| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ConcurrentSkipListSet.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/ConcurrentSkipListSet.html)    [**NO FRAMES**](http://docs.google.com/ConcurrentSkipListSet.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#2s8eyo1) | [METHOD](#35nkun2) |

## **java.util.concurrent**

Class ConcurrentSkipListSet<E>

[java.lang.Object](http://docs.google.com/java/lang/Object.html)  
 [java.util.AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<E>  
 [java.util.AbstractSet](http://docs.google.com/java/util/AbstractSet.html)<E>  
 **java.util.concurrent.ConcurrentSkipListSet<E>**

**Type Parameters:**E - the type of elements maintained by this set **All Implemented Interfaces:** [Serializable](http://docs.google.com/java/io/Serializable.html), [Cloneable](http://docs.google.com/java/lang/Cloneable.html), [Iterable](http://docs.google.com/java/lang/Iterable.html)<E>, [Collection](http://docs.google.com/java/util/Collection.html)<E>, [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<E>, [Set](http://docs.google.com/java/util/Set.html)<E>, [SortedSet](http://docs.google.com/java/util/SortedSet.html)<E>

public class **ConcurrentSkipListSet<E>**extends [AbstractSet](http://docs.google.com/java/util/AbstractSet.html)<E>implements [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<E>, [Cloneable](http://docs.google.com/java/lang/Cloneable.html), [Serializable](http://docs.google.com/java/io/Serializable.html)

A scalable concurrent [NavigableSet](http://docs.google.com/java/util/NavigableSet.html) implementation based on a [ConcurrentSkipListMap](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html). The elements of the set are kept sorted according to their [natural ordering](http://docs.google.com/java/lang/Comparable.html), or by a [Comparator](http://docs.google.com/java/util/Comparator.html) provided at set creation time, depending on which constructor is used.

This implementation provides expected average *log(n)* time cost for the contains, add, and remove operations and their variants. Insertion, removal, and access operations safely execute concurrently by multiple threads. Iterators are *weakly consistent*, returning elements reflecting the state of the set at some point at or since the creation of the iterator. They do *not* throw [ConcurrentModificationException](http://docs.google.com/java/util/ConcurrentModificationException.html), and may proceed concurrently with other operations. Ascending ordered views and their iterators are faster than descending ones.

Beware that, unlike in most collections, the size method is *not* a constant-time operation. Because of the asynchronous nature of these sets, determining the current number of elements requires a traversal of the elements. Additionally, the bulk operations addAll, removeAll, retainAll, and containsAll are *not* guaranteed to be performed atomically. For example, an iterator operating concurrently with an addAll operation might view only some of the added elements.

This class and its iterators implement all of the *optional* methods of the [Set](http://docs.google.com/java/util/Set.html) and [Iterator](http://docs.google.com/java/util/Iterator.html) interfaces. Like most other concurrent collection implementations, this class does not permit the use of null elements, because null arguments and return values cannot be reliably distinguished from the absence of elements.

This class is a member of the  [Java Collections Framework](http://docs.google.com/technotes/guides/collections/index.html).

**Since:** 1.6 **See Also:**[Serialized Form](http://docs.google.com/serialized-form.html#java.util.concurrent.ConcurrentSkipListSet)

| **Constructor Summary** | |
| --- | --- |
| [**ConcurrentSkipListSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#ConcurrentSkipListSet())()            Constructs a new, empty set that orders its elements according to their [natural ordering](http://docs.google.com/java/lang/Comparable.html). |
| [**ConcurrentSkipListSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#ConcurrentSkipListSet(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> c)            Constructs a new set containing the elements in the specified collection, that orders its elements according to their [natural ordering](http://docs.google.com/java/lang/Comparable.html). |
| [**ConcurrentSkipListSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#ConcurrentSkipListSet(java.util.Comparator))([Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> comparator)            Constructs a new, empty set that orders its elements according to the specified comparator. |
| [**ConcurrentSkipListSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#ConcurrentSkipListSet(java.util.SortedSet))([SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> s)            Constructs a new set containing the same elements and using the same ordering as the specified sorted set. |

