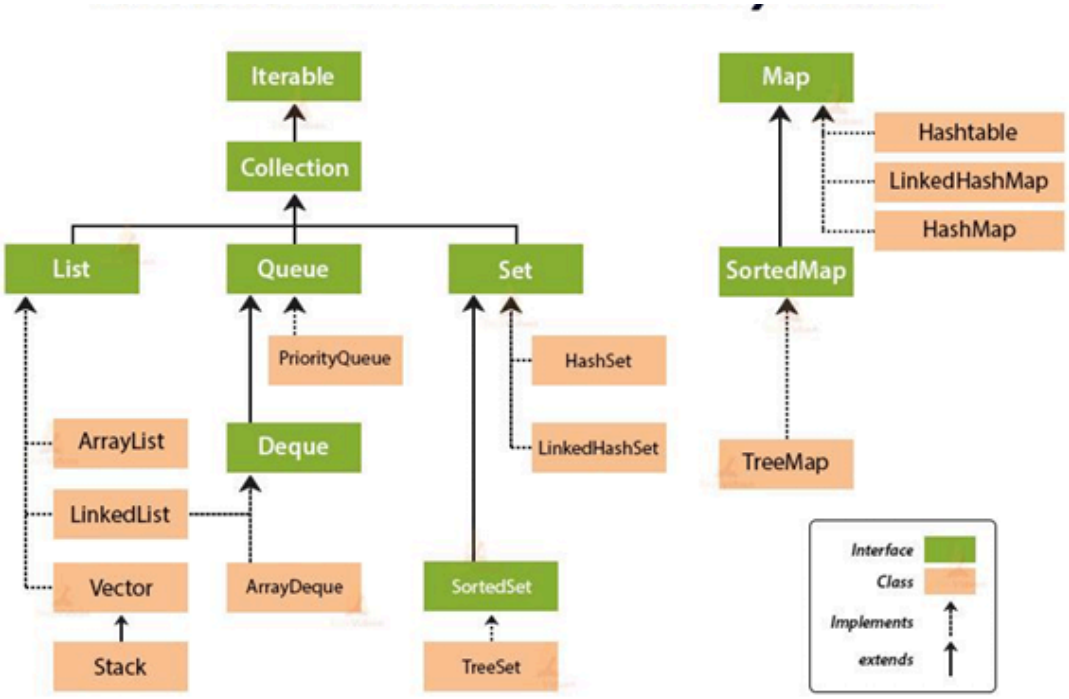
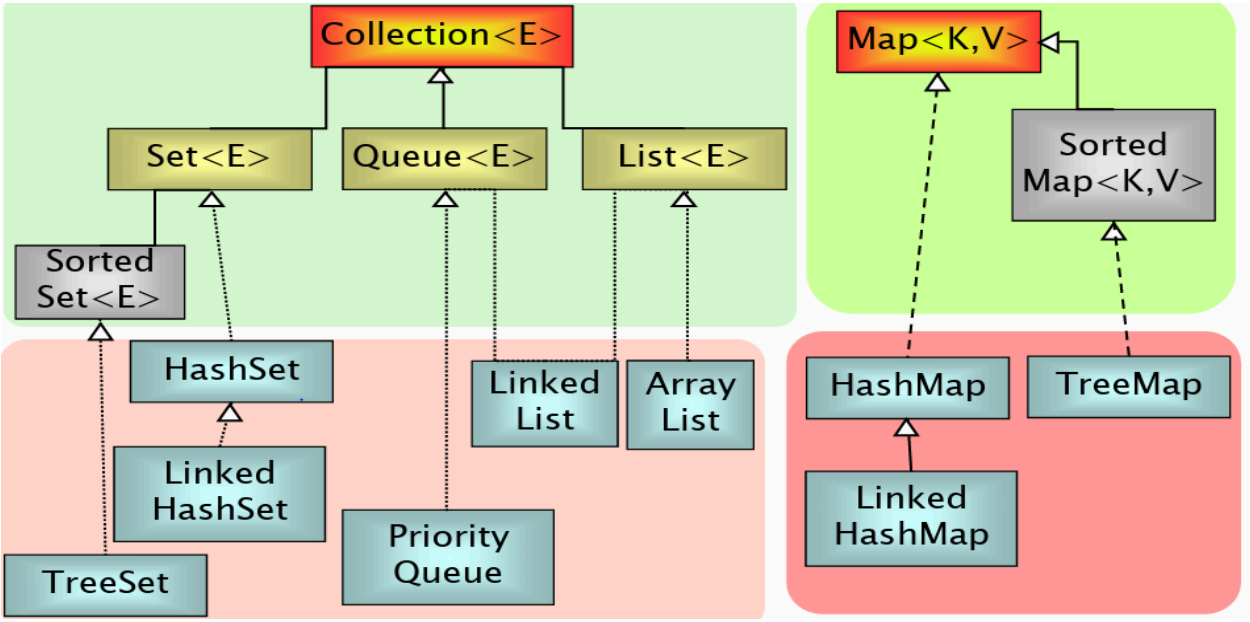
Collection Framework

