Fetching an element from a TreeSet

The following methods can be used to fetch elements from a TreeSet.

Fetching the first element

We can fetch the first element in the **TreeSet** using the `first()` method. If the TreeSet is empty, then `NoSuchElementException` is thrown.

Fetching the last element

We can fetch the last element in the **TreeSet** using the `last()` method. If the TreeSet is empty, then `NoSuchElementException` is thrown.

Fetching the subset of elements

We can use the `subSet(E fromElement, E toElement)` method to fetch a subset of **TreeSet** within a given range. This method will return the elements ranging from `fromElement` to `toElement`. Note that `fromElement` is inclusive and `toElement` is exclusive.

Fetching elements that are smaller than the given element

The `headSet(E toElement)` method returns all the elements that are smaller than the provided element. The `toElement` is not inclusive.

Fetching elements that are greater than the given element

The `tailSet(E fromElement)` method returns all the elements which are greater than the provided element. The `fromElement` is not inclusive.

```
import java.util.TreeSet;

public class TreeSetDemo {

    public static void main(String args[]) {

        TreeSet<Integer> set = new TreeSet<>();
        set.add(21);
        set.add(32);
        set.add(44);
        set.add(11);
        set.add(54);
        set.add(3);
        set.add(9);
        set.add(41);

        System.out.println("Fetching the first element in TreeSet " + set.first());
        System.out.println("Fetching the last element in TreeSet " + set.last());
        System.out.println("Fetching all the elements less than 40 " + set.headSet(40));
        System.out.println("Fetching all the elements greater than 40 " + set.tailSet(40));
    }

}
```

Run Save Reset

Removing an element from a TreeSet

To remove an element from **TreeSet**, the `remove(Object o)` method can be used. This method returns `true` if the element is removed and returns `false` if the element is not present in the TreeSet.

```
import java.util.TreeSet;

public class TreeSetDemo {

    public static void main(String args[]) {

        TreeSet<Integer> set = new TreeSet<>();
        set.add(21);
        set.add(32);
        set.add(44);
        set.add(11);
        set.add(54);
        set.add(3);
        set.add(9);
        set.add(41);

        System.out.println("Removing 44 from TreeSet " + set.remove(new Integer(44)));
        System.out.println("Removing 100 from TreeSet " + set.remove(new Integer(100)));
    }

}
```

Run Save Reset

Additional operations on a TreeSet

1. The `isEmpty()` method can be used to check if the **TreeSet** is empty or contains some elements.
2. The `size()` method can be used to get the size of the **TreeSet**.
3. The `contains(Object o)` method is used to check if a particular element is present in the **TreeSet** or not.

```
import java.util.TreeSet;

public class TreeSetDemo {

    public static void main(String args[]) {

        TreeSet<Integer> set = new TreeSet<>();
        System.out.println("Checking if TreeSet is empty: " + set.isEmpty());
        System.out.println("Checking the TreeSet size: " + set.size());
        System.out.println("Checking if TreeSet contains 44: " + set.contains(new Integer(44)));

        set.add(21);
        set.add(32);
        set.add(44);
        set.add(11);

        System.out.println("Checking if TreeSet is empty: " + set.isEmpty());
        System.out.println("Checking the TreeSet size: " + set.size());
        System.out.println("Checking if TreeSet contains 44: " + set.contains(new Integer(44)));
    }

}
```

Run Save Reset