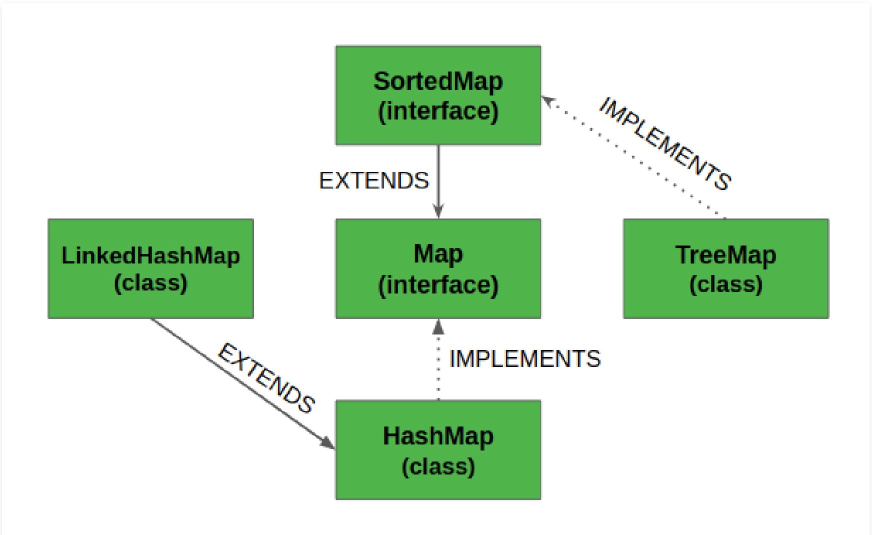
TreeMap in Java

Difficulty Level : Medium Last Updated : 18 Jan, 2022

The TreeMap in Java is used to implement Map interface and NavigableMap along with the AbstractMap 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.



MAP Hierarchy in Java

Features of a TreeMap

Some important features of the treemap are as follows:

1. This class is a member of the Java Collections Framework.
2. The class implements Map interfaces including NavigableMap, SortedMap, and extends AbstractMap class.
3. 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.
4. Entry pairs returned by the 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.

Now let us adhere forward and discuss Synchronized TreeMap. The implementation of a TreeMap is not synchronized. This means that if multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it must be synchronized externally. This is typically accomplished by using the Collections.synchronizedSortedMap method. This is best done at the creation time, to prevent accidental unsynchronized access to the set. This can be done as:

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

Geeks, now you must be wondering how does the TreemMap works internally?

The methods in a TreeMap while getting keyset and values, return an Iterator that is fail-fast in nature. Thus, any concurrent modification will throw ConcurrentModificationException. A TreeMap is based upon a red-black 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

In order to create a TreeMap, we need to create an object of the TreeMap class. The TreeMap class consists of various constructors that allow the possible creation of the TreeMap. The following are the constructors available in this class:

1. TreeMap()
2. TreeMap(Comparator comp)
3. TreeMap(Map M)
4. TreeMap(SortedMap sm)

Let us discuss them individually alongside implementing every constructor as follows:

Constructor 1: TreeMap()

This constructor is used to build an empty treemap that will be sorted by using the natural order of its keys.

Example

```
// Java Program to Demonstrate TreeMap
// using the Default Constructor

// Importing required classes
import java.util.*;
import java.util.concurrent.*;

// Main class
// TreeMapImplementation
public class GFG {

    // Method 1
    // To show TreeMap constructor
    static void Example1stConstructor()
    {
        // Creating an empty TreeMap
        TreeMap<Integer, String> tree_map
            = new TreeMap<Integer, String>();

        // Mapping string values to int keys
        // using put() method
        tree_map.put(10, "Geeks");
        tree_map.put(15, "4");
        tree_map.put(20, "Geeks");
        tree_map.put(25, "Welcomes");
        tree_map.put(30, "You");

        // Printing the elements of TreeMap
        System.out.println("TreeMap: " + tree_map);
    }

    // Method 2
    // Main driver method
    public static void main(String[] args)
    {
        System.out.println("TreeMap using "
                           + "TreeMap() constructor:\n");

        // Calling constructor
        Example1stConstructor();
    }
}
```

Output:

```
TreeMap using TreeMap() constructor:
TreeMap: {10=Geeks, 15=4, 20=Geeks, 25=Welcomes, 30=You}
```

Constructor 2: TreeMap(Comparator comp)

This constructor is used to build an empty TreeMap object in which the elements will need an external specification of the sorting order.

