java.util

Class TreeSet<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractSet<E>
      java.util.TreeSet<E>

Type Parameters:

`E` - the type of elements maintained by this set

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>

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

A `NavigableSet` implementation based on a `TreeMap`. The elements are ordered using their natural ordering, or by a `Comparator` 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` 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`. 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.

Since:

1.2

See Also:

`Collection`, `Set`, `HashSet`, `Comparable`, `Comparator`, `TreeMap`, Serialized Form

Constructor Summary

Constructors

Constructor and Description
TreeSet() Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
TreeSet(**Collection**<? extends **E**> c) Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
TreeSet(**Comparator**<? super **E**> comparator) Constructs a new, empty tree set, sorted according to the specified comparator.
TreeSet(**SortedSet**<**E**> s) Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Adds the specified element to this set if it is not already present.
boolean	**addAll**(**Collection**<? extends **E**> c) Adds all of the elements in the specified collection to this set.
E	**ceiling**(**E** e) Returns the least element in this set greater than or equal to the given element, or `null` if there is no such element.
void	**clear**() Removes all of the elements from this set.
Object	**clone**() Returns a shallow copy of this `TreeSet` instance.
Comparator<? super **E**>	**comparator**() Returns the comparator used to order the elements in this set, or `null` if this set uses the **natural ordering** of its elements.
boolean	**contains**(**Object** o) Returns `true` if this set contains the specified element.
Iterator<**E**>	**descendingIterator**() Returns an iterator over the elements in this set in descending order.
NavigableSet<**E**>	**descendingSet**() Returns a reverse order view of the elements contained in this set.
E	**first**() Returns the first (lowest) element currently in this set.
E	**floor**(**E** e) Returns the greatest element in this set less than or equal to the given element, or `null` if there is no such element.
SortedSet<**E**>	**headSet**(**E** toElement) Returns a view of the portion of this set whose elements are strictly less than `toElement`.
NavigableSet<**E**>	**headSet**(**E** toElement, boolean inclusive) Returns a view of the portion of this set whose elements are less than (or equal to, if `inclusive` is true) `toElement`.
E	**higher**(**E** e) Returns the least element in this set strictly greater than the given element, or `null` if there is no such element.

boolean	**isEmpty**() Returns true if this set contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this set in ascending order.
E	**last**() Returns the last (highest) element currently in this set.
E	**lower**(E e) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	**pollFirst**() Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	**pollLast**() Retrieves and removes the last (highest) element, or returns null if this set is empty.
boolean	**remove**(Object o) Removes the specified element from this set if it is present.
int	**size**() Returns the number of elements in this set (its cardinality).
NavigableSet<E>	**subSet**(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) Returns a view of the portion of this set whose elements range from fromElement to toElement.
SortedSet<E>	**subSet**(E fromElement, E toElement) Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
SortedSet<E>	**tailSet**(E fromElement) Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
NavigableSet<E>	**tailSet**(E fromElement, boolean inclusive) Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

Methods inherited from class java.util.AbstractSet

equals, hashCode, removeAll

Methods inherited from class java.util.AbstractCollection

containsAll, retainAll, toArray, toArray, toString

Methods inherited from class java.lang.Object

finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.Set

containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

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` 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<? super E> 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 of the elements will be used.

TreeSet

```
public TreeSet(Collection<? extends E> 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` 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` - if the elements in `c` are not `Comparable`, or are not mutually comparable

`NullPointerException` - if the specified collection is null

TreeSet

```
public TreeSet(SortedSet<E> 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` - if the specified sorted set is null

Method Detail

iterator

```
public Iterator<E> iterator()
```

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

Specified by:

`iterator` in interface `Iterable<E>`

Specified by:

`iterator` in interface `Collection<E>`

Specified by:

`iterator` in interface `NavigableSet<E>`

Specified by:

`iterator` in interface `Set<E>`

Specified by:

`iterator` in class `AbstractCollection<E>`

Returns:

an iterator over the elements in this set in ascending order

descendingIterator

```
public Iterator<E> descendingIterator()
```

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

Specified by:

`descendingIterator` in interface `NavigableSet<E>`

Returns:

an iterator over the elements in this set in descending order

Since:

1.6

descendingSet

```
public NavigableSet<E> descendingSet()
```

Description copied from interface: `NavigableSet`
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(comparator())`. The expression `s.descendingSet().descendingSet()` returns a view of `s` essentially equivalent to `s`.

