, **new** Comparator<Map.Entry<K, V>>() {

@Override

**public** **int** compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

**return** **((Comparable) (o1).getKey()).compareTo((o2).getKey())**;

}

});

The content that is underlined and bold is a warning. It tells is that we should refer generic type. But its’ ok this will do the job.

1. Now we will just copy the values to LinkedHashMap.

Map<K, V> result = **new** LinkedHashMap<>();

**for** (Map.Entry<K, V> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

This method will do the job. Below is the input.

Map<String, Integer> map = **new** HashMap<String, Integer>();

map.put("Bran", 45);

map.put("Robert", 34);

map.put("Roose", 43);

map.put("Lynaa", 54);

map.put("Aegon", 39);

map.put("John", 12);

Below is the output. Sort by Key.

{Aegon=39, Bran=45, John=12, Lynaa=54, Robert=34, Roose=43}

Below is the entire code for the method.

**public** **static** <K, V> Map<K, V> sortByKey(Map<K, V> map) {

List<Map.Entry<K, V>> list = **new** LinkedList<Map.Entry<K, V>>(

map.entrySet());

Collections.*sort*(list, **new** Comparator<Map.Entry<K, V>>() {

@Override

**public** **int** compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

**return** **((Comparable) (o1).getKey()).compareTo((o2).getKey());**

}

});

Map<K, V> result = **new** LinkedHashMap<>();

**for** (Map.Entry<K, V> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

System.***out***.println(result);

**return** result;

}

Now, we need to take care of the warning. Warning is indicated in **Bold** and **Underline.**

The problem is the generic type was not inferred. So the generic type is to be inferred for the key K. So for this we will change the return type of the method as well as some of the code. Change is minor.

In previous method following the signature of method

**public** **static** **<K, V>** Map<K, V> sortByKey(Map<K, V> map) {

Look at the **<K, V>.** Now we will change this to better thing.

<K **extends** Comparable<? **super** K>, V>

Now what is this? Well we just inferred the generic type. How?

K **extends** Comparable<? **super** K – means that key K extends Comparable of all super types of K.

Why did we wrote Comparable? We are using Comparator for Collections.sort(..) – Because we are using compareTo(..) method in Comparator’s compare(..) method. Hence we need to infer type of Comparable.

So new method will look like this.

**public** **static** **<K extends Comparable<? super K>, V>** Map<K, V> sortByKey(Map<K, V> map) {

List<Map.Entry<K, V>> list = **new** LinkedList<Map.Entry<K, V>>(

map.entrySet());

Collections.*sort*(list, **new** Comparator<Map.Entry<K, V>>() {

@Override

**public** **int** compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

**return o1.getKey().compareTo((o2).getKey());**

}

});

Map<K, V> result = **new** LinkedHashMap<>();

**for** (Map.Entry<K, V> entry : list) {

result.put(entry.getKey(), entry.getValue());

}

System.***out***.println(result);

**return** result;

}

Content in bold is changed.