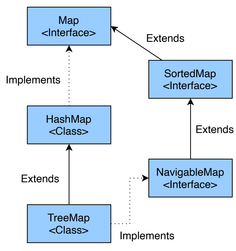
TreeMap is a class in the `java.utils` package that stores the keys in sorted order. Some of the features of **TreeMap** are:

1. The entries in **TreeMap** are sorted in the natural ordering of its keys.
2. It does not allow null keys, however there can be null values.
3. The TreeMap is not thread-safe, although it can be made thread-safe using the `synchronizedMap()` method of the **Collections** class.

Since a **TreeMap** stores the keys in sorted order, the objects that we are storing in the **TreeMap** should either implement the **Comparable** interface or we should pass a **Comparator** while creating the **TreeMap** object.



TreeMap hierarchy

Creating a TreeMap

There are four different ways to create a TreeMap object.

Using a no-arg constructor

A **TreeMap** can be created easily using the no-arg constructor. The keys that we will store in this **TreeMap** must implement the **Comparable** interface.

Using the constructor with Comparator as an argument

If the objects that we are storing in a **TreeMap** as a key do not implement the **Comparable** interface or if we need to store the keys in descending order, then we can provide a custom **Comparator** while creating the **TreeMap**. Now, when the keys are stored in the **TreeMap**, they are sorted as per the logic provided by the **Comparator**.

Using constructor with the argument of type Map

A **TreeMap** can be created from another Map as well. The keys are stored in ascending order irrespective of the order that the elements are stored in the provided Map.

Using constructor with the argument of type SortedMap

This constructor behaves as a copy constructor and creates a TreeMap with the same elements and ordering of the provided sorted map.

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

public class TreeMapDemo {

    public static void main(String args[]) {

        // Creating a TreeMap which will store all the elements in reverse order.
        TreeMap<String, Integer> reverseMap = new TreeMap<>(Comparator.reverseOrder());
        reverseMap.put("Oracle", 43);
        reverseMap.put("Microsoft", 56);
        reverseMap.put("Apple", 43);
        reverseMap.put("Novartis", 87);
        System.out.println("Elements are stored in reverse order: " + reverseMap);

        // Creating a HashMap which will store all the elements in random order.
        Map<String, Integer> hashMap = new HashMap<>();
        hashMap.put("Oracle", 43);
        hashMap.put("Microsoft", 56);
        hashMap.put("Apple", 43);
        hashMap.put("Novartis", 87);
        System.out.println("Elements are stored in random order: " + hashMap);

        // Creating a TreeMap using existing HashMap. This will store the elements in ascending order.
        TreeMap<String, Integer> treeMap1 = new TreeMap<>(hashMap);
        System.out.println("Elements are stored in ascending order: " + treeMap1);
```

Run Save Reset

Inserting elements in a TreeMap

Let's discuss all the methods that we can use to insert the key-value pairs in a **TreeMap**.

Using the `put()` method

We can use the `put(K key, V value)` method to insert a key-value pair in the **TreeMap**. If the key is not present, then a new key-value pair will be added. If the key is already present, then the value will be updated.

Using the `putAll()` method

The `putAll(Map<? extends K, ? extends V> m)` method copies all of the mappings from the specified map to this map. These mappings will replace any mappings that this map had for any of the keys currently in the specified map.

The below example shows a working example of **TreeMap**.

```
import java.util.TreeMap;

public class TreeMapDemo {

    public static void main(String args[]) {
        TreeMap<String, Integer> map = new TreeMap<>();

        map.put("Oracle", 43);
        map.put("Microsoft", 56);
        map.put("Apple", 43);
        map.put("Novartis", 87);

        System.out.println(map);

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

        map.put("Google", 65);
        map.put("Audi", 32);
        finalMap.putAll(map);

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

Run Save Reset