Specified by:

`descendingSet` in interface `NavigableSet<E>`

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` in interface `Collection<E>`

Specified by:

`size` in interface `Set<E>`

Specified by:

`size` in class `AbstractCollection<E>`

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` in interface `Collection<E>`

Specified by:

`isEmpty` in interface `Set<E>`

Overrides:

`isEmpty` in class `AbstractCollection<E>`

Returns:

`true` if this set contains no elements

contains

```
public boolean contains(Object 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` in interface `Collection<E>`

Specified by:

`contains` in interface `Set<E>`

Overrides:

`contains` in class `AbstractCollection<E>`

Parameters:

`o` - object to be checked for containment in this set

Returns:

`true` if this set contains the specified element

Throws:

`ClassCastException` - if the specified object cannot be compared with the elements currently in the set

`NullPointerException` - 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 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` in interface `Collection<E>`

Specified by:

`add` in interface `Set<E>`

Overrides:

`add` in class `AbstractCollection<E>`

Parameters:

`e` - element to be added to this set

Returns:

`true` if this set did not already contain the specified element

Throws:

`ClassCastException` - if the specified object cannot be compared with the elements currently in this set

`NullPointerException` - 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 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` in interface `Collection<E>`

Specified by:

`remove` in interface `Set<E>`

Overrides:

`remove` in class `AbstractCollection<E>`

Parameters:

`o` - object to be removed from this set, if present

Returns:

`true` if this set contained the specified element

Throws:

`ClassCastException` - if the specified object cannot be compared with the elements currently in this set

`NullPointerException` - 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` in interface `Collection<E>`

Specified by:

`clear` in interface `Set<E>`

Overrides:

`clear` in class `AbstractCollection<E>`

addAll

```
public boolean addAll(Collection<? extends E> c)
```

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

Specified by:

`addAll` in interface `Collection<E>`

Specified by:

`addAll` in interface `Set<E>`

Overrides:

`addAll` in class `AbstractCollection<E>`

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` - if the elements provided cannot be compared with the elements currently in the set

`NullPointerException` - 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)`

subSet

```
public NavigableSet<E> subSet(E fromElement,
                      boolean fromInclusive,
                      E toElement,
                      boolean toInclusive)
```

Description copied from interface: NavigableSet

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 `fromInclusive` and `toInclusive` 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` in interface `NavigableSet<E>`

Parameters:

`fromElement` - low endpoint of the returned set

`fromInclusive` - `true` if the low endpoint is to be included in the returned view

`toElement` - high endpoint of the returned set

`toInclusive` - `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` - 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` - if `fromElement` or `toElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - 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<E> headSet(E toElement,
                       boolean inclusive)
```

Description copied from interface: NavigableSet

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` in interface `NavigableSet<E>`

Parameters:

`toElement` - high endpoint of the returned set

`inclusive` - `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` - if `toElement` is not compatible with this set's comparator (or, if the set has no comparator, if `toElement` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `toElement` cannot be compared to elements currently in the set.

`NullPointerException` - if `toElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - if this set itself has a restricted range, and `toElement` lies outside the bounds of the range

Since:

1.6

tailSet

```
public NavigableSet<E> tailSet(E fromElement,
                               boolean inclusive)
```

Description copied from interface: `NavigableSet`

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` in interface `NavigableSet<E>`

Parameters:

`fromElement` - low endpoint of the returned set

`inclusive` - `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` - if `fromElement` is not compatible with this set's comparator (or, if the set has no comparator, if `fromElement` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `fromElement` cannot be compared to elements currently in the set.

