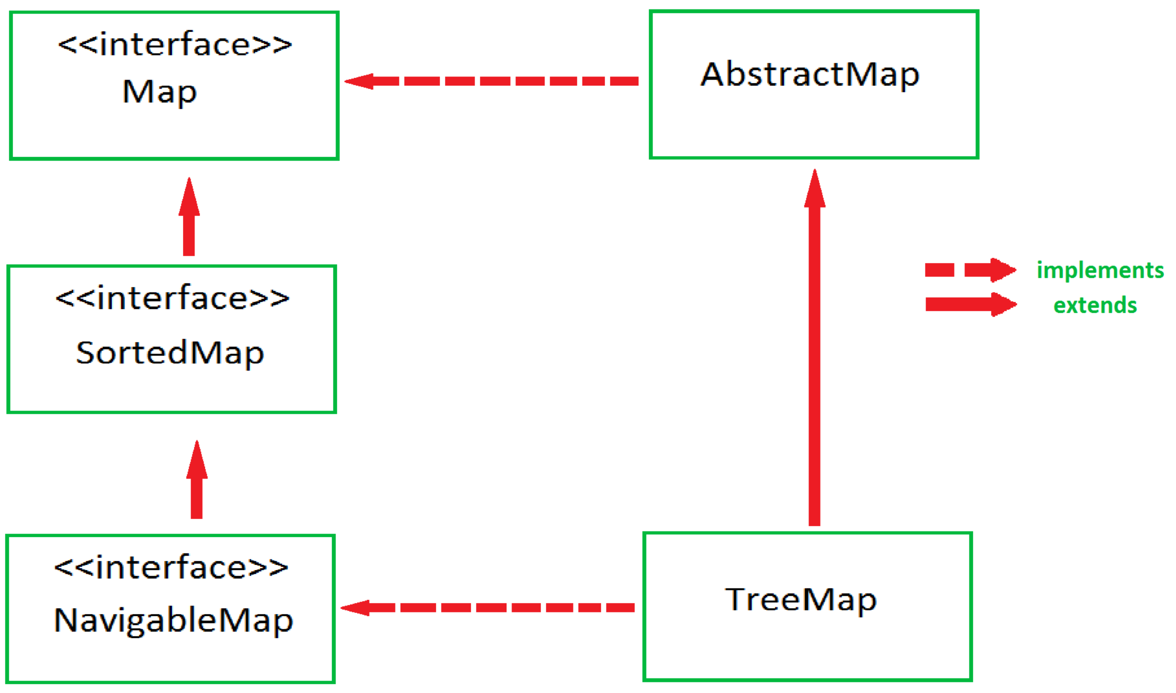
TreeMap in Java

The TreeMap in Java is used to implement Map interface and NavigableMap along with the Abstract Class. The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used. This proves to be an efficient way of sorting and storing the key-value pairs. The storing order maintained by the treemap must be consistent with equals just like any other sorted map, irrespective of the explicit comparators. The treemap implementation is not synchronized in the sense that if a map is accessed by multiple threads, concurrently and at least one of the threads modifies the map structurally, it must be synchronized externally. Some important features of the treemap are:

- This class is a member of Java Collections Framework.
- The class implements Map interfaces including NavigableMap, SortedMap and extends AbstractMap
- TreeMap in Java does not allow null keys (like Map) and thus a NullPointerException is thrown. However, multiple null values can be associated with different keys.
- All Map.Entry pairs returned by methods in this class and its views represent snapshots of mappings at the time they were produced. They do not support the Entry.setValue method.



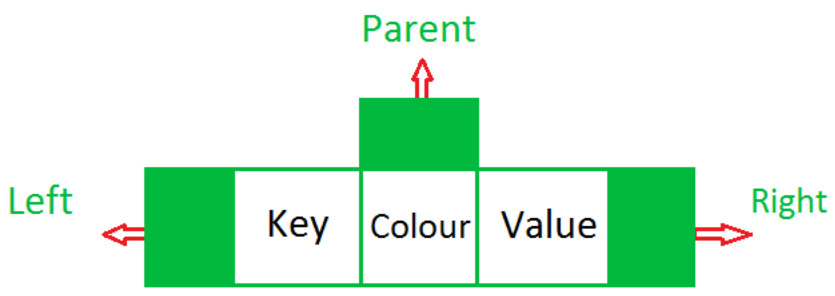
Performance factors:
TreeMap is not synchronized and thus is not thread-safe. For multithreaded environments, accidental unsynchronized access to the map is prevented by:

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

Internal structure: The methods in TreeMap while getting keyset and values, return Iterator that are fail-fast in nature, thus any concurrent modification will throw ConcurrentModificationException.

TreeMap is based upon tree data structure. Each node in the tree has,

- 3 Variables (*K key=Key, V value=Value, boolean color=Color*)
- 3 References (*Entry left = Left, Entry right = Right, Entry parent = Parent*)



Constructors in TreeMap:

1. **TreeMap() :** Constructs an empty tree map that will be sorted by using the natural order of its keys.
2. **TreeMap(Comparator comp) :** Constructs an empty tree-based map that will be sorted by using the Comparator comp.
3. **TreeMap(Map m) :** Initializes a tree map with the entries from m, which will be sorted by using the natural order of the keys.
4. **TreeMap(SortedMap sm) :** Initializes a tree map with the entries from sm, which will be sorted in the same order as sm.