| **Method Summary** | |
| --- | --- |
| boolean | [**add**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#add(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)            Adds the specified element to this set if it is not already present. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**ceiling**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#ceiling(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)            Returns the least element in this set greater than or equal to the given element, or null if there is no such element. |
| void | [**clear**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#clear())()            Removes all of the elements from this set. |
| [ConcurrentSkipListSet](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**clone**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#clone())()            Returns a shallow copy of this ConcurrentSkipListSet instance. |
| [Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**comparator**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#comparator())()            Returns the comparator used to order the elements in this set, or null if this set uses the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements. |
| boolean | [**contains**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#contains(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Returns true if this set contains the specified element. |
| [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**descendingIterator**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#descendingIterator())()            Returns an iterator over the elements in this set in descending order. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**descendingSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#descendingSet())()            Returns a reverse order view of the elements contained in this set. |
| boolean | [**equals**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#equals(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Compares the specified object with this set for equality. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**first**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#first())()            Returns the first (lowest) element currently in this set. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**floor**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#floor(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)            Returns the greatest element in this set less than or equal to the given element, or null if there is no such element. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**headSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#headSet(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement)            Returns a view of the portion of this set whose elements are strictly less than toElement. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**headSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#headSet(E,%20boolean))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) 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. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**higher**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#higher(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)            Returns the least element in this set strictly greater than the given element, or null if there is no such element. |
| boolean | [**isEmpty**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#isEmpty())()            Returns true if this set contains no elements. |
| [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**iterator**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#iterator())()            Returns an iterator over the elements in this set in ascending order. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**last**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#last())()            Returns the last (highest) element currently in this set. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**lower**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#lower(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)            Returns the greatest element in this set strictly less than the given element, or null if there is no such element. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**pollFirst**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#pollFirst())()            Retrieves and removes the first (lowest) element, or returns null if this set is empty. |
| [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) | [**pollLast**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#pollLast())()            Retrieves and removes the last (highest) element, or returns null if this set is empty. |
| boolean | [**remove**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#remove(java.lang.Object))([Object](http://docs.google.com/java/lang/Object.html) o)            Removes the specified element from this set if it is present. |
| boolean | [**removeAll**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#removeAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<?> c)            Removes from this set all of its elements that are contained in the specified collection. |
| int | [**size**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#size())()            Returns the number of elements in this set. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**subSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#subSet(E,%20boolean,%20E,%20boolean))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement, boolean fromInclusive, [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement, boolean toInclusive)            Returns a view of the portion of this set whose elements range from fromElement to toElement. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**subSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#subSet(E,%20E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement, [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement)            Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**tailSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#tailSet(E))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement)            Returns a view of the portion of this set whose elements are greater than or equal to fromElement. |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> | [**tailSet**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html#tailSet(E,%20boolean))([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) 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**](http://docs.google.com/java/util/AbstractSet.html) |
| --- |
| [hashCode](http://docs.google.com/java/util/AbstractSet.html#hashCode()) |

| **Methods inherited from class java.util.**[**AbstractCollection**](http://docs.google.com/java/util/AbstractCollection.html) |
| --- |
| [addAll](http://docs.google.com/java/util/AbstractCollection.html#addAll(java.util.Collection)), [containsAll](http://docs.google.com/java/util/AbstractCollection.html#containsAll(java.util.Collection)), [retainAll](http://docs.google.com/java/util/AbstractCollection.html#retainAll(java.util.Collection)), [toArray](http://docs.google.com/java/util/AbstractCollection.html#toArray()), [toArray](http://docs.google.com/java/util/AbstractCollection.html#toArray(T%5B%5D)), [toString](http://docs.google.com/java/util/AbstractCollection.html#toString()) |

| **Methods inherited from class java.lang.**[**Object**](http://docs.google.com/java/lang/Object.html) |
| --- |
| [finalize](http://docs.google.com/java/lang/Object.html#finalize()), [getClass](http://docs.google.com/java/lang/Object.html#getClass()), [notify](http://docs.google.com/java/lang/Object.html#notify()), [notifyAll](http://docs.google.com/java/lang/Object.html#notifyAll()), [wait](http://docs.google.com/java/lang/Object.html#wait()), [wait](http://docs.google.com/java/lang/Object.html#wait(long)), [wait](http://docs.google.com/java/lang/Object.html#wait(long,%20int)) |

| **Methods inherited from interface java.util.**[**Set**](http://docs.google.com/java/util/Set.html) |
| --- |
| [addAll](http://docs.google.com/java/util/Set.html#addAll(java.util.Collection)), [containsAll](http://docs.google.com/java/util/Set.html#containsAll(java.util.Collection)), [hashCode](http://docs.google.com/java/util/Set.html#hashCode()), [retainAll](http://docs.google.com/java/util/Set.html#retainAll(java.util.Collection)), [toArray](http://docs.google.com/java/util/Set.html#toArray()), [toArray](http://docs.google.com/java/util/Set.html#toArray(T%5B%5D)) |

| **Constructor Detail** |
| --- |

### ConcurrentSkipListSet

public **ConcurrentSkipListSet**()

Constructs a new, empty set that orders its elements according to their [natural ordering](http://docs.google.com/java/lang/Comparable.html).

