Collections in java

Collections in java is a framework that provides an architecture to store and manipulate the group of objects.

All the operations that you perform on a data such as searching, sorting, insertion, manipulation, deletion etc. can be performed by Java Collections.

Java Collection simply means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque etc.) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet etc).

What is Collection in java

Collection represents a single unit of objects i.e. a group.

What is framework in java

- provides readymade architecture.
- represents set of classes and interface.
- is optional.

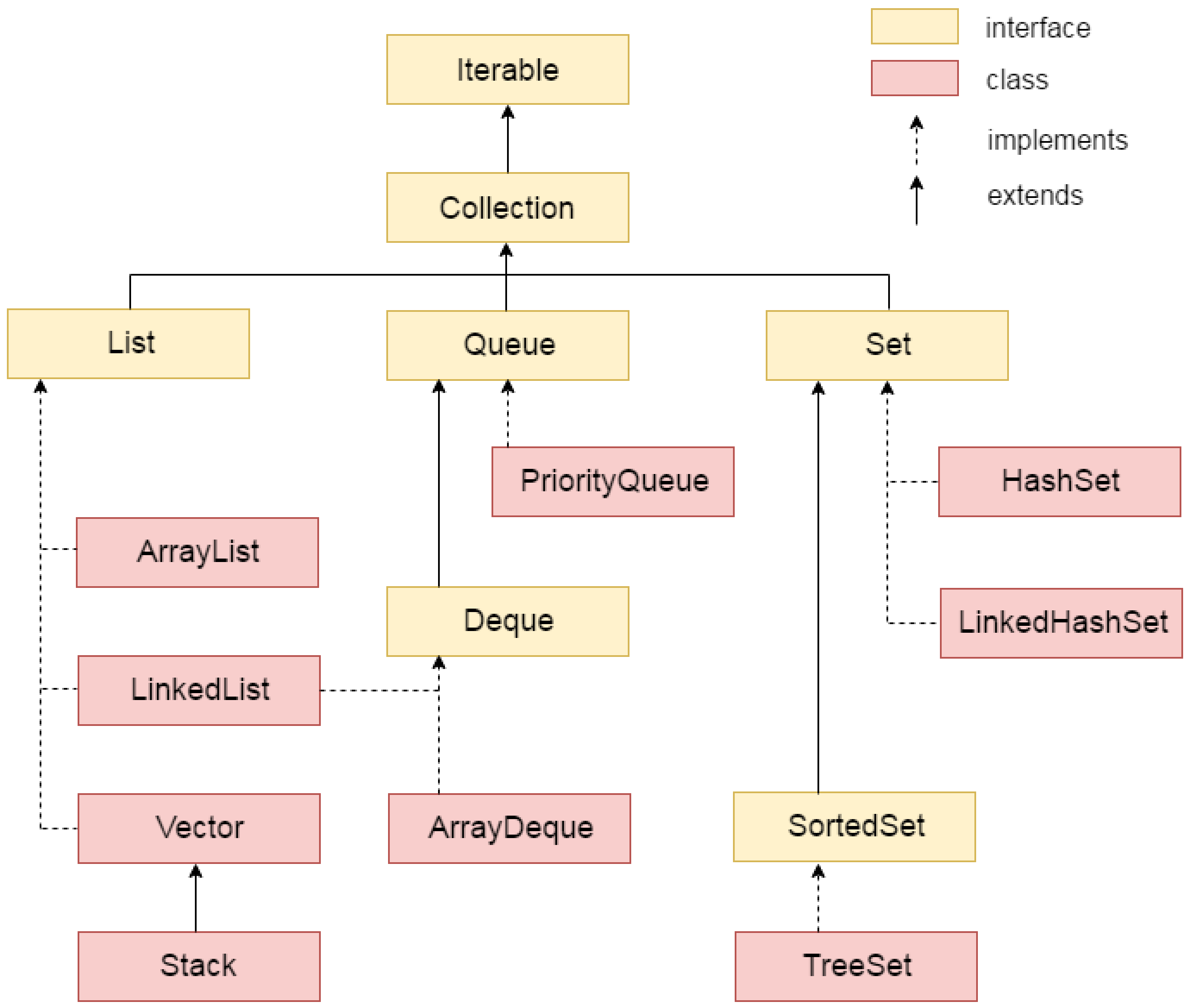
What is Collection framework

Collection framework represents a unified architecture for storing and manipulating group of objects. It has:

1. Interfaces and its implementations i.e. classes
2. Algorithm

Hierarchy of Collection Framework

Let us see the hierarchy of collection framework.The **java.util** package contains all the classes and interfaces for Collection framework.



Methods of Collection interface

There are many methods declared in the Collection interface. They are as follows:

No.	Method	Description
1	public boolean add(Object element)	is used to insert an element in this collection.

2	public boolean addAll(Collection c)	is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	is used to delete an element from this collection.
4	public boolean removeAll(Collection c)	is used to delete all the elements of specified collection from the invoking collection.
5	public boolean retainAll(Collection c)	is used to delete all the elements of invoking collection except the specified collection.
6	public int size()	return the total number of elements in the collection.
7	public void clear()	removes the total no of element from the collection.
8	public boolean contains(Object element)	is used to search an element.
9	public boolean containsAll(Collection c)	is used to search the specified collection in this collection.
10	public Iterator iterator()	returns an iterator.
11	public Object[] toArray()	converts collection into array.
12	public boolean isEmpty()	checks if collection is empty.
13	public boolean equals(Object element)	matches two collection.
14	public int hashCode()	returns the hashcode number for collection.

Iterator interface

Iterator interface provides the facility of iterating the elements in forward direction only.

Methods of Iterator interface

There are only three methods in the Iterator interface. They are:

1. **public boolean hasNext()** it returns true if iterator has more elements.
2. **public object next()** it returns the element and moves the cursor pointer to the next element.
3. **public void remove()** it removes the last elements returned by the iterator. It is rarely used.

java List Interface

List Interface is the subinterface of Collection.It contains methods to insert and delete elements in index basis.It is a factory of ListIterator interface.

List Interface declaration

1. public interface List<E> extends Collection<E>

Methods of Java List Interface

Method	Description
void add(int index,Object element)	It is used to insert element into the invoking list at the index passed in the index.
boolean addAll(int index,Collection c)	It is used to insert all elements of c into the invoking list at the index passed in the index.

object get(int index)	It is used to return the object stored at the specified index within the invoking collection.
object set(int index,Object element)	It is used to assign element to the location specified by index within the invoking list.
object remove(int index)	It is used to remove the element at position index from the invoking list and return the deleted element.
ListIterator listIterator()	It is used to return an iterator to the start of the invoking list.
ListIterator listIterator(int index)	It is used to return an iterator to the invoking list that begins at the specified index.

Java List Example

```
import java.util.*;
public class ListExample{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Amit");
al.add("Vijay");
al.add("Kumar");
al.add(1,"Sachin");
System.out.println("Element at 2nd position: "+al.get(2));
for(String s:al){
 System.out.println(s);
}
}
}
```

Output:

```
Element at 2nd position: Vijay
Amit
Sachin
Vijay
Kumar
```

Java ListIterator Interface

ListIterator Interface is used to traverse the element in backward and forward direction.

ListIterator Interface declaration

```
public interface ListIterator<E> extends Iterator<E>
```

Methods of Java ListIterator Interface:

Method	Description
boolean hasNext()	This method return true if the list iterator has more elements when traversing the list in the forward direction.
Object next()	This method return the next element in the list and advances the cursor position.
boolean hasPrevious()	This method return true if this list iterator has more elements when traversing the list in the reverse direction.
Object previous()	This method return the previous element in the list and moves the cursor position backwards.