Example:

```
import java.util.Comparator;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;

/*
 * @author Srinjoy Santra
*/
public class TreeMapImplementation {
    // May be replaced by an external class
    static class TreeCompare
           implements Comparator<String>
    {
        /* Compares keys based on the
           last word's natural ordering */
        public int compare(String a, String b)
        {
            int i,j,k;

            //Sorting by surnames
            i = a.lastIndexOf(' ');
            j = b.lastIndexOf(' ');
            k = a.substring(i).compareToIgnoreCase
                              (b.substring(j));
            if(k==0)
                return a.compareToIgnoreCase(b);
            else return k;
        }

    }
    public static void main(String[] args) {

        TreeMap<String, Double> tm = new
              TreeMap<>(new TreeCompare());

        tm.put("Head First Java", 807.34);
        tm.put("Java: A Beginners Guide 6th "+
                "Edition", 593.05);
        tm.put("Java: The Complete Reference"+
                 " 9th Edition", 531.31);
        tm.put("Core Java Volume I_Fundamentals"+
                " 9th Edition", 544.34);
        tm.put("Effective Java 2nd Edition", 373.70);

        // Values can be null
        tm.put("Java 8 in action", null);

        // Last entry with the same key
        // reflected in output
        tm.put("Java 8 in action", 539.65);

        Set<Map.Entry<String, Double>> set =
            tm.entrySet();
        for(Map.Entry<String,Double> me : set)
            System.out.println(me.getKey()+": Rs."
                               +me.getValue());

        tm.remove("Core Java Volume I_Fundamentals"+
                    " 9th Edition");
        System.out.println("...After removal of "+
                           "Core Java...");
        for(Map.Entry<String,Double> me : set)
            System.out.println(me.getKey()+": Rs."
                                +me.getValue());

    }
}
```

Output:

```
Java 8 in action: Rs.539.65
Core Java Volume I_Fundamentals 9th Edition: Rs.544.34
Effective Java 2nd Edition: Rs.373.7
Java: A Beginners Guide 6th Edition: Rs.593.05
Java: The Complete Reference 9th Edition: Rs.531.31
Head First Java: Rs.807.34
...After removal of Core Java...
Java 8 in action: Rs.539.65
Effective Java 2nd Edition: Rs.373.7
Java: A Beginners Guide 6th Edition: Rs.593.05
Java: The Complete Reference 9th Edition: Rs.531.31
Head First Java: Rs.807.34
```

Time Complexity: The algorithmic implementation is adapted from those of Red-Black Tree in Introduction to Algorithms (Eastern Economy Edition)

This provides guaranteed $log(n)$ time cost for the **containsKey, get, put and remove** operations.

Methods of TreeMap:

1. **boolean containsKey(Object key):** Returns true if this map contains a mapping for the specified key.
2. **boolean containsValue(Object value):** Returns true if this map maps one or more keys to the specified value.
3. **Object firstKey():** Returns the first (lowest) key currently in this sorted map.
4. **Object get(Object key):** Returns the value to which this map maps the specified key.
5. **Object lastKey():** Returns the last (highest) key currently in this sorted map.
6. **Object remove(Object key):** Removes the mapping for this key from this TreeMap if present.
7. **void putAll(Map map):** Copies all of the mappings from the specified map to this map.
8. **Set entrySet():** Returns a set view of the mappings contained in this map.
9. **int size():** Returns the number of key-value mappings in this map.
10. **Collection values():** Returns a collection view of the values contained in this map.
11. **Object clone():** The method returns a shallow copy of this TreeMap.
12. **void clear():** The method removes all mappings from this TreeMap and clears the map.
13. **SortedMap headMap(Object key_value):** The method returns a view of the portion of the map strictly less than the parameter key_value.
14. **Set keySet():** The method returns a Set view of the keys contained in the treemap.
15. **Object put(Object key, Object value):** The method is used to insert a mapping into a map

16. **SortedMap subMap((K startKey, K endKey):** The method returns the portion of this map whose keys range from startKey, inclusive, to endKey, exclusive.
17. **Object firstKey():** The method returns the first key currently in this tree map.

Reference:

- Official Documentation
- Java – The Complete Reference

Recommended Posts:

Java.util.TreeMap.containskey() and containsValue() in Java
Java.util.TreeMap.descendingMap() and descendingKeyset() in Java
Java.util.TreeMap.firstEntry() and firstKey() in Java
Java.util.TreeMap.pollFirstEntry() and pollLastEntry() in Java
Java.util.TreeMap.floorEntry() and floorKey() in Java
TreeMap put() Method in Java
HashMap and TreeMap in Java
TreeMap get() Method in Java
Java.util.TreeMap.put() and putAll() in Java
TreeMap lastKey() Method in Java
TreeMap headMap() Method in Java
TreeMap entrySet() Method in Java
TreeMap keySet() Method in Java
TreeMap floorKey() in Java with Examples
TreeMap lowerKey() in Java with Examples

srinjoy_santra
Check out this Author's contributed articles.

Article Tags : Java Java-Collections

Practice Tags : Java Java-Collections

4

3.3

☐ To-do ☐ Done

Based on **3** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article