### ConcurrentSkipListSet

public **ConcurrentSkipListSet**([Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> comparator)

Constructs a new, empty set that orders its elements according to the specified comparator.

**Parameters:**comparator - the comparator that will be used to order this set. If null, the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of the elements will be used.

### ConcurrentSkipListSet

public **ConcurrentSkipListSet**([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> c)

Constructs a new set containing the elements in the specified collection, that orders its elements according to their [natural ordering](http://docs.google.com/java/lang/Comparable.html).

**Parameters:**c - The elements that will comprise the new set **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the elements in c are not [Comparable](http://docs.google.com/java/lang/Comparable.html), or are not mutually comparable [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection or any of its elements are null

### ConcurrentSkipListSet

public **ConcurrentSkipListSet**([SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> s)

Constructs a new set containing the same elements and using the same ordering as the specified sorted set.

**Parameters:**s - sorted set whose elements will comprise the new set **Throws:** [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified sorted set or any of its elements are null

| **Method Detail** |
| --- |

### clone

public [ConcurrentSkipListSet](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **clone**()

Returns a shallow copy of this ConcurrentSkipListSet instance. (The elements themselves are not cloned.)

**Overrides:**[clone](http://docs.google.com/java/lang/Object.html#clone()) in class [Object](http://docs.google.com/java/lang/Object.html) **Returns:**a shallow copy of this set**See Also:**[Cloneable](http://docs.google.com/java/lang/Cloneable.html)

### size

public int **size**()

Returns the number of elements in this set. If this set contains more than Integer.MAX\_VALUE elements, it returns Integer.MAX\_VALUE.

Beware that, unlike in most collections, this method is *NOT* a constant-time operation. Because of the asynchronous nature of these sets, determining the current number of elements requires traversing them all to count them. Additionally, it is possible for the size to change during execution of this method, in which case the returned result will be inaccurate. Thus, this method is typically not very useful in concurrent applications.

**Specified by:**[size](http://docs.google.com/java/util/Collection.html#size()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[size](http://docs.google.com/java/util/Set.html#size()) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[size](http://docs.google.com/java/util/AbstractCollection.html#size()) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the number of elements in this set

### isEmpty

public boolean **isEmpty**()

Returns true if this set contains no elements.

**Specified by:**[isEmpty](http://docs.google.com/java/util/Collection.html#isEmpty()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[isEmpty](http://docs.google.com/java/util/Set.html#isEmpty()) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[isEmpty](http://docs.google.com/java/util/AbstractCollection.html#isEmpty()) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**true if this set contains no elements

### contains

public boolean **contains**([Object](http://docs.google.com/java/lang/Object.html) o)

Returns true if this set contains the specified element. More formally, returns true if and only if this set contains an element e such that o.equals(e).

**Specified by:**[contains](http://docs.google.com/java/util/Collection.html#contains(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[contains](http://docs.google.com/java/util/Set.html#contains(java.lang.Object)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[contains](http://docs.google.com/java/util/AbstractCollection.html#contains(java.lang.Object)) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**o - object to be checked for containment in this set **Returns:**true if this set contains the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the specified element cannot be compared with the elements currently in this set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### add

public boolean **add**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)

Adds the specified element to this set if it is not already present. More formally, adds the specified element e to this set if the set contains no element e2 such that e.equals(e2). If this set already contains the element, the call leaves the set unchanged and returns false.

**Specified by:**[add](http://docs.google.com/java/util/Collection.html#add(E)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[add](http://docs.google.com/java/util/Set.html#add(E)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[add](http://docs.google.com/java/util/AbstractCollection.html#add(E)) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**e - element to be added to this set **Returns:**true if this set did not already contain the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if e cannot be compared with the elements currently in this set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### remove

public boolean **remove**([Object](http://docs.google.com/java/lang/Object.html) o)

Removes the specified element from this set if it is present. More formally, removes an element e such that o.equals(e), if this set contains such an element. Returns true if this set contained the element (or equivalently, if this set changed as a result of the call). (This set will not contain the element once the call returns.)

