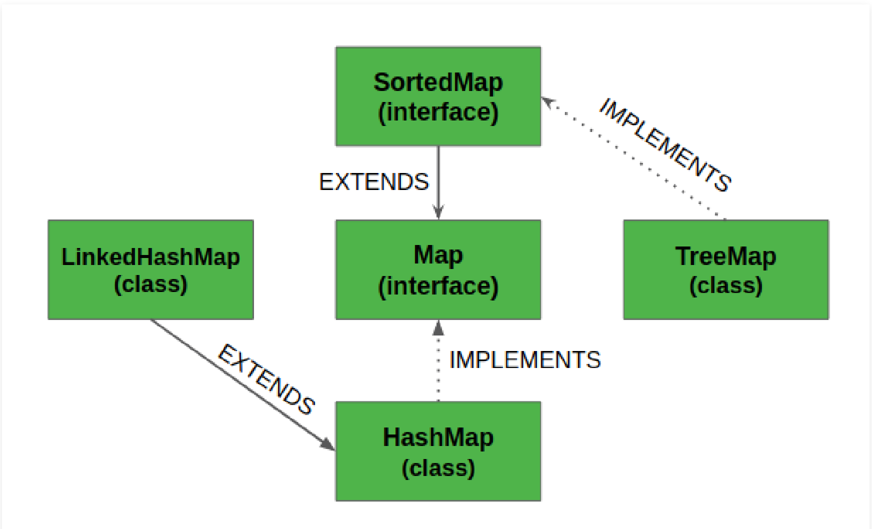
SortedMap Interface in Java with Examples

Difficulty Level : Medium Last Updated : 20 Aug, 2020

SortedMap is an interface in the collection framework. This interface extends the Map interface and provides a total ordering of its elements (elements can be traversed in sorted order of keys). The class that implements this interface is TreeMap.



MAP Hierarchy in Java

The main characteristic of a SortedMap is that it orders the keys by their natural ordering, or by a specified comparator. So consider using a TreeMap when you want a map that satisfies the following criteria:

- null key or null value is not permitted.
- The keys are sorted either by natural ordering or by a specified comparator.

Type Parameters:

- K – the type of keys maintained by this map
- V – the type of mapped values

The parent interface of SortedMap is Map<K, V>.

The subInterfaces of SortedMap are ConcurrentNavigableMap<K, V>, NavigableMap<K, V>.

SortedMap is implemented by ConcurrentSkipListMap, TreeMap.

Declaration:

```
public interface SortedMap<K, V> extends Map<K, V>
{
    Comparator comparator();
    SortedMap subMap(K fromKey, K toKey);
    SortedMap headMap(K toKey);
    SortedMap tailMap(K fromKey);
    K firstKey();
    K lastKey();
}
```

Example:

```
// Java code to demonstrate SortedMap Interface
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.SortedMap;
import java.util.TreeMap;

public class SortedMapExample {
    public static void main(String[] args)
    {
        SortedMap<Integer, String> sm
            = new TreeMap<Integer, String>();
        sm.put(new Integer(2), "practice");
        sm.put(new Integer(3), "quiz");
        sm.put(new Integer(5), "code");
        sm.put(new Integer(4), "contribute");
        sm.put(new Integer(1), "geeksforgeeks");
        Set s = sm.entrySet();

        // Using iterator in SortedMap
        Iterator i = s.iterator();

        // Traversing map. Note that the traversal
        // produced sorted (by keys) output .
        while (i.hasNext()) {
            Map.Entry m = (Map.Entry)i.next();

            int key = (Integer)m.getKey();
            String value = (String)m.getValue();

            System.out.println("Key : " + key
                               + "  value : " + value);
        }
    }
}
```

Output:

```
Key : 1  value : geeksforgeeks
Key : 2  value : practice
Key : 3  value : quiz
Key : 4  value : contribute
Key : 5  value : code
```

Creating SortedMap Objects

Since SortedMap is an interface, objects cannot be created of the type SortedMap. We always need a class that extends this list in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the SortedMap. This type-safe map can be defined as:

// Obj1, Obj2 are the type of the object to be stored in SortedMap

SortedMap<Obj1, Obj2> set = new TreeMap<Obj1, Obj2> ();

Performing Various Operations on SortedMap

Since SortedMap is an interface, it can be used only with a class that implements this interface. **TreeMap** is the class that implements the SortedMap interface. Now, let's see how to perform a few frequently used operations on the TreeMap.

