java.util.TreeSet.java Source code

Java tutorial

Introduction

Here is the source code for java.util.TreeSet.java

Source

```
/*
 * Copyright (c) 1998, 2018, Oracle and/or its affiliates. All rights reserved.
 * DO NOT ALTER OR REMOVE COPYRIGHT NOTICES OR THIS FILE HEADER.
 *
 * This code is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 only, as
 * published by the Free Software Foundation.  Oracle designates this
 * particular file as subject to the "Classpath" exception as provided
 * by Oracle in the LICENSE file that accompanied this code.
 *
 * This code is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
 * FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
 * version 2 for more details (a copy is included in the LICENSE file that
 * accompanied this code).
 *
 * You should have received a copy of the GNU General Public License version
 * 2 along with this work; if not, write to the Free Software Foundation,
 * Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA.
 *
 * Please contact Oracle, 500 Oracle Parkway, Redwood Shores, CA 94065 USA
 * or visit www.oracle.com if you need additional information or have any
 * questions.
 */

package java.util;

/**
 * A {@link NavigableSet} implementation based on a {@link TreeMap}.
 * The elements are ordered using their {@linkplain Comparable natural
 * ordering}, or by a {@link Comparator} provided at set creation
 * time, depending on which constructor is used.
 *
 * <p>This implementation provides guaranteed log(n) time cost for the basic
 * operations ({@code add}, {@code remove} and {@code contains}).
 *
 * <p>Note that the ordering maintained by a set (whether or not an explicit
 * comparator is provided) must be <i>consistent with equals</i> if it is to
 * correctly implement the {@code Set} interface.  (See {@code Comparable}
 * or {@code Comparator} for a precise definition of <i>consistent with
 * equals</i>.)  This is so because the {@code Set} interface is defined in
 * terms of the {@code equals} operation, but a {@code TreeSet} instance
 * performs all element comparisons using its {@code compareTo} (or
 * {@code 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
 * <i>is</i> well-defined even if its ordering is inconsistent with equals; it
 * just fails to obey the general contract of the {@code Set} interface.
 *
 * <p><strong>Note that this implementation is not synchronized.</strong>
 * If multiple threads access a tree set concurrently, and at least one
 * of the threads modifies the set, it <i>must</i> 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
 * {@link Collections#synchronizedSortedSet Collections.synchronizedSortedSet}
 * method.  This is best done at creation time, to prevent accidental
 * unsynchronized access to the set: <pre>
 *   SortedSet s = Collections.synchronizedSortedSet(new TreeSet(...));</pre>
 *
 * <p>The iterators returned by this class's {@code iterator} method are
 * <i>fail-fast</i>: if the set is modified at any time after the iterator is
 * created, in any way except through the iterator's own {@code remove}
 * method, the iterator will throw a {@link 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.
 *
 * <p>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 {@code ConcurrentModificationException} on a best-effort basis.
 * Therefore, it would be wrong to write a program that depended on this
 * exception for its correctness:   <i>the fail-fast behavior of iterators
 * should be used only to detect bugs.</i>
 *
 * <p>This class is a member of the
 * <a href="{@docRoot}/java.base/java/util/package-summary.html#CollectionsFramework">
 * Java Collections Framework</a>.
 *
 * @param <E> the type of elements maintained by this set
 *
```

```
 * @author  Josh Bloch
 * @see     Collection
 * @see     Set
 * @see     HashSet
 * @see     Comparable
 * @see     Comparator
 * @see     TreeMap
 * @since   1.2
 */

public class TreeSet<E> extends AbstractSet (abstractset-75d9b.html)<E> implements NavigableSet (navigableset-473f8.html)<E>, Cloneable (../lang/
    /**
     * The backing map.
     */
    private transient NavigableMap (navigablemap-523de.html)<E, Object (../lang/object-b3165.html)> m;

    // Dummy value to associate with an Object in the backing Map
    private static final Object (../lang/object-b3165.html) PRESENT = new Object (../lang/object-b3165.html)();

    /**
     * Constructs a set backed by the specified navigable map.