**Specified by:**[remove](http://docs.google.com/java/util/Collection.html#remove(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[remove](http://docs.google.com/java/util/Set.html#remove(java.lang.Object)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[remove](http://docs.google.com/java/util/AbstractCollection.html#remove(java.lang.Object)) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**o - object to be removed from this set, if present **Returns:**true if this set contained the specified element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if o cannot be compared with the elements currently in this set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### clear

public void **clear**()

Removes all of the elements from this set.

**Specified by:**[clear](http://docs.google.com/java/util/Collection.html#clear()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[clear](http://docs.google.com/java/util/Set.html#clear()) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[clear](http://docs.google.com/java/util/AbstractCollection.html#clear()) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>

### iterator

public [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **iterator**()

Returns an iterator over the elements in this set in ascending order.

**Specified by:**[iterator](http://docs.google.com/java/lang/Iterable.html#iterator()) in interface [Iterable](http://docs.google.com/java/lang/Iterable.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[iterator](http://docs.google.com/java/util/Collection.html#iterator()) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[iterator](http://docs.google.com/java/util/NavigableSet.html#iterator()) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[iterator](http://docs.google.com/java/util/Set.html#iterator()) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[iterator](http://docs.google.com/java/util/AbstractCollection.html#iterator()) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**an iterator over the elements in this set in ascending order

### descendingIterator

public [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **descendingIterator**()

Returns an iterator over the elements in this set in descending order.

**Specified by:**[descendingIterator](http://docs.google.com/java/util/NavigableSet.html#descendingIterator()) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**an iterator over the elements in this set in descending order

### equals

public boolean **equals**([Object](http://docs.google.com/java/lang/Object.html) o)

Compares the specified object with this set for equality. Returns true if the specified object is also a set, the two sets have the same size, and every member of the specified set is contained in this set (or equivalently, every member of this set is contained in the specified set). This definition ensures that the equals method works properly across different implementations of the set interface.

**Specified by:**[equals](http://docs.google.com/java/util/Collection.html#equals(java.lang.Object)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[equals](http://docs.google.com/java/util/Set.html#equals(java.lang.Object)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[equals](http://docs.google.com/java/util/AbstractSet.html#equals(java.lang.Object)) in class [AbstractSet](http://docs.google.com/java/util/AbstractSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**o - the object to be compared for equality with this set **Returns:**true if the specified object is equal to this set**See Also:**[Object.hashCode()](http://docs.google.com/java/lang/Object.html#hashCode()), [Hashtable](http://docs.google.com/java/util/Hashtable.html)

### removeAll

public boolean **removeAll**([Collection](http://docs.google.com/java/util/Collection.html)<?> c)

Removes from this set all of its elements that are contained in the specified collection. If the specified collection is also a set, this operation effectively modifies this set so that its value is the *asymmetric set difference* of the two sets.

**Specified by:**[removeAll](http://docs.google.com/java/util/Collection.html#removeAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[removeAll](http://docs.google.com/java/util/Set.html#removeAll(java.util.Collection)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Overrides:**[removeAll](http://docs.google.com/java/util/AbstractSet.html#removeAll(java.util.Collection)) in class [AbstractSet](http://docs.google.com/java/util/AbstractSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**c - collection containing elements to be removed from this set **Returns:**true if this set changed as a result of the call **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the types of one or more elements in this set are incompatible with the specified collection [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection or any of its elements are null**See Also:**[AbstractCollection.remove(Object)](http://docs.google.com/java/util/AbstractCollection.html#remove(java.lang.Object)), [AbstractCollection.contains(Object)](http://docs.google.com/java/util/AbstractCollection.html#contains(java.lang.Object))

### lower

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **lower**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#lower(E)) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.

**Specified by:**[lower](http://docs.google.com/java/util/NavigableSet.html#lower(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**e - the value to match **Returns:**the greatest element less than e, or null if there is no such element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the specified element cannot be compared with the elements currently in the set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### floor

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **floor**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#floor(E)) Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.

**Specified by:**[floor](http://docs.google.com/java/util/NavigableSet.html#floor(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**e - the value to match **Returns:**the greatest element less than or equal to e, or null if there is no such element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the specified element cannot be compared with the elements currently in the set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### ceiling

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **ceiling**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#ceiling(E)) Returns the least element in this set greater than or equal to the given element, or null if there is no such element.