- The Collection in Java is a framework that provides a facility to store and manipulate the group of objects.
- A Collection is a group of individual objects represented as a single unit.
- Collection Framework is a set of classes and interfaces that implement commonly reusable collection data structures.
- It works in the manner of a library.
- The 'java.util' package contains all the classes and interfaces for the Collection framework.
- It provided methods to perform all type of operations on data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection framework provides many interfaces such as Set, List, etc. and classes such as ArrayList, etc.

Collection Framework Hierarchy



Another perspective of hierarchy of collection framework in terms of generics is



Collection Interface

- The Collection interface is the foundation upon which the collections framework is built.
- It declares the core methods that all collections will have.
- Several of the methods can throw an UnsupportedOperationException.

Methods of Collection Interface

1. boolean add(Object obj): Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if obj is already a member of the collection, or if the collection does not allow duplicates.

2. boolean addAll(Collection c): Adds all the elements of c to the

invoking collection. Returns true if the operation succeeds (i.e., the elements were added). Otherwise, returns false.

3. void clear( ): Removes all elements from the invoking collection.

4. boolean contains(Object obj): Returns true if obj is an element of the invoking collection. Otherwise, returns false.

5. boolean containsAll(Collection c): Returns true if the invoking collection contains all elements of c. Otherwise, returns false.

6. boolean equals(Object obj): Returns true if the invoking collection and obj are equal. Otherwise, returns false.

7. int hashCode( ): Returns the hash code for the invoking collection.

8. boolean isEmpty( ): Returns true if the invoking collection is empty. Otherwise, returns false.

9. Iterator iterator( ): Returns an iterator for the invoking collection.

10. boolean remove(Object obj): Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.

11. boolean removeAll(Collection c): Removes all elements of c from the invoking collection. Returns true if the collection changed (i.e., elements were removed). Otherwise, returns false.

12. int size( ): Returns the number of elements held in the invoking collection.

13. Object[ ] toArray( ): Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.

14. Object[ ] toArray(Object array[ ]): Returns an array containing only those collection elements whose type matches that of array.

Example

```
import java.util.*;
public class CollectionDemo {
   public static void main(String args[]){
      Collection<Integer> li = new ArrayList<Integer>();
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter number of elements to store in list:");
      int n=sc.nextInt();
      System.out.println("Enter elements in list:");
     for(int i=0;i<n;i++){
         li.add(sc.nextInt());
     }
     System.out.println("size of list is:"+li.size());
     System.out.println("Elements in the list are:");
     Iterator<Integer> itr = li.iterator();
    while(itr.hasNext()){
         System.out.print(itr.next()+" ");
    }
     System.out.println();
     System.out.println("Enter key to search:");
     int key=sc.nextInt();
     if(li.contains(key)){
          System.out.println("Key found");
     }
     else{
          System.out.println("Key not found");
     }
     ArrayList<Integer> l=new ArrayList<Integer>();
     l.add(100);
```

```
        l.add(200);
        l.add(300);
       li.addAll(l);
       System.out.println("List after adding new collection:"+li);
       if(li.containsAll(l)){
            System.out.println("All elements in sublist are present");
       }
       else{
            System.out.println("Some or no elements present in the list");
       }
       li.remove(100);
       System.out.println("list after removing is:"+li);
       Object a[]=li.toArray();
       System.out.println("list elements after converting to array are:");
       for(int i=0;i<a.length;i++){
          System.out.print(a[i]+" ");
       }
    }
}
```

Iterator Interface

- An interface that allows a client to traverse the elements of a collection, regardless of its implementation.
- It remembers a position within a collection, and allows you to:
  - get the element at that position
  - advance to the next position

- (possibly) remove or change the element at that position

Methods of Iterator Interface

- hasNext() returns true if there are more elements to examine
- next() returns the next element from the collection (throws a NoSuchElementException if there are none left to examine)
- remove() removes from the collection the last value returned by next()(throws IllegalStateException if you have not called next() Yet)