Example

```java
// Java Program to Demonstrate TreeMap
// using Comparator Constructor

// Importing required classes
import java.util.*;
import java.util.concurrent.*;

// Class 1
// Helper class representing Student
class Student {

    // Attributes of a student
    int rollno;
    String name, address;

    // Constructor
    public Student(int rollno, String name, String address)
    {

        // This keyword refers to current object itself
        this.rollno = rollno;
        this.name = name;
        this.address = address;
    }

    // Method of this class
    // To print student details
    public String toString()
    {
        return this.rollno + " " + this.name + " "
            + this.address;
    }
}

// Class 2
// Helper class - Comparator implementation
class Sortbyroll implements Comparator<Student> {

    // Used for sorting in ascending order of
    // roll number
    public int compare(Student a, Student b)
    {
        return a.rollno - b.rollno;
    }
}

// Class 3
// Main class
public class GFG {

    // Calling constructor inside main()
    static void Example2ndConstructor()
    {
        // Creating an empty TreeMap
        TreeMap<Student, Integer> tree_map
            = new TreeMap<Student, Integer>(
                new Sortbyroll());

        // Mapping string values to int keys
        tree_map.put(new Student(111, "bbbb", "london"), 2);
        tree_map.put(new Student(131, "aaaa", "nyc"), 3);
        tree_map.put(new Student(121, "cccc", "jaipur"), 1);

        // Printing the elements of TreeMap
        System.out.println("TreeMap: " + tree_map);
    }

    // Main driver method
    public static void main(String[] args)
    {

        System.out.println("TreeMap using "
                           + "TreeMap(Comparator)"
                           + " constructor:\n");
        Example2ndConstructor();
    }
}
```

Output:

```
TreeMap using TreeMap(Comparator) constructor:
TreeMap: {111 bbbb london=2, 121 cccc jaipur=1, 131 aaaa nyc=3}
```

Constructor 3: TreeMap(Map M)

This constructor is used to initialize a TreeMap with the entries from the given map M which will be sorted by using the natural order of the keys.

Example

```
}
```

```java
// Java Program to Demonstrate TreeMap
// using the Default Constructor

// Importing required classes
import java.util.*;
import java.util.concurrent.*;

// Main class
public class TreeMapImplementation {

    // Method 1
    // To illustrate constructor<Map>
    static void Example3rdConstructor()
    {
        // Creating an empty HashMap
        Map<Integer, String> hash_map
            = new HashMap<Integer, String>();

        // Mapping string values to int keys
        // using put() method
        hash_map.put(10, "Geeks");
        hash_map.put(15, "4");
        hash_map.put(20, "Geeks");
        hash_map.put(25, "Welcomes");
        hash_map.put(30, "You");

        // Creating the TreeMap using the Map
        TreeMap<Integer, String> tree_map
            = new TreeMap<Integer, String>(hash_map);

        // Printing the elements of TreeMap
        System.out.println("TreeMap: " + tree_map);
    }

    // Method 2
    // Main driver method
    public static void main(String[] args)
    {

        System.out.println("TreeMap using "
                           + "TreeMap(Map)"
                           + " constructor:\n");

        Example3rdConstructor();
    }
}
```

Output:

```
TreeMap using TreeMap(Map) constructor:
TreeMap: {10=Geeks, 15=4, 20=Geeks, 25=Welcomes, 30=You}
```

Constructor 4: TreeMap(SortedMap sm)

This constructor is used to initialize a TreeMap with the entries from the given sorted map which will be stored in the same order as the given sorted map.

