| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/TreeSet.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/TreeMap.html)   [**NEXT CLASS**](http://docs.google.com/java/util/UnknownFormatConversionException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/TreeSet.html)    [**NO FRAMES**](http://docs.google.com/TreeSet.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**

Class TreeSet<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.TreeSet<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 **TreeSet<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 [NavigableSet](http://docs.google.com/java/util/NavigableSet.html) implementation based on a [TreeMap](http://docs.google.com/java/util/TreeMap.html). The elements are ordered using 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 guaranteed log(n) time cost for the basic operations (add, remove and contains).

Note that the ordering maintained by a set (whether or not an explicit comparator is provided) must be *consistent with equals* if it is to correctly implement the Set interface. (See Comparable or Comparator for a precise definition of *consistent with equals*.) This is so because the Set interface is defined in terms of the equals operation, but a TreeSet instance performs all element comparisons using its compareTo (or compare) method, so two elements that are deemed equal by this method are, from the standpoint of the set, equal. The behavior of a set *is* well-defined even if its ordering is inconsistent with equals; it just fails to obey the general contract of the Set interface.

**Note that this implementation is not synchronized.** If multiple threads access a tree set concurrently, and at least one of the threads modifies the set, it *must* be synchronized externally. This is typically accomplished by synchronizing on some object that naturally encapsulates the set. If no such object exists, the set should be "wrapped" using the [Collections.synchronizedSortedSet](http://docs.google.com/java/util/Collections.html#synchronizedSortedSet(java.util.SortedSet)) method. This is best done at creation time, to prevent accidental unsynchronized access to the set:

SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));

The iterators returned by this class's iterator method are *fail-fast*: if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a [ConcurrentModificationException](http://docs.google.com/java/util/ConcurrentModificationException.html). Thus, in the face of concurrent modification, the iterator fails quickly and cleanly, rather than risking arbitrary, non-deterministic behavior at an undetermined time in the future.

Note that the fail-fast behavior of an iterator cannot be guaranteed as it is, generally speaking, impossible to make any hard guarantees in the presence of unsynchronized concurrent modification. Fail-fast iterators throw ConcurrentModificationException on a best-effort basis. Therefore, it would be wrong to write a program that depended on this exception for its correctness: *the fail-fast behavior of iterators should be used only to detect bugs.*

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

**Since:** 1.2 **See Also:**[Collection](http://docs.google.com/java/util/Collection.html), [Set](http://docs.google.com/java/util/Set.html), [HashSet](http://docs.google.com/java/util/HashSet.html), [Comparable](http://docs.google.com/java/lang/Comparable.html), [Comparator](http://docs.google.com/java/util/Comparator.html), [TreeMap](http://docs.google.com/java/util/TreeMap.html), [Serialized Form](http://docs.google.com/serialized-form.html#java.util.TreeSet)

