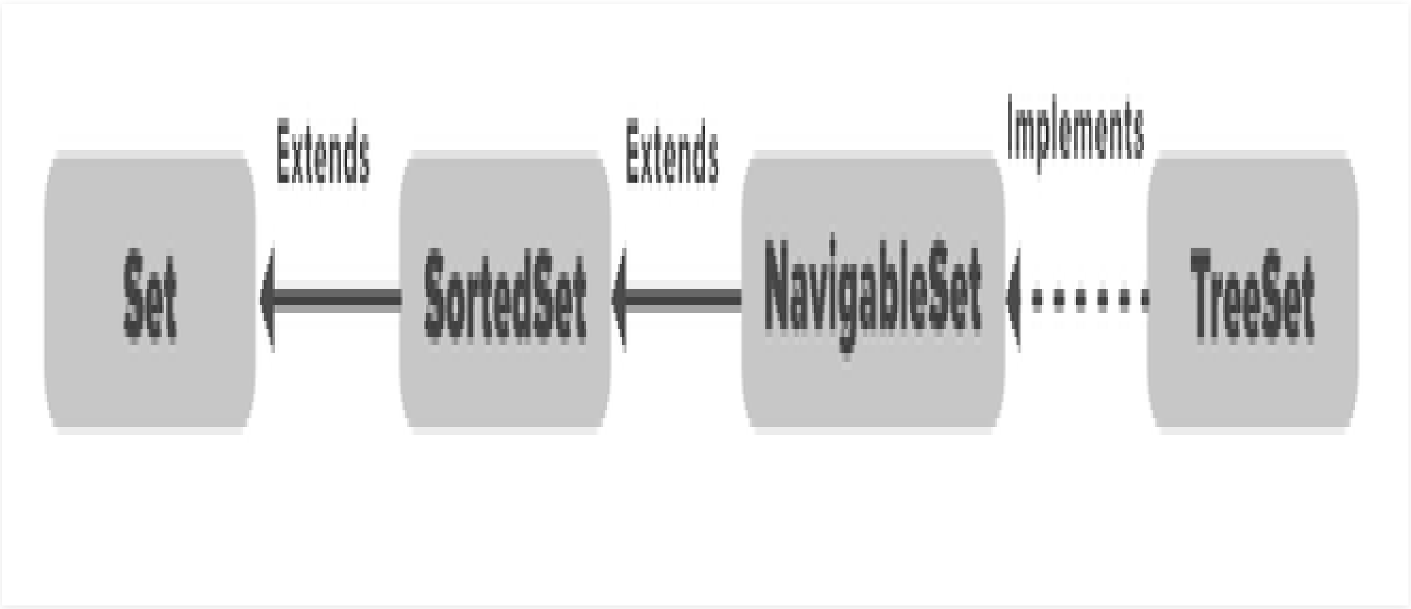
NavigableSet in Java with Examples

Difficulty Level : Easy Last Updated : 28 Jun, 2021

NavigableSet represents a navigable set in Java Collection Framework. The NavigableSet interface inherits from the SortedSet interface. It behaves like a SortedSet with the exception that we have navigation methods available in addition to the sorting mechanisms of the SortedSet.
For example, the NavigableSet interface can navigate the set in reverse order compared to the order defined in SortedSet. A NavigableSet may be accessed and traversed in either ascending or descending order. The classes that implement this interface are, TreeSet and ConcurrentSkipListSet



Here, E is the type of elements maintained by this set.

All Superinterfaces:

Collection<E>, Iterable<E>, Set<E>, SortedSet<E>

All Known Implementing Classes:

ConcurrentSkipListSet, TreeSet<E>

Declaration: The NavigableSet is declared as

public interface NavigableSet<E> extends SortedSet<E>

Creating NavigableSet Objects

Since NavigableSet is an interface, objects cannot be created of the type NavigableSet. We always need a class that extends this list in order to create an object. And also, after the introduction of Generics in Java 1.5, it is possible to restrict the type of object that can be stored in the NavigableSet. This type-safe set can be defined as:

// Obj is the type of the object to be stored in NavigableSet

NavigableSet<Obj> set = new TreeSet<Obj> ();