     */
    TreeSet(NavigableMap (navigablemap-523de.html)<E, Object (../lang/object-b3165.html)> m) {
        this.m = m;
    }

    /**
     * Constructs a new, empty tree set, sorted according to the
     * natural ordering of its elements.  All elements inserted into
     * the set must implement the {@link Comparable} interface.
     * Furthermore, all such elements must be <i>mutually
     * comparable</i>: {@code e1.compareTo(e2)} must not throw a
     * {@code ClassCastException} for any elements {@code e1} and
     * {@code 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 {@code add} call will throw a
     * {@code ClassCastException}.
     */
    public TreeSet() {
        this(new TreeMap (treemap-d0d44.html)<>());
    }

    /**
     * Constructs a new, empty tree set, sorted according to the specified
     * comparator.  All elements inserted into the set must be <i>mutually
     * comparable</i> by the specified comparator: {@code comparator.compare(e1,
     * e2)} must not throw a {@code ClassCastException} for any elements
     * {@code e1} and {@code e2} in the set.  If the user attempts to add
     * an element to the set that violates this constraint, the
     * {@code add} call will throw a {@code ClassCastException}.
     *
     * @param comparator the comparator that will be used to order this set.
     *        If {@code null}, the {@linkplain Comparable natural
     *        ordering} of the elements will be used.
     */
    public TreeSet(Comparator (comparator-1035c.html)<? super E> comparator) {
        this(new TreeMap (treemap-d0d44.html)<>(comparator));
    }

    /**
     * Constructs a new tree set containing the elements in the specified
     * collection, sorted according to the <i>natural ordering</i> of its
     * elements.  All elements inserted into the set must implement the
     * {@link Comparable} interface.  Furthermore, all such elements must be
     * <i>mutually comparable</i>: {@code e1.compareTo(e2)} must not throw a
     * {@code ClassCastException} for any elements {@code e1} and
     * {@code e2} in the set.
     *
     * @param c collection whose elements will comprise the new set
     * @throws ClassCastException if the elements in {@code c} are
     *         not {@link Comparable}, or are not mutually comparable
     * @throws NullPointerException if the specified collection is null
     */
    public TreeSet(Collection (collection-d7b1c.html)<? extends E> c) {
        this();
        addAll(c);
    }

    /**
     * Constructs a new tree set containing the same elements and
     * using the same ordering as the specified sorted set.
     *
     * @param s sorted set whose elements will comprise the new set
```

```
     * @throws NullPointerException if the specified sorted set is null
     */
    public TreeSet(SortedSet (sortedset-3c6cf.html)<E> s) {
        this(s.comparator (sortedset-3c6cf.html#52b4bb5f5391101361606352104105e4)());
        addAll(s);
    }

    /**
     * Returns an iterator over the elements in this set in ascending order.
     *
     * @return an iterator over the elements in this set in ascending order
     */
    public Iterator (iterator-63ad8.html)<E> iterator() {
        return m.navigableKeySet (navigablemap-523de.html#512ff95c797cc6a0678f6288d77d9170)().iterator();
    }

    /**
     * Returns an iterator over the elements in this set in descending order.
     *
     * @return an iterator over the elements in this set in descending order
     * @since 1.6
     */
    public Iterator (iterator-63ad8.html)<E> descendingIterator() {
        return m.descendingKeySet (navigablemap-523de.html#a66a929bd0ce0f472bbb3fae0d819557)().iterator();
    }

    /**
     * @since 1.6
     */
    public NavigableSet (navigableset-473f8.html)<E> descendingSet() {
        return new TreeSet<>(m.descendingMap (navigablemap-523de.html#bb1595bf6837629fb1fdfe4e8501e3a7)());
    }

    /**
     * Returns the number of elements in this set (its cardinality).
     *
     * @return the number of elements in this set (its cardinality)
     */
    public int size() {
        return m.size (navigablemap-523de.html#e36e7f44989ed396354b4696a3efc096)();
    }

    /**
     * Returns {@code true} if this set contains no elements.
     *
     * @return {@code true} if this set contains no elements
     */
    public boolean isEmpty() {
        return m.isEmpty (navigablemap-523de.html#afa99f6913e9b76b736f5e43b1edc98e)();
    }

    /**
     * Returns {@code true} if this set contains the specified element.
