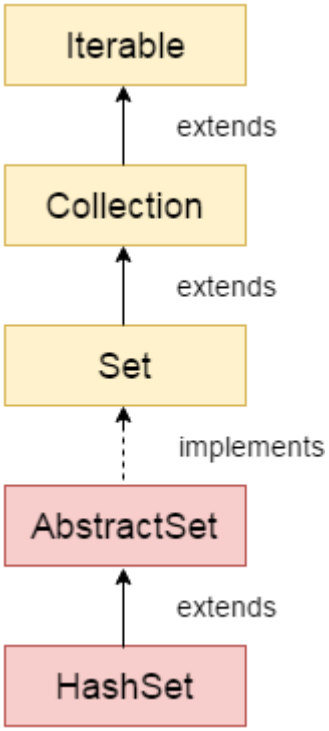
Java HashSet



Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

The important points about Java HashSet class are:

- HashSet stores the elements by using a mechanism called **hashing.**
- HashSet contains unique elements only.
- HashSet allows null value.
- HashSet class is non synchronized.
- HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
- HashSet is the best approach for search operations.
- The initial default capacity of HashSet is 16, and the load factor is 0.75.

Difference between List and Set

A list can contain duplicate elements whereas Set contains unique elements only.

Hierarchy of HashSet class

The HashSet class extends AbstractSet class which implements Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

HashSet class declaration

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

1. **public class** HashSet<E> **extends** AbstractSet<E> **implements** Set<E>, Clon eable, Serializable

Constructors of Java HashSet class

SN	Constructor	Description
1)	HashSet()	It is used to construct a default HashSet.
2)	HashSet(int capacity)	It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.
3)	HashSet(int capacity, float loadFactor)	It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.
4)	HashSet(Collection<? extends E> c)	It is used to initialize the hash set by using the elements of the collection c.

SN	Modifier & Type	Method	Description
1)	boolean	add(E e)	It is used to add the specified element to this set if it is not already present.
2)	void	clear()	It is used to remove all of the elements from the set.
3)	object	clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
4)	boolean	contains(Object o)	It is used to return true if this set contains the specified element.
5)	boolean	isEmpty()	It is used to return true if this set contains no elements.
6)	Iterator<E>	iterator()	It is used to return an iterator over the elements in this set.
7)	boolean	remove(Object o)	It is used to remove the specified element from this set if it is present.
8)	int	size()	It is used to return the number of elements in the set.

9)	Spliterator<E>	spliterator()	It is used to create a late-binding and fail-fast Spliterator over the elements in the set.

Methods of Java HashSet class

Various methods of Java HashSet class are as follows:

Java HashSet Example

Let's see a simple example of HashSet. Notice, the elements iterate in an unordered collection.

```
1. import java.util.*;
2. class HashSet1{
3.  public static void main(String args[]){
4.   //Creating HashSet and adding elements
5.     HashSet<String> set=new HashSet();
6.           set.add("One");
7.           set.add("Two");
8.           set.add("Three");
9.           set.add("Four");
10.          set.add("Five");
11.          Iterator<String> i=set.iterator();
12.          while(i.hasNext())
13.          {
14.          System.out.println(i.next());
15.          }
16. }
17.}
```

```
Five
One
Four
Two
Three
```

Java HashSet example ignoring duplicate elements

In this example, we see that HashSet doesn't allow duplicate elements.

```
1. import java.util.*;
2. class HashSet2{
3.  public static void main(String args[]){
4.   //Creating HashSet and adding elements
5.   HashSet<String> set=new HashSet<String>();
6.   set.add("Ravi");
```

```
7.    set.add("Vijay");
8.    set.add("Ravi");
9.    set.add("Ajay");
10.   //Traversing elements
11.   Iterator<String> itr=set.iterator();
12.   while(itr.hasNext()){
13.    System.out.println(itr.next());
14.   }
15.  }
16. }
```

```
Ajay
Vijay
Ravi
```