Example

```java
// Java Program to Demonstrate TreeMap
// using the SortedMap Constructor

// Importing required classes
import java.util.*;
import java.util.concurrent.*;

// Main class
// TreeMapImplementation
public class GFG {

    // Method
    // To show TreeMap(SortedMap) constructor
    static void Example4thConstructor()
    {
        // Creating a SortedMap
        SortedMap<Integer, String> sorted_map
            = new ConcurrentSkipListMap<Integer, String>();

        // Mapping string values to int keys
        // using put() method
        sorted_map.put(10, "Geeks");
        sorted_map.put(15, "4");
        sorted_map.put(20, "Geeks");
        sorted_map.put(25, "Welcomes");
        sorted_map.put(30, "You");

        // Creating the TreeMap using the SortedMap
        TreeMap<Integer, String> tree_map
            = new TreeMap<Integer, String>(sorted_map);

        // Printing the elements of TreeMap
        System.out.println("TreeMap: " + tree_map);
    }
    // Method 2
    // Main driver method
    public static void main(String[] args)
    {

        System.out.println("TreeMap using "
                           + "TreeMap(SortedMap)"
                           + " constructor:\n");

        Example4thConstructor();
    }
}
```

Output:

```
TreeMap using TreeMap(SortedMap) constructor:
TreeMap: {10=Geeks, 15=4, 20=Geeks, 25=Welcomes, 30=You}
```

Methods in the TreeMap Class

Method	Action Performed
clear()	The method removes all mappings from this TreeMap and clears the map.
clone()	The method returns a shallow copy of this TreeMap.
containsKey(Object key)	Returns true if this map contains a mapping for the specified key.
containsValue(Object value)	Returns true if this map maps one or more keys to the specified value.
entrySet()	Returns a set view of the mappings contained in this map.
firstKey()	Returns the first (lowest) key currently in this sorted map.
get(Object key)	Returns the value to which this map maps the specified key.

Method	Action Performed
headMap(Object key_value)	The method returns a view of the portion of the map strictly less than the parameter key_value.
keySet()	The method returns a Set view of the keys contained in the treemap.
lastKey()	Returns the last (highest) key currently in this sorted map.
put(Object key, Object value)	The method is used to insert a mapping into a map.
putAll(Map map)	Copies all of the mappings from the specified map to this map.
remove(Object key)	Removes the mapping for this key from this TreeMap if present.
size()	Returns the number of key-value mappings in this map.
subMap((K startKey, K endKey)	The method returns the portion of this map whose keys range from startKey, inclusive, to endKey, exclusive.
values()	Returns a collection view of the values contained in this map.

Implementation: The following programs below will demonstrate better how to create, insert, and traverse through the TreeMap.

Illustration:

```java
// Java Program to Illustrate Operations in TreeMap
// Such as Creation, insertion
// searching, and traversal

// Importing required classes
import java.util.*;
import java.util.concurrent.*;

// Main class
// Implementation of TreeMap
public class GFG {
    // Declaring a TreeMap
    static TreeMap<Integer, String> tree_map;

    // Method 1
    // To create TreeMap
    static void create()
    {

        // Creating an empty TreeMap
        tree_map = new TreeMap<Integer, String>();

        // Display message only
        System.out.println("TreeMap successfully"
                           + " created");
    }

    // Method 2
    // To Insert values in the TreeMap
    static void insert()
    {

        // Mapping string values to int keys
        // using put() method
        tree_map.put(10, "Geeks");
        tree_map.put(15, "4");
        tree_map.put(20, "Geeks");
        tree_map.put(25, "Welcomes");
        tree_map.put(30, "You");

        // Display message only
        System.out.println("\nElements successfully"
                           + " inserted in the TreeMap");
    }

    // Method 3
    // To search a key in TreeMap
    static void search(int key)
    {

        // Checking for the key
        System.out.println("\nIs key \"" + key
                           + "\" present? "
                           + tree_map.containsKey(key));
    }

    // Method 4
    // To search a value in TreeMap
    static void search(String value)
    {

        // Checking for the value
        System.out.println("\nIs value \"" + value
                           + "\" present? "
                           + tree_map.containsValue(value));
    }
    // Method 5
    // To display the elements in TreeMap
    static void display()
    {

        // Displaying the TreeMap
        System.out.println("\nDisplaying the TreeMap:");

        System.out.println("TreeMap: " + tree_map);
    }

    // Method 6
    // To traverse TreeMap
    static void traverse()
    {

        // Display message only
        System.out.println("\nTraversing the TreeMap:");

        for (Map.Entry<Integer, String> e :
             tree_map.entrySet())
            System.out.println(e.getKey() + " "
                               + e.getValue());
    }

    // Method 6
    // Main driver method
    public static void main(String[] args)
    {

        // Calling above defined methods inside main()

        // Creating a TreeMap
        create();

        // Inserting the values in the TreeMap
        insert();

        // Search key "50" in the TreeMap
        search(50);

        // Search value "Geeks" in the TreeMap
        search("Geeks");

        // Display the elements in TreeMap
        display();

        // Traversing the TreeMap
        traverse();
    }
}
```