Example:

```
// Java program to demonstrate
// the working of NavigableSet
import java.util.NavigableSet;
import java.util.TreeSet;

public class NavigableSetDemo
{
    public static void main(String[] args)
    {
        NavigableSet<Integer> ns = new TreeSet<>();
        ns.add(0);
        ns.add(1);
        ns.add(2);
        ns.add(3);
        ns.add(4);
        ns.add(5);
        ns.add(6);

        // Get a reverse view of the navigable set
        NavigableSet<Integer> reverseNs = ns.descendingSet();

        // Print the normal and reverse views
        System.out.println("Normal order: " + ns);
        System.out.println("Reverse order: " + reverseNs);

        NavigableSet<Integer> threeOrMore = ns.tailSet(3, true);
        System.out.println("3 or  more:  " + threeOrMore);
        System.out.println("lower(3): " + ns.lower(3));
        System.out.println("floor(3): " + ns.floor(3));
        System.out.println("higher(3): " + ns.higher(3));
        System.out.println("ceiling(3): " + ns.ceiling(3));
        System.out.println("pollFirst(): " + ns.pollFirst());
        System.out.println("Navigable Set:  " + ns);

        System.out.println("pollLast(): " + ns.pollLast());
        System.out.println("Navigable Set:  " + ns);

        System.out.println("pollFirst(): " + ns.pollFirst());
        System.out.println("Navigable Set:  " + ns);

        System.out.println("pollFirst(): " + ns.pollFirst());
        System.out.println("Navigable Set:  " + ns);

        System.out.println("pollFirst(): " + ns.pollFirst());
        System.out.println("Navigable Set:  " + ns);

        System.out.println("pollFirst(): " + ns.pollFirst());
        System.out.println("pollLast(): " + ns.pollLast());
    }
}
```

Output

```
Normal order: [0, 1, 2, 3, 4, 5, 6]
Reverse order: [6, 5, 4, 3, 2, 1, 0]
3 or  more:  [3, 4, 5, 6]
lower(3): 2
floor(3): 3
higher(3): 4
ceiling(3): 3
pollFirst(): 0
Navigable Set:  [1, 2, 3, 4, 5, 6]
pollLast(): 6
Navigable Set:  [1, 2, 3, 4, 5]
pollFirst(): 1
Navigable Set:  [2, 3, 4, 5]
pollFirst(): 2
Navigable Set:  [3, 4, 5]
pollFirst(): 3
Navigable Set:  [4, 5]
pollFirst(): 4
pollLast(): 5
```

Performing Various Operations on NavigableSet

Since NavigableSet is an interface, it can be used only with a class that implements this interface. TreeSet is the class that implements the NavigableSet interface. Now, let's see how to perform a few frequently used operations on the TreeSet.

1. Adding Elements: In order to add an element to the NavigableSet, we can use the add() method. However, the insertion order is not retained in the TreeSet. Internally, for every element, the values are compared and sorted in the ascending order. We need to keep a note that duplicate elements are not allowed and all the duplicate elements are ignored. And also, Null values are not accepted by the NavigableSet.

```java
// Java code to demonstrate
// adding of elements in
// NavigableSet
import java.util.*;
import java.io.*;

class NavigableSetDemo {

    public static void main(String[] args)
    {
        NavigableSet<String> ts = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("A");
        ts.add("B");
        ts.add("C");
        ts.add("A");

        System.out.println(ts);
    }
}
```

Output:

```
[A, B, C]
```

2. Accessing the Elements: After adding the elements, if we wish to access the elements, we can use inbuilt methods like contains(), first(), last(), etc.

- contains()
- first()
- last()

```java
// Java program to access
// the elements of NavigableSet
import java.util.*;
import java.io.*;

class NavigableSetDemo {

    public static void main(String[] args)
    {
        NavigableSet<String> ts = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("A");
        ts.add("B");
        ts.add("C");
        ts.add("A");

        System.out.println("Navigable Set is " + ts);

        String check = "D";

        // Check if the above string exists in
        // the NavigableSet or not
        System.out.println("Contains " + check + " "
                           + ts.contains(check));

        // Print the first element in
        // the NavigableSet
        System.out.println("First Value " + ts.first());

        // Print the last element in
        // the NavigableSet
        System.out.println("Last Value " + ts.last());
    }
}
```

Output:

```
Navigable Set is [A, B, C]
Contains D false
First Value A
Last Value C
```

3. Removing the Values: The values can be removed from the NavigableSet using the remove(), pollFirst(), pollLast() methods.