     * More formally, returns {@code true} if and only if this set
     * contains an element {@code e} such that
     * {@code Objects.equals(o, e)}.
     *
     * @param o object to be checked for containment in this set
     * @return {@code true} if this set contains the specified element
     * @throws ClassCastException if the specified object cannot be compared
     *         with the elements currently in the set
     * @throws NullPointerException if the specified element is null
     *         and this set uses natural ordering, or its comparator
     *         does not permit null elements
     */
    public boolean contains(Object (../lang/object-b3165.html) o) {
        return m.containsKey (navigablemap-523de.html#a012a53656fb2ca97b80ba7922275aab)(o);
    }

    /**
     * Adds the specified element to this set if it is not already present.
     * More formally, adds the specified element {@code e} to this set if
     * the set contains no element {@code e2} such that
     * {@code Objects.equals(e, e2)}.
     * If this set already contains the element, the call leaves the set
     * unchanged and returns {@code false}.
     *
     * @param e element to be added to this set
     * @return {@code 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
     * @throws NullPointerException if the specified element is null
     *         and this set uses natural ordering, or its comparator
```

```
     *         does not permit null elements
     */
    public boolean add(E e) {
        return m.put (navigablemap-523de.html#227079bc1a5e35f1afec175b38ad9588)(e, PRESENT) == null;
    }

    /**
     * Removes the specified element from this set if it is present.
     * More formally, removes an element {@code e} such that
     * {@code Objects.equals(o, e)},
     * if this set contains such an element.  Returns {@code 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.)
     *
     * @param o object to be removed from this set, if present
     * @return {@code true} if this set contained the specified element
     * @throws ClassCastException if the specified object cannot be compared
     *         with the elements currently in this set
     * @throws NullPointerException if the specified element is null
     *         and this set uses natural ordering, or its comparator
     *         does not permit null elements
     */
    public boolean remove(Object (../lang/object-b3165.html) o) {
        return m.remove (navigablemap-523de.html#da9956407dd90dee5b71dd3b6affe075)(o) == PRESENT;
    }

    /**
     * Removes all of the elements from this set.
     * The set will be empty after this call returns.
     */
    public void clear() {
        m.clear (navigablemap-523de.html#7cd83ad44a3b851cbd057a236d6bd823)();
    }

    /**
     * Adds all of the elements in the specified collection to this set.
     *
     * @param c collection containing elements to be added to this set
     * @return {@code 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
     * @throws 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
     */
    public boolean addAll(Collection (collection-d7b1c.html)<? extends E> c) {
        // Use linear-time version if applicable
        if (m.size (navigablemap-523de.html#e36e7f44989ed396354b4696a3efc096)() == 0 && c.size (collection-d7b1c.html#eff254f53227b478ecd1992eba9
            SortedSet (sortedset-3c6cf.html)<? extends E> set = (SortedSet (sortedset-3c6cf.html)<? extends E>) c;
            TreeMap (treemap-d0d44.html)<E, Object (../lang/object-b3165.html)> map = (TreeMap (treemap-d0d44.html)<E, Object (../lang/object-b31
            Comparator (comparator-1035c.html)<?> cc = set.comparator (sortedset-3c6cf.html#52b4bb5f5391101361606352104105e4)();
            Comparator (comparator-1035c.html)<? super E> mc = map.comparator (treemap-d0d44.html#eb89387299ff5b61a1917771f6b33646)();
            if (cc == mc || (cc != null && cc.equals (comparator-1035c.html#b9c67834df26cf007e60d584c0127f81)(mc))) {
                map.addAllForTreeSet (treemap-d0d44.html#2ac156743d811f4a58b983b5af829189)(set, PRESENT);
                return true;
            }
        }
        return super.addAll(c);
    }

    /**
     * @throws ClassCastException {@inheritDoc}
     * @throws NullPointerException if {@code fromElement} or {@code toElement}
     *         is null and this set uses natural ordering, or its comparator
     *         does not permit null elements
     * @throws IllegalArgumentException {@inheritDoc}
     * @since 1.6
     */
    public NavigableSet (navigableset-473f8.html)<E> subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) {
