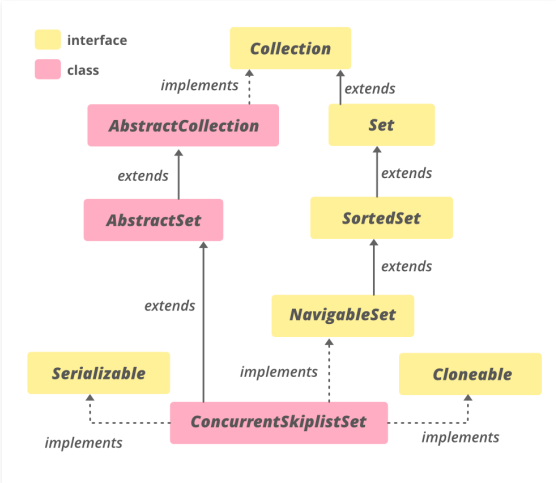
ConcurrentSkipListSet in Java with Examples

Last Updated : 11 Nov, 2020

The **ConcurrentSkipListSet** class in Java is a part of the Java Collection Framework and implements the **Collection interface** and the **AbstractSet class**. It provides a scalable and concurrent version of NavigableSet in Java. The implementation of ConcurrentSkipListSet is based on **ConcurrentSkipListMap**. The elements in ConcurrentSkipListSet are sorted by default in their natural ordering or by a Comparator provided at set creation time, depending on which constructor is used.

Since it implements **SortedSet<E>** and **NavigableSet<E>**, it is similar to TreeSet with an added feature of being concurrent. Since it is a thread-safe, it can be used by multiple threads concurrently whereas TreeSet is not thread-safe.

Class Hierarchy:



Declaration:

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

Where E is the type of elements maintained by this collection
```

Some important points on ConcurrentSkipListSet:

- It implements **Serializable**, **Cloneable**, **Iterable<E>**, **Collection<E>**, NavigableSet<E>, Set<E>, SortedSet<E> interfaces.
- It does not allow null elements, because null arguments and return values cannot be reliably distinguished from the absence of elements.
- Its implementation provides average log(n) time cost for contains, add, and remove operations and their variants.
- It is thread-safe.
- It should be preferred over implementing Set interface when concurrent modification of set is required.

Constructors:

1. ConcurrentSkipListSet(): This constructor is used to construct an empty set.

ConcurrentSkipListSet<E> set = new ConcurrentSkipListSet<E>();

2. ConcurrentSkipListSet(Collection<E> c): This constructor is used to construct a set with the elements of the Collection passed as the parameter.

ConcurrentSkipListSet<E> set = new ConcurrentSkipListSet<E>(Collection<E> c);

3. ConcurrentSkipListSet(Comparator<E> comparator): This constructor is used to construct a new, empty set that orders its elements according to the specified comparator.

ConcurrentSkipListSet<E> set = new ConcurrentSkipListSet<E>(Comparator<E> comparator);

4. ConcurrentSkipListSet(SortedSet<E> s): This constructor is used to construct a new set containing the same elements and using the same ordering as the specified sorted set.

ConcurrentSkipListSet<E> set = new ConcurrentSkipListSet<E>(SortedSet<E> s);

Example 1:

```
// Java program to demonstrate ConcurrentSkipListSet

import java.util.*;
import java.util.concurrent.ConcurrentSkipListSet;

class ConcurrentSkipListSetLastExample1 {
    public static void main(String[] args)
    {

        // Initializing the set using
        // ConcurrentSkipListSet()
        ConcurrentSkipListSet<Integer> set
            = new ConcurrentSkipListSet<Integer>();

        // Adding elements to this set
        set.add(78);
        set.add(64);
        set.add(12);
        set.add(45);
        set.add(8);

        // Printing the ConcurrentSkipListSet
        System.out.println("ConcurrentSkipListSet: " + set);

        // Initializing the set using
        // ConcurrentSkipListSet(Collection)
        ConcurrentSkipListSet<Integer> set1
            = new ConcurrentSkipListSet<Integer>(set);

        // Printing the ConcurrentSkipListSet1
        System.out.println("ConcurrentSkipListSet1: "
```