Java HashSet example to remove elements

Here, we see different ways to remove an element.

```
1.  import java.util.*;
2.  class HashSet3{
3.   public static void main(String args[]){
4.    HashSet<String> set=new HashSet<String>();
5.          set.add("Ravi");
6.          set.add("Vijay");
7.          set.add("Arun");
8.          set.add("Sumit");
9.          System.out.println("An initial list of elements: "+set);
10.         //Removing specific element from HashSet
11.         set.remove("Ravi");
12.         System.out.println("After invoking remove(object) method: "+set);
13.         HashSet<String> set1=new HashSet<String>();
14.         set1.add("Ajay");
15.         set1.add("Gaurav");
16.         set.addAll(set1);
17.         System.out.println("Updated List: "+set);
18.         //Removing all the new elements from HashSet
19.         set.removeAll(set1);
20.         System.out.println("After invoking removeAll() method: "+set);
21.         //Removing elements on the basis of specified condition
22.         set.removeIf(str->str.contains("Vijay"));
23.         System.out.println("After invoking removeIf() method: "+set);
24.         //Removing all the elements available in the set
25.         set.clear();
26.         System.out.println("After invoking clear() method: "+set);
```

```
27. }
28.}
```

```
An initial list of elements: [Vijay, Ravi, Arun, Sumit]
After invoking remove(object) method: [Vijay, Arun, Sumit]
Updated List: [Vijay, Arun, Gaurav, Sumit, Ajay]
After invoking removeAll() method: [Vijay, Arun, Sumit]
After invoking removeIf() method: [Arun, Sumit]
After invoking clear() method: []
```

Java HashSet from another Collection

```
1. import java.util.*;
2. class HashSet4{
3.  public static void main(String args[]){
4.    ArrayList<String> list=new ArrayList<String>();
5.          list.add("Ravi");
6.          list.add("Vijay");
7.          list.add("Ajay");
8.
9.          HashSet<String> set=new HashSet(list);
10.         set.add("Gaurav");
11.         Iterator<String> i=set.iterator();
12.         while(i.hasNext())
13.         {
14.         System.out.println(i.next());
15.         }
16. }
17.}
```

```
Vijay
Ravi
Gaurav
Ajay
```

Java HashSet Example: Book

Let's see a HashSet example where we are adding books to set and printing all the books.

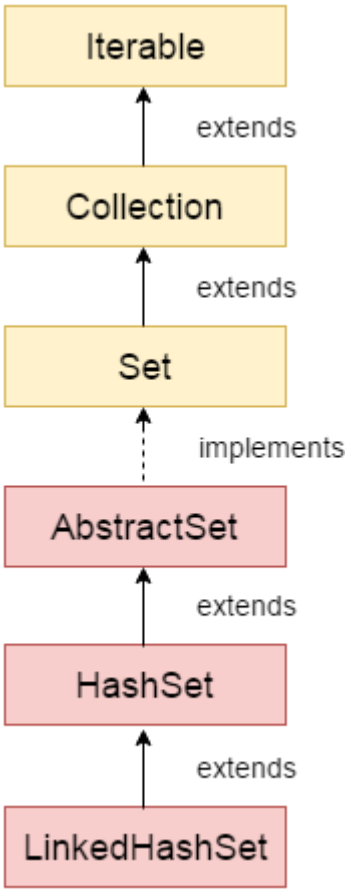
```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {

7.     this.id = id;
8.     this.name = name;
```

```
9.      this.author = author;
10.     this.publisher = publisher;
11.     this.quantity = quantity;
12. }
13. }
14. public class HashSetExample {
15. public static void main(String[] args) {
16.     HashSet<Book> set=new HashSet<Book>();
17.     //Creating Books
18.     Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
19.     Book b2=new Book(102,"Data Communications & Networking","Forouzan",
    "Mc Graw Hill",4);
20.     Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
21.     //Adding Books to HashSet
22.     set.add(b1);
23.     set.add(b2);
24.     set.add(b3);
25.     //Traversing HashSet
26.     for(Book b:set){
27.     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qu
    antity);
28.     }
29. }
30. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java LinkedHashSet Class



Java LinkedHashSet class is a Hashtable and Linked list implementation of the Set interface. It inherits the HashSet class and implements the Set interface.

The important points about the Java LinkedHashSet class are:

- Java LinkedHashSet class contains unique elements only like HashSet.
- Java LinkedHashSet class provides all optional set operations and permits null elements.
- Java LinkedHashSet class is non-synchronized.
- Java LinkedHashSet class maintains insertion order.

> Note: Keeping the insertion order in the LinkedHashset has some additional costs, both in terms of extra memory and extra CPU cycles. Therefore, if it is not required to maintain the insertion order, go for the lighter-weight HashMap or the HashSet instead.

Hierarchy of LinkedHashSet class

The LinkedHashSet class extends the HashSet class, which implements the Set interface. The Set interface inherits Collection and Iterable interfaces in hierarchical order.

LinkedHashSet Class Declaration

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