Example of ListIterator Interface

```
import java.util.*;
public class TestCollection8{
public static void main(String args[]){
ArrayList<String> al=new ArrayList<String>();
al.add("Amit");
al.add("Vijay");
al.add("Kumar");
al.add(1,"Sachin");
System.out.println("element at 2nd position: "+al.get(2));
ListIterator<String> itr=al.listIterator();
System.out.println("traversing elements in forward direction...");
while(itr.hasNext()){
System.out.println(itr.next());
}
System.out.println("traversing elements in backward direction...");
while(itr.hasPrevious()){
System.out.println(itr.previous());
}
}
}
```

Test it Now

Output:

```
element at 2nd position: Vijay
traversing elements in forward direction...
Amit
Sachin
Vijay
Kumar
traversing elements in backward direction...
Kumar
Vijay
Sachin
Amit
```

Example of ListIterator Interface: Book

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class ListExample {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new ArrayList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
```

```
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Set

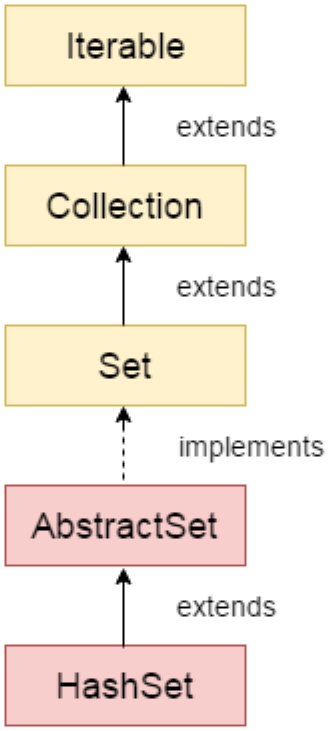
A Set is a Collection that cannot contain duplicate elements. The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

The methods declared by Set are summarized in the following table −

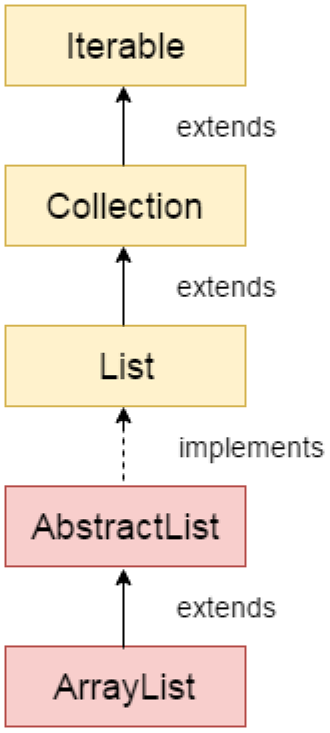
Sr.No.	Method & Description
1	**add( )** Adds an object to the collection.
2	**clear( )** Removes all objects from the collection.
3	**contains( )** Returns true if a specified object is an element within the collection.
4	**isEmpty( )** Returns true if the collection has no elements.
5	**iterator( )** Returns an Iterator object for the collection, which may be used to retrieve an object.
6	**remove( )** Removes a specified object from the collection.
7	**size( )** Returns the number of elements in the collection.

Difference between List and Set

List can contain duplicate elements whereas Set contains unique elements only.



Java ArrayList class



Java ArrayList class uses a dynamic array for storing the elements. It inherits AbstractList class and implements List interface.

The important points about Java ArrayList class are:

- Java ArrayList class can contain duplicate elements.
- Java ArrayList class maintains insertion order.
- Java ArrayList class is non synchronized.
- Java ArrayList allows random access because array works at the index basis.
- In Java ArrayList class, manipulation is slow because a lot of shifting needs to be occurred if any element is removed from the array list.

Hierarchy of ArrayList class

As shown in above diagram, Java ArrayList class extends AbstractList class which implements List interface. The List interface extends Collection and Iterable interfaces in hierarchical order.

ArrayList class declaration

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

1. public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Cloneable , Serializable

Constructors of Java ArrayList

Constructor	Description
ArrayList()	It is used to build an empty array list.
ArrayList(Collection c)	It is used to build an array list that is initialized with the elements of the collection c.
ArrayList(int capacity)	It is used to build an array list that has the specified initial capacity.