**Specified by:**[ceiling](http://docs.google.com/java/util/NavigableSet.html#ceiling(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**e - the value to match **Returns:**the least element greater than or equal to e, or null if there is no such element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the specified element cannot be compared with the elements currently in the set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### higher

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **higher**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) e)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#higher(E)) Returns the least element in this set strictly greater than the given element, or null if there is no such element.

**Specified by:**[higher](http://docs.google.com/java/util/NavigableSet.html#higher(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**e - the value to match **Returns:**the least element greater than e, or null if there is no such element **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if the specified element cannot be compared with the elements currently in the set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified element is null

### pollFirst

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **pollFirst**()

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#pollFirst()) Retrieves and removes the first (lowest) element, or returns null if this set is empty.

**Specified by:**[pollFirst](http://docs.google.com/java/util/NavigableSet.html#pollFirst()) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the first element, or null if this set is empty

### pollLast

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **pollLast**()

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#pollLast()) Retrieves and removes the last (highest) element, or returns null if this set is empty.

**Specified by:**[pollLast](http://docs.google.com/java/util/NavigableSet.html#pollLast()) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the last element, or null if this set is empty

### comparator

public [Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **comparator**()

**Description copied from interface:** [**SortedSet**](http://docs.google.com/java/util/SortedSet.html#comparator()) Returns the comparator used to order the elements in this set, or null if this set uses the [natural ordering](http://docs.google.com/java/lang/Comparable.html) of its elements.

**Specified by:**[comparator](http://docs.google.com/java/util/SortedSet.html#comparator()) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements

### first

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **first**()

**Description copied from interface:** [**SortedSet**](http://docs.google.com/java/util/SortedSet.html#first()) Returns the first (lowest) element currently in this set.

**Specified by:**[first](http://docs.google.com/java/util/SortedSet.html#first()) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the first (lowest) element currently in this set **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if this set is empty

### last

public [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) **last**()

**Description copied from interface:** [**SortedSet**](http://docs.google.com/java/util/SortedSet.html#last()) Returns the last (highest) element currently in this set.

**Specified by:**[last](http://docs.google.com/java/util/SortedSet.html#last()) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**the last (highest) element currently in this set **Throws:** [NoSuchElementException](http://docs.google.com/java/util/NoSuchElementException.html) - if this set is empty

### subSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **subSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement,  
 boolean fromInclusive,  
 [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement,  
 boolean toInclusive)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#subSet(E,%20boolean,%20E,%20boolean)) Returns a view of the portion of this set whose elements range from fromElement to toElement. If fromElement and toElement are equal, the returned set is empty unless fromExclusive and toExclusive are both true. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

**Specified by:**[subSet](http://docs.google.com/java/util/NavigableSet.html#subSet(E,%20boolean,%20E,%20boolean)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**fromElement - low endpoint of the returned setfromInclusive - true if the low endpoint is to be included in the returned viewtoElement - high endpoint of the returned settoInclusive - true if the high endpoint is to be included in the returned view **Returns:**a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if fromElement and toElement cannot be compared to one another using this set's comparator (or, if the set has no comparator, using natural ordering). Implementations may, but are not required to, throw this exception if fromElement or toElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if fromElement or toElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromElement is greater than toElement; or if this set itself has a restricted range, and fromElement or toElement lies outside the bounds of the range.

### headSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **headSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement,  
 boolean inclusive)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#headSet(E,%20boolean)) Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

**Specified by:**[headSet](http://docs.google.com/java/util/NavigableSet.html#headSet(E,%20boolean)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**toElement - high endpoint of the returned setinclusive - true if the high endpoint is to be included in the returned view **Returns:**a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if toElement is not compatible with this set's comparator (or, if the set has no comparator, if toElement does not implement [Comparable](http://docs.google.com/java/lang/Comparable.html)). Implementations may, but are not required to, throw this exception if toElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if toElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if this set itself has a restricted range, and toElement lies outside the bounds of the range

### tailSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **tailSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement,  
 boolean inclusive)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#tailSet(E,%20boolean)) Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

**Specified by:**[tailSet](http://docs.google.com/java/util/NavigableSet.html#tailSet(E,%20boolean)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**fromElement - low endpoint of the returned setinclusive - true if the low endpoint is to be included in the returned view **Returns:**a view of the portion of this set whose elements are greater than or equal to fromElement **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if fromElement is not compatible with this set's comparator (or, if the set has no comparator, if fromElement does not implement [Comparable](http://docs.google.com/java/lang/Comparable.html)). Implementations may, but are not required to, throw this exception if fromElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if fromElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if this set itself has a restricted range, and fromElement lies outside the bounds of the range

### subSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **subSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement,  
 [E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#subSet(E,%20E)) Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. (If fromElement and toElement are equal, the returned set is empty.) The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

Equivalent to subSet(fromElement, true, toElement, false).