```
1. public class LinkedHashSet<E> extends HashSet<E> implements Set<E>, Cl
   oneable, Serializable
```

Constructors of Java LinkedHashSet Class

Constructor	Description
HashSet()	It is used to construct a default HashSet.
HashSet(Collection c)	It is used to initialize the hash set by using the elements of the collection c.
LinkedHashSet(int capacity)	It is used to initialize the capacity of the linked hash set to the given integer value capacity.
LinkedHashSet(int capacity, float fillRatio)	It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument.

Java LinkedHashSet Example

Let's see a simple example of the Java LinkedHashSet class. Here you can notice that the elements iterate in insertion order.

FileName: LinkedHashSet1.java

```
1. import java.util.*;
2. class LinkedHashSet1{
3.  public static void main(String args[]){
4.  //Creating HashSet and adding elements
5.        LinkedHashSet<String> set=new LinkedHashSet();
6.             set.add("One");
7.             set.add("Two");
8.             set.add("Three");
9.             set.add("Four");
10.            set.add("Five");
11.            Iterator<String> i=set.iterator();
12.            while(i.hasNext())
13.            {
14.            System.out.println(i.next());
15.            }
16. }
17.}
```

Output:

```
One
Two
```

```
Three
Four
Five
```

Note: We can also use the enhanced for loop for displaying the elements.

Java LinkedHashSet example ignoring duplicate Elements

sFileName: LinkedHashSet2.java

```
1. import java.util.*;
2. class LinkedHashSet2{
3.  public static void main(String args[]){
4.   LinkedHashSet<String> al=new LinkedHashSet<String>();
5.   al.add("Ravi");
6.   al.add("Vijay");
7.   al.add("Ravi");
8.   al.add("Ajay");
9.   Iterator<String> itr=al.iterator();
10.  while(itr.hasNext()){
11.   System.out.println(itr.next());
12.  }
13. }
14.}
```

Output:

```
Ravi
Vijay
Ajay
```

Remove Elements Using LinkeHashSet Class

FileName: LinkedHashSet3.java

```
1. import java.util.*;
2.
3. public class LinkedHashSet3
4. {
5.
6. // main method
7. public static void main(String argvs[])
8. {
9.
10. // Creating an empty LinekdhashSet of string type
11. LinkedHashSet<String> lhs = new LinkedHashSet<String>();
12.
```

```
13. // Adding elements to the above Set
14. // by invoking the add() method
15. lhs.add("Java");
16. lhs.add("T");
17. lhs.add("Point");
18. lhs.add("Good");
19. lhs.add("Website");
20.
21. // displaying all the elements on the console
22. System.out.println("The hash set is: " + lhs);
23.
24. // Removing an element from the above linked Set
25.
26. // since the element "Good" is present, therefore, the method remove()
27. // returns true
28. System.out.println(lhs.remove("Good"));
29.
30. // After removing the element
31. System.out.println("After removing the element, the hash set is: " + lhs);
32.
33. // since the element "For" is not present, therefore, the method remove()
34. // returns false
35. System.out.println(lhs.remove("For"));
36.
37. }
38. }
```

Output:

```
The hash set is: [Java, T, Point, Good, Website]
true
After removing the element, the hash set is: [Java, T, Point, Website]
false
```

Java LinkedHashSet Example: Book

FileName: Book.java

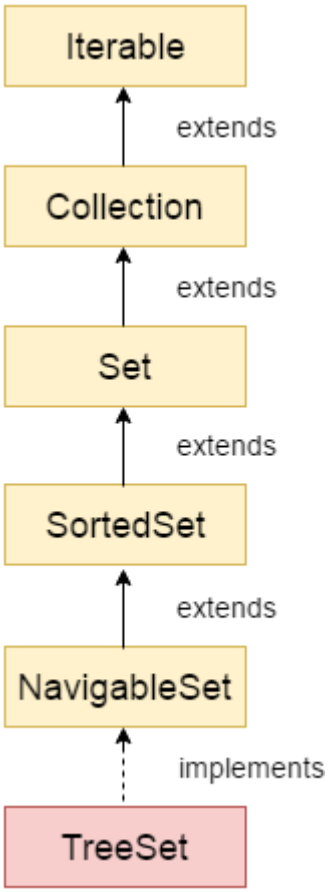
```
1. import java.util.*;
2. class Book {
3. int id;
4. String name,author,publisher;
5. int quantity;
6. public Book(int id, String name, String author, String publisher, int quantity) {
```

```
7.      this.id = id;
8.      this.name = name;
9.      this.author = author;
10.     this.publisher = publisher;
11.     this.quantity = quantity;
12. }
13. }
14. public class LinkedHashSetExample {
15. public static void main(String[] args) {
16.     LinkedHashSet<Book> hs=new LinkedHashSet<Book>();
17.     //Creating Books
18.     Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
19.     Book b2=new Book(102,"Data Communications & Networking","Forouzan",
    "Mc Graw Hill",4);
20.     Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
21.     //Adding Books to hash table
22.     hs.add(b1);
23.     hs.add(b2);
24.     hs.add(b3);
25.     //Traversing hash table
26.     for(Book b:hs){
27.     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qu
    antity);
28.     }
29. }
30. }
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

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 the 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.

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

- The TreeSet can only allow those generic types that are comparable. For example The Comparable interface is being implemented by the StringBuffer class.

Internal Working of The TreeSet Class

TreeSet is being implemented using a binary search tree, which is self-balancing just like a Red-Black Tree. Therefore, operations such as a search, remove, and add consume O(log(N)) time. The reason behind this is there in the self-balancing tree. It is there to ensure that the tree height never exceeds O(log(N)) for all of the mentioned operations. Therefore, it is one of the efficient data structures in order to keep the large data that is sorted and also to do operations on it.

Synchronization of The TreeSet Class

As already mentioned above, the TreeSet class is not synchronized. It means if more than one thread concurrently accesses a tree set, and one of the accessing threads modify it, then the synchronization must be done manually. It is usually done by doing some object synchronization that encapsulates the set. However, in the case where no such object is found, then the set must be wrapped with the help of the Collections.synchronizedSet() method. It is advised to use the method during creation time in order to avoid the unsynchronized access of the set. The following code snippet shows the same.

```
1. TreeSet treeSet = new TreeSet();
2. Set syncrSet = Collections.synchronziedSet(treeSet);
```

Hierarchy of TreeSet class

As shown in the above diagram, the 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 a comparator that arranges elements in order.
Iterator descendingIterator()	It is used to 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.

FileName: TreeSet1.java

```
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

Output:

```
Ajay
Ravi
Vijay
```

Java TreeSet Example 2:

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

FileName: TreeSet2.java

```
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 or
     der");
9.          Iterator i=set.descendingIterator();
10.         while(i.hasNext())
11.         {
12.             System.out.println(i.next());
13.         }
14.
15.  }
16. }
```

Test it Now

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.

FileName: TreeSet3.java

```
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("Lowest Value: "+set.pollFirst());
10.         System.out.println("Highest Value: "+set.pollLast());
11.  }
12. }
```

Output:

```
Lowest Value: 12
Highest Value: 66
```

Java TreeSet Example 4:

In this example, we perform various NavigableSet operations.

FileName: TreeSet4.java

```
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);
11.
12.        System.out.println("Reverse Set: "+set.descendingSet());
13.
14.        System.out.println("Head Set: "+set.headSet("C", true));
15.
16.        System.out.println("SubSet: "+set.subSet("A", false, "E", true));
17.
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 5:

In this example, we perform various SortedSetSet operations.