| **Constructor Summary** | |
| --- | --- |
| [**TreeSet**](http://docs.google.com/java/util/TreeSet.html#TreeSet())()            Constructs a new, empty tree set, sorted according to the natural ordering of its elements. |
| [**TreeSet**](http://docs.google.com/java/util/TreeSet.html#TreeSet(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/TreeSet.html)> c)            Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements. |
| [**TreeSet**](http://docs.google.com/java/util/TreeSet.html#TreeSet(java.util.Comparator))([Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/TreeSet.html)> comparator)            Constructs a new, empty tree set, sorted according to the specified comparator. |
| [**TreeSet**](http://docs.google.com/java/util/TreeSet.html#TreeSet(java.util.SortedSet))([SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> s)            Constructs a new tree 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/TreeSet.html#add(E))([E](http://docs.google.com/java/util/TreeSet.html) e)            Adds the specified element to this set if it is not already present. |
| boolean | [**addAll**](http://docs.google.com/java/util/TreeSet.html#addAll(java.util.Collection))([Collection](http://docs.google.com/java/util/Collection.html)<? extends [E](http://docs.google.com/java/util/TreeSet.html)> c)            Adds all of the elements in the specified collection to this set. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**ceiling**](http://docs.google.com/java/util/TreeSet.html#ceiling(E))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html#clear())()            Removes all of the elements from this set. |
| [Object](http://docs.google.com/java/lang/Object.html) | [**clone**](http://docs.google.com/java/util/TreeSet.html#clone())()            Returns a shallow copy of this TreeSet instance. |
| [Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/TreeSet.html)> | [**comparator**](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.html)> | [**descendingIterator**](http://docs.google.com/java/util/TreeSet.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/TreeSet.html)> | [**descendingSet**](http://docs.google.com/java/util/TreeSet.html#descendingSet())()            Returns a reverse order view of the elements contained in this set. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**first**](http://docs.google.com/java/util/TreeSet.html#first())()            Returns the first (lowest) element currently in this set. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**floor**](http://docs.google.com/java/util/TreeSet.html#floor(E))([E](http://docs.google.com/java/util/TreeSet.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. |
| [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> | [**headSet**](http://docs.google.com/java/util/TreeSet.html#headSet(E))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html)> | [**headSet**](http://docs.google.com/java/util/TreeSet.html#headSet(E,%20boolean))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html) | [**higher**](http://docs.google.com/java/util/TreeSet.html#higher(E))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.html)> | [**iterator**](http://docs.google.com/java/util/TreeSet.html#iterator())()            Returns an iterator over the elements in this set in ascending order. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**last**](http://docs.google.com/java/util/TreeSet.html#last())()            Returns the last (highest) element currently in this set. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**lower**](http://docs.google.com/java/util/TreeSet.html#lower(E))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html) | [**pollFirst**](http://docs.google.com/java/util/TreeSet.html#pollFirst())()            Retrieves and removes the first (lowest) element, or returns null if this set is empty. |
| [E](http://docs.google.com/java/util/TreeSet.html) | [**pollLast**](http://docs.google.com/java/util/TreeSet.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/TreeSet.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. |
| int | [**size**](http://docs.google.com/java/util/TreeSet.html#size())()            Returns the number of elements in this set (its cardinality). |
| [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> | [**subSet**](http://docs.google.com/java/util/TreeSet.html#subSet(E,%20boolean,%20E,%20boolean))([E](http://docs.google.com/java/util/TreeSet.html) fromElement, boolean fromInclusive, [E](http://docs.google.com/java/util/TreeSet.html) toElement, boolean toInclusive)            Returns a view of the portion of this set whose elements range from fromElement to toElement. |
| [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> | [**subSet**](http://docs.google.com/java/util/TreeSet.html#subSet(E,%20E))([E](http://docs.google.com/java/util/TreeSet.html) fromElement, [E](http://docs.google.com/java/util/TreeSet.html) toElement)            Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive. |
| [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> | [**tailSet**](http://docs.google.com/java/util/TreeSet.html#tailSet(E))([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html)> | [**tailSet**](http://docs.google.com/java/util/TreeSet.html#tailSet(E,%20boolean))([E](http://docs.google.com/java/util/TreeSet.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) |
| --- |
| [equals](http://docs.google.com/java/util/AbstractSet.html#equals(java.lang.Object)), [hashCode](http://docs.google.com/java/util/AbstractSet.html#hashCode()), [removeAll](http://docs.google.com/java/util/AbstractSet.html#removeAll(java.util.Collection)) |

| **Methods inherited from class java.util.**[**AbstractCollection**](http://docs.google.com/java/util/AbstractCollection.html) |
| --- |
| [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) |
| --- |
| [containsAll](http://docs.google.com/java/util/Set.html#containsAll(java.util.Collection)), [equals](http://docs.google.com/java/util/Set.html#equals(java.lang.Object)), [hashCode](http://docs.google.com/java/util/Set.html#hashCode()), [removeAll](http://docs.google.com/java/util/Set.html#removeAll(java.util.Collection)), [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** |
| --- |

### TreeSet

public **TreeSet**()

Constructs a new, empty tree set, sorted according to the natural ordering of its elements. All elements inserted into the set must implement the [Comparable](http://docs.google.com/java/lang/Comparable.html) interface. Furthermore, all such elements must be *mutually comparable*: e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the set. If the user attempts to add an element to the set that violates this constraint (for example, the user attempts to add a string element to a set whose elements are integers), the add call will throw a ClassCastException.

### TreeSet

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

Constructs a new, empty tree set, sorted according to the specified comparator. All elements inserted into the set must be *mutually comparable* by the specified comparator: comparator.compare(e1, e2) must not throw a ClassCastException for any elements e1 and e2 in the set. If the user attempts to add an element to the set that violates this constraint, the add call will throw a ClassCastException.

**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.

### TreeSet

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

Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements. All elements inserted into the set must implement the [Comparable](http://docs.google.com/java/lang/Comparable.html) interface. Furthermore, all such elements must be *mutually comparable*: e1.compareTo(e2) must not throw a ClassCastException for any elements e1 and e2 in the set.