Methods of Java ArrayList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.

boolean addAll(Collection c)	It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
void clear()	It is used to remove all of the elements from this list.
int lastIndexOf(Object o)	It is used to return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object[] toArray()	It is used to return an array containing all of the elements in this list in the correct order.
Object[] toArray(Object[] a)	It is used to return an array containing all of the elements in this list in the correct order.
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean addAll(int index, Collection c)	It is used to insert all of the elements in the specified collection into this list, starting at the specified position.
Object clone()	It is used to return a shallow copy of an ArrayList.
int indexOf(Object o)	It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
void trimToSize()	It is used to trim the capacity of this ArrayList instance to be the list's current size.

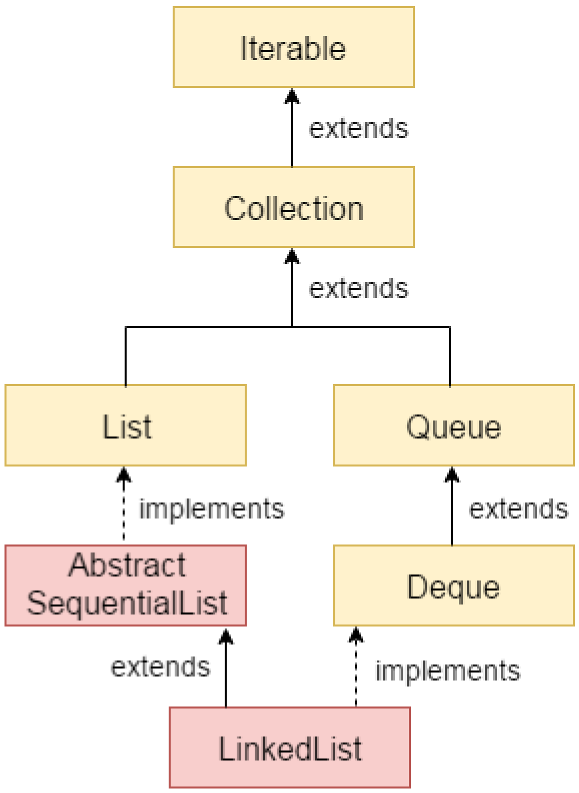
Java ArrayList Example

```
import java.util.*;
class TestCollection1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
  list.add("Ravi");//Adding object in arraylist
  list.add("Vijay");
  list.add("Ravi");
  list.add("Ajay");
  //Traversing list through Iterator
  Iterator itr=list.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

output

```
Ravi
Vijay
Ravi
Ajay
```

Java LinkedList class



Java LinkedList class uses doubly linked list to store the elements. It provides a linked-list data structure. It inherits the AbstractList class and implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.

- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to be occurred.
- Java LinkedList class can be used as list, stack or queue.

Hierarchy of LinkedList class

As shown in above diagram, Java LinkedList class extends AbstractSequentialList class and implements List and Deque interfaces.

Doubly Linked List

In case of doubly linked list, we can add or remove elements from both side.

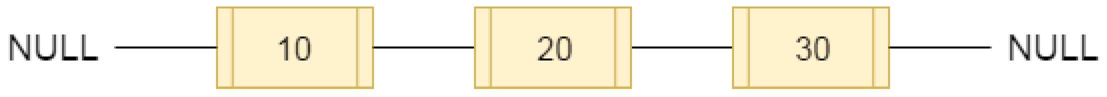


fig- doubly linked list

LinkedList class declaration

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

1. public class LinkedList<E> extends AbstractSequentialList<E> implements List<E>, Deque<E>, Cloneable, Serializable

Constructors of Java LinkedList

Constructor	Description
LinkedList()	It is used to construct an empty list.
LinkedList(Collection c)	It is used to construct a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Methods of Java LinkedList

Method	Description
void add(int index, Object element)	It is used to insert the specified element at the specified position index in a list.
void addFirst(Object o)	It is used to insert the given element at the beginning of a list.

void addLast(Object o)	It is used to append the given element to the end of a list.
int size()	It is used to return the number of elements in a list
boolean add(Object o)	It is used to append the specified element to the end of a list.
boolean contains(Object o)	It is used to return true if the list contains a specified element.
boolean remove(Object o)	It is used to remove the first occurence of the specified element in a list.
Object getFirst()	It is used to return the first element in a list.
Object getLast()	It is used to return the last element in a list.
int indexOf(Object o)	It is used to return the index in a list of the first occurrence of the specified element, or -1 if the list does not contain any element.
int lastIndexOf(Object o)	It is used to return the index in a list of the last occurrence of the specified element, or -1 if the list does not contain any element.

Java LinkedList Example