FileName: TreeSet5.java

```
1. import java.util.*;
2. class TreeSet5{
```

```
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.
11.       System.out.println("Intial Set: "+set);
12.
13.       System.out.println("Head Set: "+set.headSet("C"));
14.
15.       System.out.println("SubSet: "+set.subSet("A", "E"));
16.
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 the 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.

FileName: TreeSetExample.java

```
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. // implementing the abstract method
14. public int compareTo(Book b) {
15.     if(id>b.id){
16.         return 1;
17.     }else if(id<b.id){
18.         return -1;
19.     }else{
20.     return 0;
21.     }
22. }
23. }
24. public class TreeSetExample {
25. public static void main(String[] args) {
26.     Set<Book> set=new TreeSet<Book>();
27.     //Creating Books
28.     Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
29.     Book b2=new Book(233,"Operating System","Galvin","Wiley",6);
30.     Book b3=new Book(101,"Data Communications & Networking","Forouzan",
    "Mc Graw Hill",4);
31.     //Adding Books to TreeSet
32.     set.add(b1);
33.     set.add(b2);
34.     set.add(b3);
35.     //Traversing TreeSet
36.     for(Book b:set){
37.     System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qu
    antity);
38.     }
39. }
40. }
```

Output:

```
101 Data Communications & Networking Forouzan Mc Graw Hill 4
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6
```

ClassCast Exception in TreeSet

If we add an object of the class that is not implementing the Comparable interface, the ClassCast Exception is raised. Observe the following program.

FileName: ClassCastExceptionTreeSet.java

```
1.  // important import statement
2.  import java.util.*;
3.
4.  class Employee
5.  {
6.
7.  int empId;
8.  String name;
9.
10. // getting the name of the employee
11. String getName()
12. {
13.     return this.name;
14. }
15.
16. // setting the name of the employee
17. void setName(String name)
18. {
19. this.name = name;
20. }
21.
22. // setting the employee id
23. // of the employee
24. void setId(int a)
25. {
26. this.empId = a;
27. }
28.
29. // retrieving the employee id of
30. // the employee
31. int getId()
32. {
33. return this.empId;
34. }
35.
36. }
37.
38. public class ClassCastExceptionTreeSet
39. {
40.
41. // main method
42. public static void main(String[] argvs)
```

```
43. {
44. // creating objects of the class Employee
45. Employee obj1 = new Employee();
46.
47. Employee obj2 = new Employee();
48.
49. TreeSet<Employee> ts =  new TreeSet<Employee>();
50.
51. // adding the employee objects to
52. // the TreeSet class
53. ts.add(obj1);
54. ts.add(obj2);
55.
56. System.out.println("The program has been executed successfully.");
57.
58. }
59. }
```

When we compile the above program, we get the ClassCastException, as shown below.

```
Exception in thread "main" java.lang.ClassCastException: class Employee
cannot be cast to class java.lang.Comparable (Employee is in unnamed module
of loader 'app'; java.lang.Comparable is in module java.base of loader
'bootstrap')
        at java.base/java.util.TreeMap.compare(TreeMap.java:1569)
        at java.base/java.util.TreeMap.addEntryToEmptyMap(TreeMap.java:776)
        at java.base/java.util.TreeMap.put(TreeMap.java:785)
        at java.base/java.util.TreeMap.put(TreeMap.java:534)
        at java.base/java.util.TreeSet.add(TreeSet.java:255)
        at
ClassCastExceptionTreeSet.main(ClassCastExceptionTreeSet.java:52)
```

Explanation: In the above program, it is required to implement a Comparable interface. It is because the TreeSet maintains the sorting order, and for doing the sorting the comparison of different objects that are being inserted in the TreeSet is must, which is accomplished by implementing the Comparable interface.