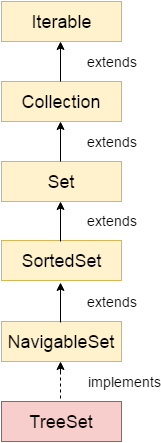
Java TreeSet class



Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

The important points about Java TreeSet class are:

* Java TreeSet class contains unique elements only like HashSet.
* Java TreeSet class access and retrieval times are quiet fast.
* Java TreeSet class doesn't allow null element.
* Java TreeSet class is non synchronized.
* Java TreeSet class maintains ascending order.

Hierarchy of TreeSet class

As shown in the above diagram, Java TreeSet class implements the NavigableSet interface. The NavigableSet interface extends SortedSet, Set, Collection and Iterable interfaces in hierarchical order.

TreeSet class declaration

Let's see the declaration for java.util.TreeSet class.

1. **public** **class** TreeSet<E> **extends** AbstractSet<E> **implements** NavigableSet<E>, Cloneable, Serializable

Constructors of Java TreeSet class

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| TreeSet() | It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set. |
| TreeSet(Collection<? extends E> c) | It is used to build a new tree set that contains the elements of the collection c. |
| TreeSet(Comparator<? super E> comparator) | It is used to construct an empty tree set that will be sorted according to given comparator. |
| TreeSet(SortedSet<E> s) | It is used to build a TreeSet that contains the elements of the given SortedSet. |

Methods of Java TreeSet class

|  |  |
| --- | --- |
| **Method** | **Description** |
| boolean add(E e) | It is used to add the specified element to this set if it is not already present. |
| boolean addAll(Collection<? extends E> c) | It is used to add all of the elements in the specified collection to this set. |
| E ceiling(E e) | It returns the equal or closest greatest element of the specified element from the set, or null there is no such element. |
| Comparator<? super E> comparator() | It returns comparator that arranged elements in order. |
| Iterator descendingIterator() | It is used iterate the elements in descending order. |
| NavigableSet descendingSet() | It returns the elements in reverse order. |
| E floor(E e) | It returns the equal or closest least element of the specified element from the set, or null there is no such element. |
| SortedSet headSet(E toElement) | It returns the group of elements that are less than the specified element. |
| NavigableSet headSet(E toElement, boolean inclusive) | It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element. |
| E higher(E e) | It returns the closest greatest element of the specified element from the set, or null there is no such element. |
| Iterator iterator() | It is used to iterate the elements in ascending order. |
| E lower(E e) | It returns the closest least element of the specified element from the set, or null there is no such element. |
| E pollFirst() | It is used to retrieve and remove the lowest(first) element. |
| E pollLast() | It is used to retrieve and remove the highest(last) element. |
| Spliterator spliterator() | It is used to create a late-binding and fail-fast spliterator over the elements. |
| NavigableSet subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) | It returns a set of elements that lie between the given range. |
| SortedSet subSet(E fromElement, E toElement)) | It returns a set of elements that lie between the given range which includes fromElement and excludes toElement. |
| SortedSet tailSet(E fromElement) | It returns a set of elements that are greater than or equal to the specified element. |
| NavigableSet tailSet(E fromElement, boolean inclusive) | It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element. |
| boolean contains(Object o) | It returns true if this set contains the specified element. |
| boolean isEmpty() | It returns true if this set contains no elements. |
| boolean remove(Object o) | It is used to remove the specified element from this set if it is present. |
| void clear() | It is used to remove all of the elements from this set. |
| Object clone() | It returns a shallow copy of this TreeSet instance. |
| E first() | It returns the first (lowest) element currently in this sorted set. |
| E last() | It returns the last (highest) element currently in this sorted set. |
| int size() | It returns the number of elements in this set. |

Java TreeSet Examples

Java TreeSet Example 1:

Let's see a simple example of Java TreeSet.

1. **import** java.util.\*;
2. **class** TreeSet1{
3. **public** **static** **void** main(String args[]){
4. //Creating and adding elements
5. TreeSet<String> al=**new** TreeSet<String>();
6. al.add("Ravi");
7. al.add("Vijay");
8. al.add("Ravi");
9. al.add("Ajay");
10. //Traversing elements
11. Iterator<String> itr=al.iterator();
12. **while**(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. }
16. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection11" \t "_blank)**

Output:

Ajay

Ravi

Vijay

Java TreeSet Example 2:

Let's see an example of traversing elements in descending order.