```java
import java.util.*;
public class TestCollection7{
 public static void main(String args[]){

  LinkedList<String> al=new LinkedList<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");

  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Test it Now

```
Output:Ravi
       Vijay
       Ravi
       Ajay
```

Java LinkedList Example: Book

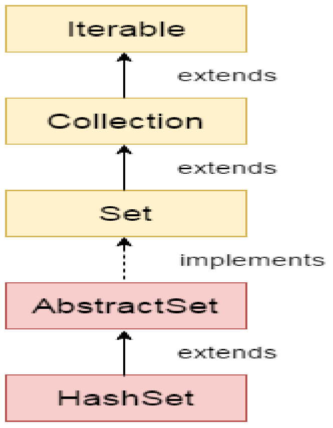
```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class LinkedListExample {
public static void main(String[] args) {
    //Creating list of Books
    List<Book> list=new LinkedList<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to list
    list.add(b1);
    list.add(b2);
    list.add(b3);
    //Traversing list
    for(Book b:list){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

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 HashSet class



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.

Difference between List and Set

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>, Cloneable, Serializable

Constructors of Java HashSet 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.
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.

Methods of Java HashSet class:

Method	Description
void clear()	It is used to remove all of the elements from this set.
boolean contains(Object o)	It is used to return true if this set contains the specified element.
boolean add(Object o)	It is used to adds the specified element to this set if it is not already present.
boolean isEmpty()	It is used to return 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.
Object clone()	It is used to return a shallow copy of this HashSet instance: the elements themselves are not cloned.
Iterator iterator()	It is used to return an iterator over the elements in this set.
int size()	It is used to return the number of elements in this set.

Java HashSet Example

```
import java.util.*;
class TestCollection9{
 public static void main(String args[]){
  //Creating HashSet and adding elements
  HashSet<String> set=new HashSet<String>();
  set.add("Ravi");
  set.add("Vijay");
  set.add("Ravi");
  set.add("Ajay");
  //Traversing elements
  Iterator<String> itr=set.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

Test it Now

```
Ajay
Vijay
Ravi
```

Java HashSet Example: Book

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

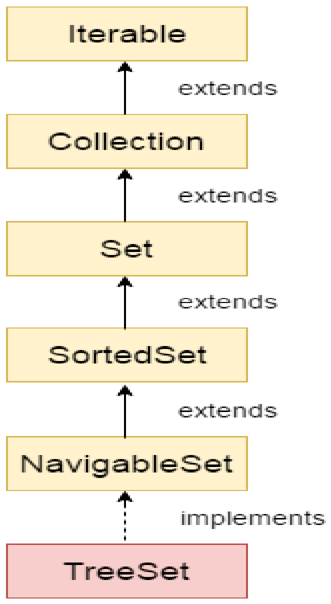
```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class HashSetExample {
public static void main(String[] args) {
    HashSet<Book> set=new HashSet<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to HashSet
    set.add(b1);
    set.add(b2);
    set.add(b3);
    //Traversing HashSet
    for(Book b:set){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

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.

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, Serializ able

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