        return new TreeSet<>(m.subMap (navigablemap-523de.html#6d6e026e85339a55d565b5bb52a39ceb)(fromElement, fromInclusive, toElement, toInclusi
    }

    /**
     * @throws ClassCastException {@inheritDoc}
     * @throws NullPointerException if {@code toElement} is null and
     *         this set uses natural ordering, or its comparator does
     *         not permit null elements
     * @throws IllegalArgumentException {@inheritDoc}
     * @since 1.6
     */
    public NavigableSet (navigableset-473f8.html)<E> headSet(E toElement, boolean inclusive) {
        return new TreeSet<>(m.headMap (navigablemap-523de.html#5a730569d7c6ccfc606ab84b3a2bc1dd)(toElement, inclusive));
    }
```

```
/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if {@code fromElement} is null and
 *         this set uses natural ordering, or its comparator does
 *         not permit null elements
 * @throws IllegalArgumentException {@inheritDoc}
 * @since 1.6
 */
public NavigableSet (navigableset-473f8.html)<E> tailSet(E fromElement, boolean inclusive) {
    return new TreeSet<>(m.tailMap (navigablemap-523de.html#dfce6736543068d5cea491bda88d667b)(fromElement, inclusive));
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if {@code fromElement} or
 *         {@code toElement} is null and this set uses natural ordering,
 *         or its comparator does not permit null elements
 * @throws IllegalArgumentException {@inheritDoc}
 */
public SortedSet (sortedset-3c6cf.html)<E> subSet(E fromElement, E toElement) {
    return subSet(fromElement, true, toElement, false);
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if {@code toElement} is null
 *         and this set uses natural ordering, or its comparator does
 *         not permit null elements
 * @throws IllegalArgumentException {@inheritDoc}
 */
public SortedSet (sortedset-3c6cf.html)<E> headSet(E toElement) {
    return headSet(toElement, false);
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if {@code fromElement} is null
 *         and this set uses natural ordering, or its comparator does
 *         not permit null elements
 * @throws IllegalArgumentException {@inheritDoc}
 */
public SortedSet (sortedset-3c6cf.html)<E> tailSet(E fromElement) {
    return tailSet(fromElement, true);
}

public Comparator (comparator-1035c.html)<? super E> comparator() {
    return m.comparator (navigablemap-523de.html#6ae889adbd697c4abdabfd2f71306375)();
}

/**
 * @throws NoSuchElementException {@inheritDoc}
 */
public E first() {
    return m.firstKey (navigablemap-523de.html#0ecbff4f13d4d580a45db86af52e0d9a)();
}

/**
 * @throws NoSuchElementException {@inheritDoc}
 */
public E last() {
    return m.lastKey (navigablemap-523de.html#699547ad6c74014d9f9eca45a2e72461)();
}

// NavigableSet API methods

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if the specified element is null
 *         and this set uses natural ordering, or its comparator
 *         does not permit null elements
 * @since 1.6
 */
public E lower(E e) {
    return m.lowerKey (navigablemap-523de.html#c5c212739e7c2037ca58d5283303a76a)(e);
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if the specified element is null
 *         and this set uses natural ordering, or its comparator
 *         does not permit null elements
 * @since 1.6
 */
```

```
public E floor(E e) {
    return m.floorKey (navigablemap-523de.html#ab7fc8a15520be3d5ce65beb427f6e65)(e);
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if the specified element is null
 *         and this set uses natural ordering, or its comparator
 *         does not permit null elements
 * @since 1.6
 */
public E ceiling(E e) {
    return m.ceilingKey (navigablemap-523de.html#2dc91eef9201cad7097b7bf0f26a3a39)(e);
}

/**
 * @throws ClassCastException {@inheritDoc}
 * @throws NullPointerException if the specified element is null
 *         and this set uses natural ordering, or its comparator
 *         does not permit null elements
 * @since 1.6
 */
public E higher(E e) {
    return m.higherKey (navigablemap-523de.html#59d1ba111228b4317b3de5aca7e5d89a)(e);
}

/**
 * @since 1.6
 */
public E pollFirst() {
    Map.Entry (map-1765c.html)<E, ?> e = m.pollFirstEntry (navigablemap-523de.html#a84e8f8fd586516e484b0ea2873ecd18)();
    return (e == null) ? null : e.getKey();
}

/**
 * @since 1.6
 */
public E pollLast() {
    Map.Entry (map-1765c.html)<E, ?> e = m.pollLastEntry (navigablemap-523de.html#77bfc9671ec2e45105628dae953cfaf7)();
    return (e == null) ? null : e.getKey();
}

/**
 * Returns a shallow copy of this {@code TreeSet} instance. (The elements
 * themselves are not cloned.)
