Java Collection Framework Treeset

TreeSet class used to store unique elements in ascending order. It is similar to **HashSet** except that it sorts the elements in the ascending order while HashSet doesn’t maintain any order.

Java TreeSet class implements the Set interface and use tree based data structure storage. It extends AbstractSet class and implements the NavigableSet interface. The declaration of the class is given below.

TreeSet class Declaration

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

Important Points

1. It stores the elements in ascending order.
2. It uses a Tree structure to store elements.
3. It contains unique elements only like HashSet.
4. It's access and retrieval times are quite fast.
5. It doesn't allow null element.
6. It is non synchronized.

TreeSet Constructors

```
TreeSet()

TreeSet( Collection C )

TreeSet( Comparator comp )

TreeSet( SortedSet ss )
```

Copy

Example: Creating an TreeSet

Lets create a TreeSet to store string elements. Here set is empty because we did not add elements to it.

```
import java.util.*;

class Demo

{

  public static void main(String[] args)

   {
    // Creating an TreeSet

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


    // Displaying TreeSet

    System.out.println(fruits);

  }

}
```

Copy

```
[]
```

TreeSet Methods

Method	Description
boolean add(E e)	It adds the specified element to this set if it is not already present.
boolean addAll(Collection<? extends E> c)	It adds all of the elements in the specified collection to this set.
E ceiling(E e)	It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.
Comparator<? super E> comparator()	It returns comparator that arranged elements in order.
Iterator descendingIterator()	It is used iterate the elements in descending order.
NavigableSet descendingSet()	It returns the elements in reverse order.
E floor(E e)	It returns the equal or closest least element of the specified

	element from the set, or null there is no such element.
SortedSet headSet(E toElement)	It returns the group of elements that are less than the specified element.
NavigableSet headSet(E toElement, boolean inclusive)	It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element.
E higher(E e)	It returns the closest greatest element of the specified element from the set, or null there is no such element.
Iterator iterator()	It is used to iterate the elements in ascending order.
E lower(E e)	It returns the closest least element of the specified element from the set, or null there is no such element.
E pollFirst()	It is used to retrieve and remove the lowest(first) element.
E pollLast()	It is used to retrieve and remove the highest(last) element.
Spliterator spliterator()	It is used to create a late-binding and fail-fast spliterator over the elements.
NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	It returns a set of elements that lie between the given range.
SortedSet subSet(E fromElement, E toElement))	It returns a set of elements that lie between the given range

	which includes fromElement and excludes toElement.
SortedSet tailSet(E fromElement)	It returns a set of elements that are greater than or equal to the specified element.
NavigableSet tailSet(E fromElement, boolean inclusive)	It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element.
boolean contains(Object o)	It returns true if this set contains the specified element.
boolean isEmpty()	It returns true if this set contains no elements.
boolean remove(Object o)	It is used to remove the specified element from this set if it is present.
void clear()	It is used to remove all of the elements from this set.
Object clone()	It returns a shallow copy of this TreeSet instance.
E first()	It returns the first (lowest) element currently in this sorted set.
E last()	It returns the last (highest) element currently in this sorted set.
int size()	It returns the number of elements in this set.

Add Elements To TreeSet

Lets take an example to create a TreeSet that contains duplicate elements. But you can notice that it prints unique elements that means it does not allow duplicate elements.

```java
import java.util.*;

class Demo{

  public static void main(String args[]){

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

    al.add("Ravi");

    al.add("Vijay");

    al.add("Ravi");

    al.add("Ajay");


    Iterator itr=al.iterator();

    while(itr.hasNext()){

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

    }

  }

}
```

Copy

```
Ajay

Ravi

Vijay
```

Removing Elements From the TreeSet

We can use `remove()` method of this class to remove the elements. See the below example.

```java
import java.util.*;

class Demo{

  public static void main(String args[]){

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

    al.add("Ravi");

    al.add("Vijay");

    al.add("Ravi");

    al.add("Ajay");
    Iterator itr=al.iterator();
    while(itr.hasNext()){
      System.out.println(itr.next());
    }

    al.remove("Ravi");
    System.out.println("After Removing: "+al);
  }
}
```

Copy

```
Ajay
Ravi
Vijay
After Removing: [Ajay, Vijay]
```

Search an Element in TreeSet

TreeSet provides contains() method that true if elements is present in the set.

```
import java.util.*;
class Demo{
  public static void main(String args[]){
    TreeSet<String> al = new TreeSet<String>();
    al.add("Ravi");
    al.add("Vijay");
    al.add("Ravi");
    al.add("Ajay");

    Iterator itr=al.iterator();
    while(itr.hasNext()){
      System.out.println(itr.next());
    }


    boolean iscontain = al.contains("Ravi");

    System.out.println("Is contain Ravi: "+iscontain);

  }

}
```

Copy

```
Ajay

Ravi

Vijay

Is contain Ravi: true
```

Traverse TreeSet in Ascending and Descending Order

We can traverse elements of Treeset in both ascending and descending order. TreeSet provides `descendingIterator()` method that returns iterator type for descending traversing. See the below example.

```
import java.util.*;

class Demo{

  public static void main(String args[]){

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

    al.add("Ravi");

    al.add("Vijay");

    al.add("Ravi");

    al.add("Ajay");

    System.out.println("Ascending...");

    Iterator itr=al.iterator();

    while(itr.hasNext()){

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

    }

    System.out.println("Descending...");

    Iterator itr2=al.descendingIterator();
    while(itr2.hasNext()){

      System.out.println(itr2.next());

    }

  }

}
```

Copy

```
OUTPUT

Ascending...

Ajay

Ravi

Vijay

Descending...

Vijay

Ravi

Ajay
```