1. Adding Elements: In order to add an element to the SortedMap, we can use the **put()** method. However, the insertion order is not retained in the TreeMap. Internally, for every element, the keys are compared and sorted in the ascending order.

```
// Java program add the elements in the SortedMap
import java.io.*;
import java.util.*;
class GFG {

    // Main Method
    public static void main(String args[])
    {
        // Default Initialization of a
        // SortedMap
        SortedMap tm1 = new TreeMap();

        // Initialization of a SortedMap
        // using Generics
        SortedMap<Integer, String> tm2
            = new TreeMap<Integer, String>();

        // Inserting the Elements
        tm1.put(3, "Geeks");
        tm1.put(2, "For");
        tm1.put(1, "Geeks");

        tm2.put(new Integer(3), "Geeks");
        tm2.put(new Integer(2), "For");
        tm2.put(new Integer(1), "Geeks");

        System.out.println(tm1);
        System.out.println(tm2);
    }
}
```

Output:

```
{1=Geeks, 2=For, 3=Geeks}
{1=Geeks, 2=For, 3=Geeks}
```

2. Changing Elements: After adding the elements if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the SortedMap are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

```
// Java program to change
// the elements in SortedMap
import java.io.*;
import java.util.*;
class GFG {

      // Main Method
    public static void main(String args[])
    {
        // Initialization of a SortedMap
        // using Generics
        SortedMap<Integer, String> tm
            = new TreeMap<Integer, String>();

        // Inserting the Elements
        tm.put(3, "Geeks");
        tm.put(2, "Geeks");
        tm.put(1, "Geeks");

        System.out.println(tm);

        tm.put(2, "For");

        System.out.println(tm);
    }
}
```

Output:

```
{1=Geeks, 2=Geeks, 3=Geeks}
{1=Geeks, 2=For, 3=Geeks}
```

3. Removing Element: In order to remove an element from the SortedMap, we can use the **remove()** method. This method takes the key value and removes the mapping for the key from this SortedMap if it is present in the map.

```
// Java program to remove the
// elements from SortedMap
import java.io.*;
import java.util.*;

class GFG {
      // Main Method
    public static void main(String args[])
    {
        // Initialization of a SortedMap
        // using Generics
        SortedMap<Integer, String> tm
            = new TreeMap<Integer, String>();

        // Inserting the Elements
        tm.put(3, "Geeks");
        tm.put(2, "Geeks");
        tm.put(1, "Geeks");
        tm.put(4, "For");

        System.out.println(tm);

        tm.remove(4);

        System.out.println(tm);
    }
}
```

Output:

```
{1=Geeks, 2=Geeks, 3=Geeks, 4=For}
{1=Geeks, 2=Geeks, 3=Geeks}
```

4. Iterating through the SortedMap: There are multiple ways to iterate through the Map. The most famous way is to use an enhanced for loop and get the keys. The value of the key is found by using the getValue() method.

```
// Java program to iterate through SortedMap
import java.util.*;

class GFG {

      // Main Method
    public static void main(String args[])
    {
        // Initialization of a SortedMap
        // using Generics
        SortedMap<Integer, String> tm
```

```
            = new TreeMap<Integer, String>();

        // Inserting the Elements
        tm.put(3, "Geeks");
        tm.put(2, "For");
        tm.put(1, "Geeks");

        for (Map.Entry mapElement : tm.entrySet()) {
            int key = (int)mapElement.getKey();

            // Finding the value
            String value = (String)mapElement.getValue();

            System.out.println(key + " : " + value);
        }
    }
}
```

Output:

```
1 : Geeks
2 : For
3 : Geeks
```

The class which implements the SortedMap interface is TreeMap.