 *
 * @return a shallow copy of this set
 */
@SuppressWarnings("unchecked")
public Object (../lang/object-b3165.html) clone() {
    TreeSet<E> clone;
    try {
        clone = (TreeSet<E>) super.clone();
    } catch (CloneNotSupportedException (../lang/clonenotsupportedexception-081de.html) e) {
        throw new InternalError (../lang/internalerror-29b35.html)(e);
    }

    clone.m = new TreeMap (treemap-d0d44.html)<>(m);
    return clone;
}

/**
 * Save the state of the {@code TreeSet} instance to a stream (that is,
 * serialize it).
 *
 * @serialData Emits the comparator used to order this set, or
 *             {@code null} if it obeys its elements' natural ordering
 *             (Object), followed by the size of the set (the number of
 *             elements it contains) (int), followed by all of its
 *             elements (each an Object) in order (as determined by the
 *             set's Comparator, or by the elements' natural ordering if
 *             the set has no Comparator).
 */
private void writeObject(java.io.ObjectOutputStream s) throws java.io.IOException {
    // Write out any hidden stuff
    s.defaultWriteObject();

    // Write out Comparator
    s.writeObject(m.comparator (navigablemap-523de.html#6ae889adbd697c4abdabfd2f71306375)());

    // Write out size
    s.writeInt(m.size (navigablemap-523de.html#e36e7f44989ed396354b4696a3efc096)());

    // Write out all elements in the proper order.
```

```
        for (E e : m.keySet (navigablemap-523de.html#5792a7190e0858016d2ba31945a68875)())
            s.writeObject(e);
    }

    /**
     * Reconstitute the {@code TreeSet} instance from a stream (that is,
     * deserialize it).
     */
    private void readObject(java.io.ObjectInputStream s) throws java.io.IOException, ClassNotFoundException (../lang/classnotfoundexception-20a5c
        // Read in any hidden stuff
        s.defaultReadObject();

        // Read in Comparator
        @SuppressWarnings("unchecked")
        Comparator (comparator-1035c.html)<? super E> c = (Comparator (comparator-1035c.html)<? super E>) s.readObject();

        // Create backing TreeMap
        TreeMap (treemap-d0d44.html)<E, Object (../lang/object-b3165.html)> tm = new TreeMap (treemap-d0d44.html)<>(c);
        m = tm;

        // Read in size
        int size = s.readInt();

        tm.readTreeSet (treemap-d0d44.html#54d55aee7d475cb6fdba9c9f1479bba1)(size, s, PRESENT);
    }

    /**
     * Creates a <em><a href="Spliterator.html#binding">late-binding</a></em>
     * and <em>fail-fast</em> {@link Spliterator} over the elements in this
     * set.
     *
     * <p>The {@code Spliterator} reports {@link Spliterator#SIZED},
     * {@link Spliterator#DISTINCT}, {@link Spliterator#SORTED}, and
     * {@link Spliterator#ORDERED}.  Overriding implementations should document
     * the reporting of additional characteristic values.
     *
     * <p>The spliterator's comparator (see
     * {@link java.util.Spliterator#getComparator()}) is {@code null} if
     * the tree set's comparator (see {@link #comparator()}) is {@code null}.
     * Otherwise, the spliterator's comparator is the same as or imposes the
     * same total ordering as the tree set's comparator.
     *
     * @return a {@code Spliterator} over the elements in this set
     * @since 1.8
     */
    public Spliterator (spliterator-60016.html)<E> spliterator() {
        return TreeMap.keySpliteratorFor (treemap-d0d44.html#44c57ffce7af32363fc06bdbfb92eaf0)(m);
    }

    private static final long serialVersionUID = -2479143000061671589L;
}
```