TREESET

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.
* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quite fast.
* Java TreeSet class doesn't allow null elements.
* Java TreeSet class is non-synchronized.
* Java TreeSet class maintains ascending order.

INTERNAL WORKING

TreeSet is being implemented using a binary search tree, which is self-balancing just like a Red-Black Tree. Therefore, operations such as a search, remove, and add consume O(log(N)) time. The reason behind this is there in the self-balancing tree. It is there to ensure that the tree height never exceeds O(log(N)) for all of the mentioned operations. Therefore, it is one of the efficient data structures in order to keep the large data that is sorted and also to do operations on it.

SYNCRONIZATION

As already mentioned above, the TreeSet class is not synchronized. It means if more than one thread concurrently accesses a tree set, and one of the accessing threads modify it, then the synchronization must be done manually. It is usually done by doing some object synchronization that encapsulates the set. However, in the case where no such object is found, then the set must be wrapped with the help of the Collections.synchronizedSet() method. It is advised to use the method during creation time in order to avoid the unsynchronized access of the set. The following code snippet shows the same.

*TreeSet treeSet = new TreeSet();*

*Set syncrSet = Collections.synchronziedSet(treeSet);*

HIERARCHY

Application

Description automatically generated with low confidence

DECLARATION

*public class TreeSet<E> extends AbstractSet<E> implements NavigableSet<E>, Cloneable, Serializable*

CONSTRUCTORS

1. TreeSet()

It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.

1. TreeSet(Collection<? extends E> c)

It is used to build a new tree set that contains the elements of the collection c.

1. TreeSet(Comparator<? super E> comparator)

It is used to construct an empty tree set that will be sorted according to given comparator.

1. TreeSet(SortedSet<E> s)

It is used to build a TreeSet that contains the elements of the given SortedSet.

METHODS OF TREESET

* add(Object o) : This method will add the specified element according to the same sorting order mentioned during the creation of the TreeSet. Duplicate entries will not get added.
* addAll(Collection c): This method will add all elements of the specified Collection to the set. Elements in the Collection should be homogeneous otherwise ClassCastException will be thrown. Duplicate Entries of Collection will not be added to TreeSet.
* clear(): This method will remove all the elements.
* clone(): The method is used to return a shallow copy of the set, which is just a simple copied set.
* contains(): This method will return true if a given element is present in TreeSet else it will return false.
* descendingIterator(): This method returns an iterator over the elements in this set in descending order.
* isEmpty(): This method is used to return true if this set contains no elements or is empty and false for the opposite case.
* last(): This method will return the last element in TreeSet if TreeSet is not null else it will throw NoSuchElementException.
* size(): This method is used to return the size of the set or the number of elements present in the set.

SAMPLE PROGRAMS

1. SAMPLE WORKING OF TREESET

import java.util.\*;

class GFG {

public static void main(String[] args)

{

Set<String> ts1 = new TreeSet<>();

ts1.add("A");

ts1.add("B");

ts1.add("C");

ts1.add("C");

System.out.println(ts1);

}

}

OUTPUT:

Ajay

Ravi

Vijay

import java.util.\*;

class TreeSet2{

public static void main(String args[]){

TreeSet<String> set=new TreeSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ajay");

System.out.println("Traversing element through Iterator in descending order");

Iterator i=set.descendingIterator();

while(i.hasNext())

{

System.out.println(i.next());

}

}

}

OUTPUT

Traversing element through Iterator in descending order

Vijay

Ravi

Ajay

Traversing element through NavigableSet in descending order

Vijay

Ravi

Ajay

3.

import java.util.\*;

class TreeSet3{

public static void main(String args[]){

TreeSet<Integer> set=new TreeSet<Integer>();

set.add(24);

set.add(66);

set.add(12);

set.add(15);

System.out.println("Lowest Value: "+set.pollFirst());

System.out.println("Highest Value: "+set.pollLast());

}

}

OUTPUT

Lowest Value: 12

Highest Value: 66