**Specified by:**[subSet](http://docs.google.com/java/util/NavigableSet.html#subSet(E,%20E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[subSet](http://docs.google.com/java/util/SortedSet.html#subSet(E,%20E)) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**fromElement - low endpoint (inclusive) of the returned settoElement - high endpoint (exclusive) of the returned set **Returns:**a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if fromElement and toElement cannot be compared to one another using this set's comparator (or, if the set has no comparator, using natural ordering). Implementations may, but are not required to, throw this exception if fromElement or toElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if fromElement or toElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if fromElement is greater than toElement; or if this set itself has a restricted range, and fromElement or toElement lies outside the bounds of the range

### headSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **headSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) toElement)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#headSet(E)) Returns a view of the portion of this set whose elements are strictly less than toElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

Equivalent to headSet(toElement, false).

**Specified by:**[headSet](http://docs.google.com/java/util/NavigableSet.html#headSet(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[headSet](http://docs.google.com/java/util/SortedSet.html#headSet(E)) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**toElement - high endpoint (exclusive) of the returned set **Returns:**a view of the portion of this set whose elements are strictly less than toElement **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if toElement is not compatible with this set's comparator (or, if the set has no comparator, if toElement does not implement [Comparable](http://docs.google.com/java/lang/Comparable.html)). Implementations may, but are not required to, throw this exception if toElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if toElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if this set itself has a restricted range, and toElement lies outside the bounds of the range na

### tailSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **tailSet**([E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html) fromElement)

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.html#tailSet(E)) Returns a view of the portion of this set whose elements are greater than or equal to fromElement. The returned set is backed by this set, so changes in the returned set are reflected in this set, and vice-versa. The returned set supports all optional set operations that this set supports.

The returned set will throw an IllegalArgumentException on an attempt to insert an element outside its range.

Equivalent to tailSet(fromElement, true).

**Specified by:**[tailSet](http://docs.google.com/java/util/NavigableSet.html#tailSet(E)) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)>**Specified by:**[tailSet](http://docs.google.com/java/util/SortedSet.html#tailSet(E)) in interface [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Parameters:**fromElement - low endpoint (inclusive) of the returned set **Returns:**a view of the portion of this set whose elements are greater than or equal to fromElement **Throws:** [ClassCastException](http://docs.google.com/java/lang/ClassCastException.html) - if fromElement is not compatible with this set's comparator (or, if the set has no comparator, if fromElement does not implement [Comparable](http://docs.google.com/java/lang/Comparable.html)). Implementations may, but are not required to, throw this exception if fromElement cannot be compared to elements currently in the set. [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if fromElement is null [IllegalArgumentException](http://docs.google.com/java/lang/IllegalArgumentException.html) - if this set itself has a restricted range, and fromElement lies outside the bounds of the range

### descendingSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **descendingSet**()

Returns a reverse order view of the elements contained in this set. The descending set is backed by this set, so changes to the set are reflected in the descending set, and vice-versa.

The returned set has an ordering equivalent to [Collections.reverseOrder](http://docs.google.com/java/util/Collections.html#reverseOrder(java.util.Comparator))(comparator()). The expression s.descendingSet().descendingSet() returns a view of s essentially equivalent to s.

**Specified by:**[descendingSet](http://docs.google.com/java/util/NavigableSet.html#descendingSet()) in interface [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/concurrent/ConcurrentSkipListSet.html)> **Returns:**a reverse order view of this set

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ConcurrentSkipListSet.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/ConcurrentSkipListMap.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/CopyOnWriteArrayList.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/ConcurrentSkipListSet.html)    [**NO FRAMES**](http://docs.google.com/ConcurrentSkipListSet.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | [CONSTR](#3znysh7) | [METHOD](#2et92p0) | DETAIL: FIELD | [CONSTR](#2s8eyo1) | [METHOD](#35nkun2) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).