TreeMap class which is implemented in the collections framework is an implementation of the SortedMap Interface and SortedMap extends Map Interface. It behaves like a simple map with the exception that it stores keys in a sorted format. TreeMap uses a tree data structure for storage. Objects are stored in sorted, ascending order. But we can also store in descending order by passing a comparator. Let’s see how to create a SortedMap object using this class.

```
// Java program to demonstrate the
// creation of SortedMap object using
// the TreeMap class

import java.util.*;

class GFG {
    public static void main(String[] args)
    {
        SortedMap<String, String> tm
            = new TreeMap<String, String>(new Comparator<String>() {
                  public int compare(String a, String b)
                  {
                      return b.compareTo(a);
                  }
              });

        // Adding elements into the TreeMap
        // using put()
        tm.put("India", "1");
        tm.put("Australia", "2");
        tm.put("South Africa", "3");

        // Displaying the TreeMap
        System.out.println(tm);

        // Removing items from TreeMap
        // using remove()
        tm.remove("Australia");
        System.out.println("Map after removing "
                           + "Australia:" + tm);
    }
}
```

Output:

```
{South Africa=3, India=1, Australia=2}
Map after removing Australia:{South Africa=3, India=1}
```

Methods of SortedMap Interface

METHOD	DESCRIPTION
comparator()	Returns the comparator used to order the keys in this map, or null if this map uses the natural ordering of its keys.
entrySet()	Returns a Set view of the mappings contained in this map.
firstKey()	Returns the first (lowest) key currently in this map.

METHOD	DESCRIPTION
headMap(K toKey)	Returns a view of the portion of this map whose keys are strictly less than toKey.
keySet()	Returns a Set view of the keys contained in this map.
lastKey()	Returns the last (highest) key currently in this map.
subMap(K fromKey, K toKey)	Returns a view of the portion of this map whose keys range from fromKey, inclusive, to toKey, exclusive.
tailMap(K fromKey)	Returns a view of the portion of this map whose keys are greater than or equal to fromKey.
values()	Returns a Collection view of the values contained in this map.

Methods inherited from interface java.util.Map

METHOD	DESCRIPTION
clear()	This method is used to clear and remove all of the elements or mappings from a specified Map collection.
containsKey(Object)	This method is used to check whether a particular key is being mapped into the Map or not. It takes the key element as a parameter and returns True if that element is mapped in the map.
containsValue(Object)	This method is used to check whether a particular value is being mapped by a single or more than one key in the Map. It takes the value as a parameter and returns True if that value is mapped by any of the key in the map.
entrySet()	This method is used to create a set out of the same elements contained in the map. It basically returns a set view of the map or we can create a new set and store the map elements into them.

METHOD	DESCRIPTION
equals(Object)	This method is used to check for equality between two maps. It verifies whether the elements of one map passed as a parameter is equal to the elements of this map or not.
get(Object)	This method is used to retrieve or fetch the value mapped by a particular key mentioned in the parameter. It returns NULL when the map contains no such mapping for the key.
hashCode()	This method is used to generate a hashCode for the given map containing key and values.
isEmpty()	This method is used to check if a map is having any entry for key and value pairs. If no mapping exists, then this returns true.
keySet()	This method is used to return a Set view of the keys contained in this map. The set is backed by the map, so changes to the map are reflected in the set, and vice-versa.
put(Object, Object)	This method is used to associate the specified value with the specified key in this map.
putAll(Map)	This method is used to copy all of the mappings from the specified map to this map.
remove(Object)	This method is used to remove the mapping for a key from this map if it is present in the map.
size()	This method is used to return the number of key/value pairs available in the map.
values()	This method is used to create a collection out of the values of the map. It basically returns a Collection view of the values in the HashMap.

This article is contributed by **Pratik Agarwal**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.