Output:

```
TreeMap successfully created
Elements successfully inserted in the TreeMap
Is key "50" present? false
Is value "Geeks" present? true
Displaying the TreeMap:
TreeMap: {10=Geeks, 15=4, 20=Geeks, 25=Welcomes, 30=You}
Traversing the TreeMap:
10 Geeks
15 4
20 Geeks
25 Welcomes
30 You
```

Performing Various Operations on TreeMap

After the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the TreeMap. Now, let's see how to perform a few frequently used operations on the TreeMap.

Operation 1: Adding Elements

In order to add an element to the TreeMap, 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 ascending order.

Example

```java
// Java Program to Illustrate Addition of Elements
// in TreeMap using put() Method

// Importing required classes
import java.util.*;

// Main class
class GFG {

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

        // Inserting the elements in TreeMap
        // using put() method
        tm1.put(3, "Geeks");
        tm1.put(2, "For");
        tm1.put(1, "Geeks");

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

        // Inserting the elements in TreeMap
        // again using put() method
        tm2.put(new Integer(3), "Geeks");
        tm2.put(new Integer(2), "For");
        tm2.put(new Integer(1), "Geeks");

        // Printing the elements of both TreeMaps

        // Map 1
        System.out.println(tm1);
        // Map 2
        System.out.println(tm2);
    }
}
```

Output:

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

Operation 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 treemap 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.

Example

```java
// Java program to Illustrate Updation of Elements
// in TreeMap using put() Method

// Importing required classes
import java.util.*;

// Main class
class GFG {

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

        // Inserting the elements in Map
        // using put() method
        tm.put(3, "Geeks");
        tm.put(2, "Geeks");
        tm.put(1, "Geeks");

        // Print all current elements in map
        System.out.println(tm);

        // Inserting the element at specified
        // corresponding to specified key
        tm.put(2, "For");

        // Printing the updated elements of Map
        System.out.println(tm);
    }
}
```

Output:

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

Operation 3: Removing Element

In order to remove an element from the TreeMap, we can use the remove() method. This method takes the key value and removes the mapping for the key from this treemap if it is present in the map.

Example

```java
// Java program to Illustrate Removal of Elements
// in TreeMap using remove() Method

// Importing required classes
import java.util.*;

// Main class
class GFG {

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

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

        // Printing all elements of Map
        System.out.println(tm);

        // Removing the element corresponding to key
        tm.remove(4);

        //  Printing updated TreeMap
        System.out.println(tm);
    }
}
```

Output:

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

Operation 4: Iterating through the TreeMap

There are multiple ways to iterate through the Map. The most famous way is to use a for-each loop and get the keys. The value of the key is found by using the *getValue() method*.

Example

```
// Java Program to Illustrate Iterating over TreeMap
// using

// Importing required classes
import java.util.*;

// Main class
class GFG {

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

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

        // For-each loop for traversal over Map
        // via entrySet() Method
        for (Map.Entry mapElement : tm.entrySet()) {

            int key = (int)mapElement.getKey();

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

            // Printing the key and value
            System.out.println(key + " : " + value);
        }
    }
}
```

Output:

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