Java NavigableSet Interface

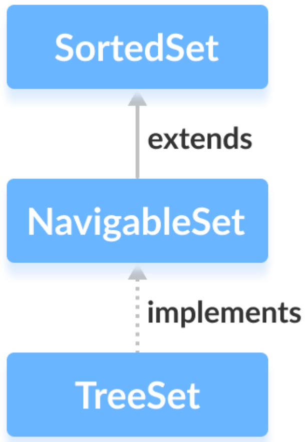
In this tutorial, we will learn about the Java NavigableSet interface and its methods with the help of an example.

The `NavigableSet` interface of the Java Collections framework provides the features to navigate among the set elements.

It is considered as a type of SortedSet.

Class that implements NavigableSet

In order to use the functionalities of the `NavigableSet` interface, we need to use the `TreeSet` class that implements `NavigableSet`.



How to use NavigableSet?

In Java, we must import the `java.util.NavigableSet` package to use `NavigableSet`. Once we import the package, here's how we can create navigable sets.

```
// SortedSet implementation by TreeSet class
NavigableSet<String> numbers = new TreeSet<>();
```

Here, we have created a navigable set named `numbers` of the `TreeSet` class.

Methods of NavigableSet

The `NavigableSet` is considered as a type of `SortedSet`. It is because `NavigableSet` extends the `SortedSet` interface.

Hence, all `SortedSet` methods are also available in `NavigableSet`. To learn how these methods, visit Java SortedSet.

However, some of the methods of `SortedSet` (`headSet()`, `tailSet()` and `subSet()`) are defined differently in `NavigableSet`.

Let's see how these methods are defined in `NavigableSet`.

---

headSet(element, booleanValue)

The `headSet()` method returns all the elements of a navigable set before the specified `element` (which is passed as an argument).

The `booleanValue` parameter is optional. Its default value is `false`.

If `true` is passed as a `booleanValue`, the method returns all the elements before the specified element including the specified element.

---

tailSet(element, booleanValue)

The `tailSet()` method returns all the elements of a navigable set after the specified `element` (which is passed as an argument) including the specified element.

The `booleanValue` parameter is optional. Its default value is `true`.

If `false` is passed as a `booleanValue`, the method returns all the elements after the specified element without including the specified element.

---

subSet(e1, bv1, e2, bv2)

The `subSet()` method returns all the elements between `e1` and `e2` including `e1`.

The `bv1` and `bv2` are optional parameters. The default value of `bv1` is `true`, and the default value of `bv2` is `false`.

If `false` is passed as `bv1`, the method returns all the elements between `e1` and `e2` without including `e1`.

If `true` is passed as `bv2`, the method returns all the elements between `e1` and `e2`, including `e1`.

Methods for Navigation

The `NavigableSet` provides various methods that can be used to navigate over its elements.

- **descendingSet()** - reverses the order of elements in a set
- **descendingIterator()** - returns an iterator that can be used to iterate over a set in reverse order
- **ceiling()** - returns the lowest element among those elements that are greater than or equal to the specified element
- **floor()** - returns the greatest element among those elements that are less than or equal to the specified element
- **higher()** - returns the lowest element among those elements that are greater than the specified element
- **lower()** - returns the greatest element among those elements that are less than the specified element
- **pollFirst()** - returns and removes the first element from the set
- **pollLast()** - returns and removes the last element from the set

To learn more about the `NavigableSet`, visit Java NavigableSet (official Java documentation).

Implementation of NavigableSet in TreeSet Class

```java
import java.util.NavigableSet;
import java.util.TreeSet;

class Main {

    public static void main(String[] args) {
        // Creating NavigableSet using the TreeSet
        NavigableSet<Integer> numbers = new TreeSet<>();

        // Insert elements to the set
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        System.out.println("NavigableSet: " + numbers);

        // Access the first element
        int firstElement = numbers.first();
        System.out.println("First Number: " + firstElement);

        // Access the last element
        int lastElement = numbers.last();
        System.out.println("Last Element: " + lastElement);

        // Remove the first element
        int number1 = numbers.pollFirst();
        System.out.println("Removed First Element: " + number1);

        // Remove the last element
        int number2 = numbers.pollLast();
```

Output

```
NavigableSet: [1, 2, 3]
First Element: 1
Last Element: 3
Removed First Element: 1
Removed Last Element: 3
```