**Parameters:**c - collection whose elements 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 is null

### TreeSet

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

Constructs a new tree 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 is null

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

### iterator

public [Iterator](http://docs.google.com/java/util/Iterator.html)<[E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.html)> **Returns:**an iterator over the elements in this set in descending order**Since:** 1.6

### descendingSet

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

**Description copied from interface:** [**NavigableSet**](http://docs.google.com/java/util/NavigableSet.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. If either set is modified while an iteration over either set is in progress (except through the iterator's own remove operation), the results of the iteration are undefined.

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/TreeSet.html)> **Returns:**a reverse order view of this set**Since:** 1.6

### size

public int **size**()

Returns the number of elements in this set (its cardinality).

**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/TreeSet.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/TreeSet.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/TreeSet.html)> **Returns:**the number of elements in this set (its cardinality)

### 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/TreeSet.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/TreeSet.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/TreeSet.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==null ? e==null : 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/TreeSet.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/TreeSet.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/TreeSet.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 object 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 and this set uses natural ordering, or its comparator does not permit null elements

### add

public boolean **add**([E](http://docs.google.com/java/util/TreeSet.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==null ? e2==null : 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/TreeSet.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/TreeSet.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/TreeSet.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 the specified object 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 and this set uses natural ordering, or its comparator does not permit null elements

### 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==null ? e==null : 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/TreeSet.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/TreeSet.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/TreeSet.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 the specified object 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 and this set uses natural ordering, or its comparator does not permit null elements

### clear

public void **clear**()

Removes all of the elements from this set. The set will be empty after this call returns.

**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/TreeSet.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/TreeSet.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/TreeSet.html)>

### addAll

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

Adds all of the elements in the specified collection to this set.

**Specified by:**[addAll](http://docs.google.com/java/util/Collection.html#addAll(java.util.Collection)) in interface [Collection](http://docs.google.com/java/util/Collection.html)<[E](http://docs.google.com/java/util/TreeSet.html)>**Specified by:**[addAll](http://docs.google.com/java/util/Set.html#addAll(java.util.Collection)) in interface [Set](http://docs.google.com/java/util/Set.html)<[E](http://docs.google.com/java/util/TreeSet.html)>**Overrides:**[addAll](http://docs.google.com/java/util/AbstractCollection.html#addAll(java.util.Collection)) in class [AbstractCollection](http://docs.google.com/java/util/AbstractCollection.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **Parameters:**c - collection containing elements to be added to 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 elements provided cannot be compared with the elements currently in the set [NullPointerException](http://docs.google.com/java/lang/NullPointerException.html) - if the specified collection is null or if any element is null and this set uses natural ordering, or its comparator does not permit null elements**See Also:**[AbstractCollection.add(Object)](http://docs.google.com/java/util/AbstractCollection.html#add(E))

### subSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **subSet**([E](http://docs.google.com/java/util/TreeSet.html) fromElement,  
 boolean fromInclusive,  
 [E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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.**Since:** 1.6

### headSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **headSet**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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**Since:** 1.6

### tailSet

public [NavigableSet](http://docs.google.com/java/util/NavigableSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **tailSet**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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**Since:** 1.6

### subSet

public [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **subSet**([E](http://docs.google.com/java/util/TreeSet.html) fromElement,  
 [E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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 [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **headSet**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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 [SortedSet](http://docs.google.com/java/util/SortedSet.html)<[E](http://docs.google.com/java/util/TreeSet.html)> **tailSet**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements [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

### comparator

public [Comparator](http://docs.google.com/java/util/Comparator.html)<? super [E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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/TreeSet.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

### lower

public [E](http://docs.google.com/java/util/TreeSet.html) **lower**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements**Since:** 1.6

### floor

public [E](http://docs.google.com/java/util/TreeSet.html) **floor**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements**Since:** 1.6

### ceiling

public [E](http://docs.google.com/java/util/TreeSet.html) **ceiling**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements**Since:** 1.6

### higher

public [E](http://docs.google.com/java/util/TreeSet.html) **higher**([E](http://docs.google.com/java/util/TreeSet.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/TreeSet.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 and this set uses natural ordering, or its comparator does not permit null elements**Since:** 1.6

### pollFirst

public [E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html)> **Returns:**the first element, or null if this set is empty**Since:** 1.6

### pollLast

public [E](http://docs.google.com/java/util/TreeSet.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/TreeSet.html)> **Returns:**the last element, or null if this set is empty**Since:** 1.6

### clone

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

Returns a shallow copy of this TreeSet 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)

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/TreeSet.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/TreeMap.html)   [**NEXT CLASS**](http://docs.google.com/java/util/UnknownFormatConversionException.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/TreeSet.html)    [**NO FRAMES**](http://docs.google.com/TreeSet.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).