```
                            + set1);

        // Initializing the set using
        // ConcurrentSkipListSet()
        ConcurrentSkipListSet<String> set2
            = new ConcurrentSkipListSet<>();

        // Adding elements to this set
        set2.add("Apple");
        set2.add("Lemon");
        set2.add("Banana");
        set2.add("Apple");

        // creating an iterator
        Iterator<String> itr = set2.iterator();

        System.out.print("Fruits Set: ");
        while (itr.hasNext()) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

Output:

```
ConcurrentSkipListSet: [8, 12, 45, 64, 78]
ConcurrentSkipListSet1: [8, 12, 45, 64, 78]
Fruits Set: Apple Banana Lemon
```

Example 2:

```
// Java program to demonstrate ConcurrentSkipListSet

import java.util.concurrent.ConcurrentSkipListSet;

class ConcurrentSkipListSetLastExample1 {

    public static void main(String[] args)
    {

        // Initializing the set using ConcurrentSkipListSet()
        ConcurrentSkipListSet<Integer>
            set = new ConcurrentSkipListSet<Integer>();

        // Adding elements to this set
        // using add() method
        set.add(78);
        set.add(64);
        set.add(12);
        set.add(45);
        set.add(8);

        // Printing the ConcurrentSkipListSet
        System.out.println("ConcurrentSkipListSet: "
                           + set);
```

```
        // Printing the highest element of the set
        // using last() method
        System.out.println("The highest element of the set: "
                           + set.last());

        // Retrieving and removing first element of the set
        System.out.println("The first element of the set: "
                           + set.pollFirst());

        // Checks if 9 is present in the set
        // using contains() method
        if (set.contains(9))
            System.out.println("9 is present in the set.");
        else
            System.out.println("9 is not present in the set.");

        // Printing the size of the set
        // using size() method
        System.out.println("Number of elements in the set = "
                           + set.size());
    }
}
```

Output:

```
ConcurrentSkipListSet: [8, 12, 45, 64, 78]
The highest element of the set: 78
The first element of the set: 8
9 is not present in the set.
Number of elements in the set = 4
```

Methods of ConcurrentSkipListSet

METHOD	DESCRIPTION
add(E e)	Adds the specified element to this set if it is not already present.
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.
clear()	Removes all of the elements from this set.
clone()	Returns a shallow copy of this ConcurrentSkipListSet instance.
comparator()	Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.
contains(Object o)	Returns true if this set contains the specified 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.
equals(Object o)	Compares the specified object with this set for equality.

METHOD	DESCRIPTION
first()	Returns the first (lowest) element currently 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.
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.
isEmpty()	Returns an iterator over the elements in this set in ascending order.
last()	Returns the last (highest) element currently in this set.
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.
remove(Object o)	Removes the specified element from this set if it is present.
removeAll(Collection<?> c)	Removes from this set all of its elements that are contained in the specified collection.
size()	Returns the number of elements in this set.
spliterator()	Returns a Spliterator over the elements in this set.
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 class java.util.AbstractSet

METHOD	DESCRIPTION

METHOD	DESCRIPTION
hashCode()	Returns the hash code value for this set.

Methods inherited from class java.util.AbstractCollection

METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this collection (optional operation).
containsAll(Collection<?> c)	Returns true if this collection contains all of the elements in the specified collection.
retainAll(Collection<?> c)	Retains only the elements in this collection that are contained in the specified collection (optional operation).
toArray()	Returns an array containing all of the elements in this collection.
toArray(T[] a)	Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.
toString()	Returns a string representation of this collection.

Methods inherited from interface java.util.Set

METHOD	DESCRIPTION
addAll(Collection<? extends E> c)	Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
containsAll(Collection<?> c)	Returns true if this set contains all of the elements of the specified collection.
hashCode()	Returns the hash code value for this set.
retainAll(Collection<?> c)	Retains only the elements in this set that are contained in the specified collection (optional operation).
toArray()	Returns an array containing all of the elements in this 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 inherited from interface java.util.Collection

METHOD	DESCRIPTION

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.

Methods inherited from 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.