`NullPointerException` - if `fromElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - if this set itself has a restricted range, and `fromElement` lies outside the bounds of the range

Since:

1.6

subSet

```
public SortedSet<E> subSet(E fromElement,
                           E toElement)
```

Description copied from interface: `NavigableSet`

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` in interface `NavigableSet<E>`

Specified by:

`subSet` in interface `SortedSet<E>`

Parameters:

`fromElement` - low endpoint (inclusive) of the returned set

`toElement` - 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` - 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` - if `fromElement` or `toElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - 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<E> headSet(E toElement)
```

Description copied from interface: `NavigableSet`

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` in interface `NavigableSet<E>`

Specified by:

`headSet` in interface `SortedSet<E>`

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` - if `toElement` is not compatible with this set's comparator (or, if the set has no comparator, if `toElement` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `toElement` cannot be compared to elements currently in the set.

`NullPointerException` - if `toElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - if this set itself has a restricted range, and `toElement` lies outside the bounds of the range na

tailSet

```
public SortedSet<E> tailSet(E fromElement)
```

Description copied from interface: `NavigableSet`
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` in interface `NavigableSet<E>`

Specified by:

`tailSet` in interface `SortedSet<E>`

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` - if `fromElement` is not compatible with this set's comparator (or, if the set has no comparator, if `fromElement` does not implement `Comparable`). Implementations may, but are not required to, throw this exception if `fromElement` cannot be compared to elements currently in the set.

`NullPointerException` - if `fromElement` is null and this set uses natural ordering, or its comparator does not permit null elements

`IllegalArgumentException` - if this set itself has a restricted range, and `fromElement` lies outside the bounds of the range

comparator

```
public Comparator<? super E> comparator()
```

Description copied from interface: `SortedSet`
Returns the comparator used to order the elements in this set, or `null` if this set uses the natural ordering of its elements.

Specified by:

`comparator` in interface `SortedSet<E>`

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 first()
```

Description copied from interface: `SortedSet`
Returns the first (lowest) element currently in this set.

Specified by:

`first` in interface `SortedSet<E>`

Returns:

the first (lowest) element currently in this set

Throws:

`NoSuchElementException` - if this set is empty

last

```
public E last()
```

Description copied from interface: `SortedSet`
Returns the last (highest) element currently in this set.

Specified by:

`last` in interface `SortedSet<E>`

Returns:

the last (highest) element currently in this set

Throws:

`NoSuchElementException` - if this set is empty

lower

```
public E lower(E e)
```

Description copied from interface: `NavigableSet`
Returns the greatest element in this set strictly less than the given element, or `null` if there is no such element.

Specified by:

`lower` in interface `NavigableSet<E>`

Parameters:

`e` - the value to match

Returns:

the greatest element less than `e`, or `null` if there is no such element

Throws:

`ClassCastException` - if the specified element cannot be compared with the elements currently in the set

`NullPointerException` - 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 floor(E e)
```

Description copied from interface: `NavigableSet`

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` in interface `NavigableSet<E>`

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` - if the specified element cannot be compared with the elements currently in the set

`NullPointerException` - 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 ceiling(E e)
```

Description copied from interface: `NavigableSet`
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` in interface `NavigableSet<E>`

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` - if the specified element cannot be compared with the elements currently in the set

`NullPointerException` - 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 higher(E e)
```

Description copied from interface: `NavigableSet`
Returns the least element in this set strictly greater than the given element, or `null` if there is no such element.

Specified by:

`higher` in interface `NavigableSet<E>`

Parameters:

`e` - the value to match

Returns:

the least element greater than `e`, or `null` if there is no such element

Throws:

`ClassCastException` - if the specified element cannot be compared with the elements currently in the set

`NullPointerException` - 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 pollFirst()
```

Description copied from interface: `NavigableSet`
Retrieves and removes the first (lowest) element, or returns `null` if this set is empty.

Specified by:

`pollFirst` in interface `NavigableSet<E>`

Returns:

the first element, or `null` if this set is empty

Since:

1.6

pollLast

```
public E pollLast()
```

Description copied from interface: `NavigableSet`
Retrieves and removes the last (highest) element, or returns `null` if this set is empty.

Specified by:

`pollLast` in interface `NavigableSet<E>`

Returns:

the last element, or `null` if this set is empty

Since:

1.6

clone

```
public Object clone()
```

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

Overrides:

`clone` in class `Object`

Returns:

a shallow copy of this set

See Also:

`Cloneable`