- remove()
- pollFirst()
- pollLast()

```java
// Java Program to remove the
// elements from NavigableSet
import java.util.*;
import java.io.*;

class NavigableSetDemo {

    public static void main(String[] args)
    {
        NavigableSet<String> ts = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("A");
        ts.add("B");
        ts.add("C");
        ts.add("B");
        ts.add("D");
        ts.add("E");

        System.out.println("Initial TreeSet " + ts);
        // Removing the element b
        ts.remove("B");

        System.out.println("After removing element " + ts);

        // Remove the First element of TreeSet
        ts.pollFirst();

        System.out.println(
            "After the removal of First Element " + ts);

        // Remove the Last element of TreeSet
        ts.pollLast();

        System.out.println(
            "After the removal of Last Element " + ts);
    }
}
```

Output:

```
Initial TreeSet [A, B, C, D, E]
After removing element [A, C, D, E]
After the removal of First Element [C, D, E]
After the removal of Last Element [C, D]
```

4. Iterating through the NavigableSet: There are various ways to iterate through the NavigableSet. The most famous one is to use the enhanced for loop.

```
// Java program to iterate
// through NavigableSet

import java.util.*;
import java.io.*;

class NavigableSetDemo {

    public static void main(String[] args)
    {
        NavigableSet<String> ts = new TreeSet<String>();

        // Elements are added using add() method
        ts.add("C");
        ts.add("D");
        ts.add("E");
        ts.add("A");
        ts.add("B");
        ts.add("Z");

        // Iterating though the NavigableSet
        for (String value : ts)
            System.out.print(value + ", ");
        System.out.println();
    }
}
```

Output:

```
A, B, C, D, E, Z,
```

Methods of Navigable Set

The following are the methods present in the NavigableSet interface.

METHOD	DESCRIPTION
ceiling(E e)	Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
descendingIterator()	Returns an iterator over the elements in this set, in descending order.
descendingSet()	Returns a reverse order view of the elements contained in this set.
floor(E e)	Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
headSet(E toElement)	Returns a view of the portion of this set whose elements are strictly less than toElement.

METHOD	DESCRIPTION
headSet(E toElement, boolean inclusive)	Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
higher(E e)	Returns the least element in this set strictly greater than the given element, or null if there is no such element.
iterator()	Returns an iterator over the elements in this set, in ascending order.
lower(E e)	Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
pollFirst()	Retrieves and removes the first (lowest) element, or returns null if this set is empty.
pollLast()	Retrieves and removes the last (highest) element, or returns null if this set is empty.
subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)	Returns a view of the portion of this set whose elements range from fromElement to toElement.
subSet(E fromElement, E toElement)	Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
tailSet(E fromElement)	Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
tailSet(E fromElement, boolean inclusive)	Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

Methods inherited from interface java.util.SortedSet

METHOD	DESCRIPTION
comparator()	This method returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.

METHOD	DESCRIPTION
first()	This method returns the first(lowest) element present in this set.
last()	This method returns the last(highest) element present in the set.
spliterator()	Creates a Spliterator over the elements in this sorted set.

Methods inherited from interface java.util.Set

METHOD	DESCRIPTION
add(element)	This method is used to add a specific element to the set. The function adds the element only if the specified element is not already present in the set else the function returns False if the element is already present in the Set.
addAll(collection)	This method is used to append all of the elements from the mentioned collection to the existing set. The elements are added randomly without following any specific order.
clear()	This method is used to remove all the elements from the set but not delete the set. The reference for the set still exists.
contains(element)	This method is used to check whether a specific element is present in the Set or not.
containsAll(collection)	This method is used to check whether the set contains all the elements present in the given collection or not. This method returns true if the set contains all the elements and returns false if any of the elements are missing.
equals()	Compares the specified object with this set for equality.
hashCode()	This method is used to get the hashCode value for this instance of the Set. It returns an integer value which is the hashCode value for this instance of the Set.
isEmpty()	This method is used to check if a NavigableSet is empty or not.

METHOD	DESCRIPTION
remove(element)	This method is used to remove the given element from the set. This method returns True if the specified element is present in the Set otherwise it returns False.
removeAll(collection)	This method is used to remove all the elements from the collection which are present in the set. This method returns true if this set changed as a result of the call.
retainAll(collection)	This method is used to retain all the elements from the set which are mentioned in the given collection. This method returns true if this set changed as a result of the call.
size()	This method is used to get the size of the set. This returns an integer value which signifies the number of elements.
toArray()	This method is used to form an array of the same elements as that of the Set.
toArray(T[] a)	Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Methods declared in interface java.util.Collection

METHOD	DESCRIPTION
parallelStream()	Returns a possibly parallel Stream with this collection as its source.
removeIf (Predicate<? super E> filter)	Removes all of the elements of this collection that satisfy the given predicate.
stream()	Returns a sequential Stream with this collection as its source.
toArray (IntFunction<T[]> generator)	Returns an array containing all of the elements in this collection, using the provided generator function to allocate the returned array.

Methods declared in interface java.lang.Iterable

METHOD	DESCRIPTION
forEach (Consumer<? super T> action)	Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.