1. **import** java.util.\*;
2. **class** TreeSet2{
3. **public** **static** **void** main(String args[]){
4. TreeSet<String> set=**new** TreeSet<String>();
5. set.add("Ravi");
6. set.add("Vijay");
7. set.add("Ajay");
8. System.out.println("Traversing element through Iterator in descending order");
9. Iterator i=set.descendingIterator();
10. **while**(i.hasNext())
11. {
12. System.out.println(i.next());
13. }
15. }
16. }

**[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestCollection11" \t "_blank)**

Output:

Traversing element through Iterator in descending order

Vijay

Ravi

Ajay

Traversing element through NavigableSet in descending order

Vijay

Ravi

Ajay

Java TreeSet Example 3:

Let's see an example to retrieve and remove the highest and lowest Value.

1. **import** java.util.\*;
2. **class** TreeSet3{
3. **public** **static** **void** main(String args[]){
4. TreeSet<Integer> set=**new** TreeSet<Integer>();
5. set.add(24);
6. set.add(66);
7. set.add(12);
8. set.add(15);
9. System.out.println("Highest Value: "+set.pollFirst());
10. System.out.println("Lowest Value: "+set.pollLast());
11. }
12. }

Output:

Highest Value: 12

Lowest Value: 66

Java TreeSet Example 4:

In this example, we perform various NavigableSet operations.

1. **import** java.util.\*;
2. **class** TreeSet4{
3. **public** **static** **void** main(String args[]){
4. TreeSet<String> set=**new** TreeSet<String>();
5. set.add("A");
6. set.add("B");
7. set.add("C");
8. set.add("D");
9. set.add("E");
10. System.out.println("Initial Set: "+set);
12. System.out.println("Reverse Set: "+set.descendingSet());
14. System.out.println("Head Set: "+set.headSet("C", **true**));
16. System.out.println("SubSet: "+set.subSet("A", **false**, "E", **true**));
18. System.out.println("TailSet: "+set.tailSet("C", **false**));
19. }
20. }

Output:

Initial Set: [A, B, C, D, E]

Reverse Set: [E, D, C, B, A]

Head Set: [A, B, C]

SubSet: [B, C, D, E]

TailSet: [D, E]

Java TreeSet Example 4:

In this example, we perform various SortedSetSet operations.

1. **import** java.util.\*;
2. **class** TreeSet4{
3. **public** **static** **void** main(String args[]){
4. TreeSet<String> set=**new** TreeSet<String>();
5. set.add("A");
6. set.add("B");
7. set.add("C");
8. set.add("D");
9. set.add("E");
11. System.out.println("Intial Set: "+set);
13. System.out.println("Head Set: "+set.headSet("C"));
15. System.out.println("SubSet: "+set.subSet("A", "E"));
17. System.out.println("TailSet: "+set.tailSet("C"));
18. }
19. }

Output:

Intial Set: [A, B, C, D, E]

Head Set: [A, B]

SubSet: [A, B, C, D]

TailSet: [C, D, E]

Java TreeSet Example: Book

Let's see a TreeSet example where we are adding books to set and printing all the books. The elements in TreeSet must be of a Comparable type. String and Wrapper classes are Comparable by default. To add user-defined objects in TreeSet, you need to implement the Comparable interface.

1. **import** java.util.\*;
2. **class** Book **implements** Comparable<Book>{
3. **int** id;
4. String name,author,publisher;
5. **int** quantity;
6. **public** Book(**int** id, String name, String author, String publisher, **int** quantity) {
7. **this**.id = id;
8. **this**.name = name;
9. **this**.author = author;
10. **this**.publisher = publisher;
11. **this**.quantity = quantity;
12. }
13. **public** **int** compareTo(Book b) {
14. **if**(id>b.id){
15. **return** 1;
16. }**else** **if**(id<b.id){
17. **return** -1;
18. }**else**{
19. **return** 0;
20. }
21. }
22. }
23. **public** **class** TreeSetExample {
24. **public** **static** **void** main(String[] args) {
25. Set<Book> set=**new** TreeSet<Book>();
26. //Creating Books
27. Book b1=**new** Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
28. Book b2=**new** Book(233,"Operating System","Galvin","Wiley",6);
29. Book b3=**new** Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);
30. //Adding Books to TreeSet
31. set.add(b1);
32. set.add(b2);
33. set.add(b3);
34. //Traversing TreeSet
35. **for**(Book b:set){
36. System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
37. }
38. }
39. }

Output:

101 Data Communications & Networking Forouzan Mc Graw Hill 4

121 Let us C Yashwant Kanetkar BPB 8

233 Operating System Galvin Wiley 6