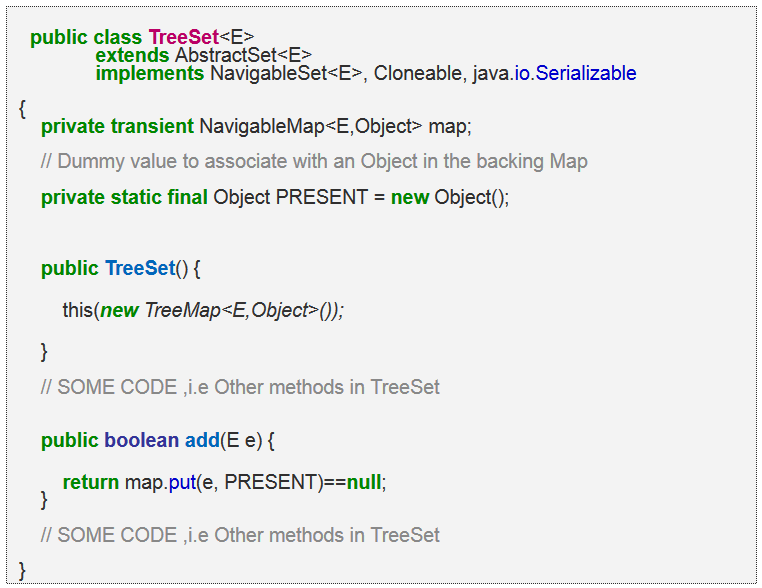
**What is TreeSet ?**  
TreeSet is like HashSet which contains the unique elements only but in a sorted manner. The major difference is that TreeSet provides a total ordering of the elements. The elements are ordered using their natural ordering, or by a Comparator typically provided at sorted set creation time. The set's iterator will traverse the set in ascending element order.  
  
**How TreeSet works in Java ?**

If you look into the TreeSet Api in rt.jar , you will find the following code :



Hence , whenever you are adding element to the TreeSet object , it [works just like HashSet](http://javahungry.blogspot.com/2013/08/how-sets-are-implemented-internally-in.html) , The only difference is that instead of HashMap here we have TreeMap object in the constructor.

As we know in TreeMap each key is unique as it internally uses HashMap . So what we do in the TreeSet is that we pass the argument in the add(Elemene E) that is E as a key in the TreeSet . Now we need to associate some value to the key , so what Java apis developer did is to pass the Dummy  value that is ( new Object () ) which is referred by Object reference PRESENT .

So , actually when you are adding a line in TreeSet like  treeset.add(3)   what java does internally is that it will put that element E here 3 as a key in the TreeMap(created during TreeSet object creation) and some dummy value that is Object's object is passed as a value to the key .  
  
Now if you see the code of the TreeMap put(Key k,Value V) method , you will find something like this  
  
 public V put(K key, V value) {  
//Some code  
}  
  
The main point to notice in above code is that put (key,value) will return  
  
1.  null , if key is unique and added to the map  
2.  Old Value of the key , if key is duplicate  
  
So , in TreeSet add() method ,  we check the return value of map.put(key,value) method with null value  
i.e.  
  
   public boolean add(E e) {  
            return map.put(e, PRESENT)==null;  
       }  
  
So , if map.put(key,value) returns null ,then  
map.put(e, PRESENT)==null      will return true and element is added to the TreeSet.  
  
  
  
So , if map.put(key,value) returns old value of the key ,then  
map.put(e, PRESENT)==null      will return false and element is  not added to the TreeSet .

**How to find the index of  any element in the TreeSet ?**  
  
There are many ways to find out the index of element in the TreeSet. Below is the one liner :  
  
set.headSet(element).size()  
  
  
**Note  :**  headSet(element) method returns the sub TreeSet(portion of TreeSet) whose values  are less than input element. Then we are calling size() method on the sub TreeSet , which returns the index of the element as sub TreeSet is already sorted.

**Why and when we use TreeSet ?**  
We prefer TreeSet in order  to maintain the unique elements  in the sorted order .  
 **What is the runtime performance of the add() method of the TreeSet and HashSet , where n represents the number of elements?**

TreeSet implementation provides guaranteed log(n) time cost for the basic operations (add, remove  
and contains ) method.

HashSet provides constant time performance O(1) for basic operations  (add, remove and contains) method assuming the hash  function disperses the elements properly among the buckets.  
 **One-liner :**TreeSet : O(log(n))  HashSet : O(1)

**What is natural ordering in TreeSet ?**  
  
"Natural" ordering is the ordering implied by the implementation of Comparable interface by the objects in the TreeSet . Essentially RBTree must be able to tell which object is smaller than other object , and there are two  ways to supply that logic to the RB Tree implementation :  
  
1. We need to implement the Comparable interface in the class(es) used as objects in TreeSet.  
2. Supply an implementation of the Comparator would do comparing outside the class itself.

**Which data structure you will prefer  in your code : HashSet and TreeSet ?**  
TreeSet contains the elements in the sorted order while HashSet is faster. Thus , deciding which one to choose depends upon the conditions :  
  
If you want to maintain the order of the elements then TreeSet should be used because the result is alphabetically sorted.  
  
If you do not want to sort the elements and  avoid duplicate elements . Your task involves mainly insert and retrieve operations then prefer HashSet.While iterating HashSet there is no ordering of elements while TreeSet iterates in the natural order.

**What happens if the TreeSet is concurrently modified while iterating the elements ?**  
The iterator's returned by the TreeSet class iterator method are fail-fast. Fail-fast means if the set is modified at any time after the iterator is created , in any way except the iterator's own remove method  , the iterator will throw a ConcurrentModificationException. Thus , in the face of concurrent modification , the iterator fails quickly and cleanly .

**What happens if the TreeSet is concurrently modified while iterating the elements ?**  
The iterator's returned by the TreeSet class iterator method are fail-fast.  fail-fast means if the set is modified at any time after the iterator is created , in any way except the iterator's own remove method  , the iterator will throw a ConcurrentModificationException. Thus , in the face of concurrent modification , the iterator fails quickly and cleanly .