```
import java.util.*;
class TreeSet1{
 public static void main(String args[]){
  //Creating and adding elements
  TreeSet<String> al=new TreeSet<String>();
  al.add("Ravi");
  al.add("Vijay");
  al.add("Ravi");
  al.add("Ajay");
  //Traversing elements
  Iterator<String> itr=al.iterator();
  while(itr.hasNext()){
   System.out.println(itr.next());
  }
 }
}
```

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

```
import java.util.*;
class TreeSet2{
 public static void main(String args[]){
 TreeSet<String> set=new TreeSet<String>();
         set.add("Ravi");
         set.add("Vijay");
         set.add("Ajay");
         System.out.println("Traversing element through Iterator in descending order");
         Iterator i=set.descendingIterator();
         while(i.hasNext())
         {
            System.out.println(i.next());
         }

 }
}
```

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

```
import java.util.*;
class TreeSet3{
 public static void main(String args[]){
 TreeSet<Integer> set=new TreeSet<Integer>();
        set.add(24);
        set.add(66);
        set.add(12);
        set.add(15);
        System.out.println("Lowest Value: "+set.pollFirst());
        System.out.println("Highest Value: "+set.pollLast());
 }
}
```

Output:

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

Java TreeSet Example 4:

In this example, we perform various NavigableSet operations.

FileName: TreeSet4.java

```
import java.util.*;
class TreeSet4{
 public static void main(String args[]){
  TreeSet<String> set=new TreeSet<String>();
        set.add("A");
        set.add("B");
        set.add("C");
        set.add("D");
        set.add("E");
        System.out.println("Initial Set: "+set);

        System.out.println("Reverse Set: "+set.descendingSet());
```

```
       
       System.out.println("Head Set: "+set.headSet("C", true));
       
       System.out.println("SubSet: "+set.subSet("A", false, "E", true));
       
       System.out.println("TailSet: "+set.tailSet("C", false));
 }
}
```

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

```
import java.util.*;
class TreeSet5{
 public static void main(String args[]){
  TreeSet<String> set=new TreeSet<String>();
       set.add("A");
       set.add("B");
       set.add("C");
       set.add("D");
       set.add("E");
       
       System.out.println("Intial Set: "+set);
       
       System.out.println("Head Set: "+set.headSet("C"));
       
       System.out.println("SubSet: "+set.subSet("A", "E"));
       
       System.out.println("TailSet: "+set.tailSet("C"));
 }
```

```java
}
```

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

```java
import java.util.*;
class Book implements Comparable<Book>{
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
   this.id = id;
   this.name = name;
   this.author = author;
   this.publisher = publisher;
   this.quantity = quantity;
}
// implementing the abstract method
public int compareTo(Book b) {
   if(id>b.id){
      return 1;
   }else if(id<b.id){
      return -1;
   }else{
   return 0;
   }
}
}
public class TreeSetExample {
public static void main(String[] args) {
   Set<Book> set=new TreeSet<Book>();
   //Creating Books
   Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
   Book b2=new Book(233,"Operating System","Galvin","Wiley",6);
   Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

   //Adding Books to TreeSet
   set.add(b1);
   set.add(b2);
   set.add(b3);
   //Traversing TreeSet
   for(Book b:set){
   System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
   }
}
}
```

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

```
// important import statement
import java.util.*;

class Employee
{

int empId;
```

```java
String name;

// getting the name of the employee
String getName()
{
   return this.name;
}

// setting the name of the employee
void setName(String name)
{
this.name = name;
}

// setting the employee id
// of the employee
void setId(int a)
{
this.empId = a;
}

// retrieving the employee id of
// the employee
int getId()
{
return this.empId;
}

}

public class ClassCastExceptionTreeSet
{

// main method
public static void main(String[] argvs)
{
// creating objects of the class Employee
Employee obj1 = new Employee();

Employee obj2 = new Employee();

TreeSet<Employee> ts =  new TreeSet<Employee>();

// adding the employee objects to
// the TreeSet class
ts.add(obj1);
ts.add(obj2);

System.out.println("The program has been executed successfully.");

}
}
```