Example:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class IteratorDemo {
 public static void main(String args[]){
String removeElem = "Perl";
 List<String> myList = new ArrayList<String>();
  myList.add("Java");
 myList.add("Unix");
  myList.add("Oracle");
  myList.add("C++");
  myList.add("Perl");
```

```
System.out.println("Before remove:");
 System.out.println(myList);
  Iterator<String> itr = myList.iterator();
  while(itr.hasNext()){
  if(removeElem.equals(itr.next())){
      itr.remove();
    }
   }
  System.out.println("After remove:");
  System.out.println(myList);
  }
}
```

ListIterator Interface

- ListIterator extends Iterator to allow bi-directional traversal of a list, and the modification of elements.
- ListIterator is a generic interface which is declared as
- interface ListIterator<E>
- E specifies the datatype of elements to be iterated.

Methods of ListIterator Interface

- void add(Object obj)Inserts obj into the list in front of the element that will be returned by the next call to next( ).
- boolean hasNext( )Returns true if there is a next element.

Otherwise, returns false.

- boolean hasPrevious( )Returns true if there is a previous element. Otherwise, returns false.

- Object next( )Returns the next element. A NoSuchElementException is thrown if there is not a next element.

- int nextIndex( )Returns the index of the next element. If there is not a next element, returns the size of the list.

- Object previous( )Returns the previous element. A NoSuchElementException is thrown if there is not a previous element.

- int previousIndex( )Returns the index of the previous element. If there is not a previous element, returns -1.

- void remove( )Removes the current element from the list. An IllegalStateException is thrown if remove( ) is called before next( ) or previous( ) is invoked.

- void set(Object obj)Assigns obj to the current element. This is the element last returned by a call to either next( ) or previous( ).

Example:

```
import java.util.*;
public class ListIteratorDemo {
```

```
    public static void main(String args[]){
        List<Integer> li = new ArrayList<Integer>();
        ListIterator<Integer> litr = null;
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter number of elements to store in
list:");
        int n=sc.nextInt();
        System.out.println("Enter elements in list:");
       for(int i=0;i<n;i++){
            li.add(sc.nextInt());
       }
        litr=li.listIterator();
        System.out.println("Elements in forward directiton");
        while(litr.hasNext()){
            System.out.println(litr.next());
        }
        litr.remove();
        System.out.println("Elements in backward directiton");
        while(litr.hasPrevious()){
            System.out.println(litr.previous());
        }
        litr.set(0);
        System.out.println("Elements in forward directiton");
        while(litr.hasNext()){
            System.out.println(litr.next());
        }
    }

}
```

List Interface

- The list interface extends collection interface.
- It declares the behavior of the collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- List is a generic interface with declaration as
- interface List<E>
- E specifies the datatype of elements of the list.

Methods of List Interface

- void add(int index, Object obj)Inserts obj into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
- boolean addAll(int index, Collection c)Inserts all elements of c into the invoking list at the index passed in index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
- Object get(int index)Returns the object stored at the specified

index within the invoking collection.

- int indexOf(Object obj)Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, -1 is returned.

- int lastIndexOf(Object obj)Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, -1 is returned.

- ListIterator listIterator( )Returns an iterator to the start of the invoking list.

- ListIterator listIterator(int index)Returns an iterator to the invoking list that begins at the specified index.

- Object remove(int index)Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one

- Object set(int index, Object obj)Assigns obj to the location specified by index within theinvoking list.

- List subList(int start, int end)Returns a list that includes elements from start to end-1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

- List interface has been implemented in various classes like ArrayList or LinkedList, etc.

ArrayList Class

- The ArrayList class extends AbstractList and implements the List interface.
- ArrayList supports dynamic arrays that can grow as needed.
- Standard Java arrays are of a fixed length.
- After arrays are created, they cannot grow or shrink, which means that you must know in advance how many elements an array will hold.
- Array lists are created with an initial size.
- When this size is exceeded, the collection is automatically enlarged.
- When objects are removed, the array may be shrunk.

Methods of ArrayList class

- void add(int index, Object element):Inserts the specified element at the specified position index in this list. Throws

IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).

- boolean add(Object o): Appends the specified element to the end of this list.

- boolean addAll(Collection c): Appends 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. Throws NullPointerException, if the specified collection is null.

- boolean addAll(int index, Collection c): Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.

- void clear(): Removes all of the elements from this list.

- Object clone(): Returns a shallow copy of this ArrayList.

- boolean contains(Object o): Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).

- void ensureCapacity(int minCapacity):Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.

- Object get(int index): Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).

- protected void removeRange(int fromIndex, int toIndex): Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.

- int indexOf(Object o): Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

- int lastIndexOf(Object o): Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

- Object remove(int index): Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if the index out is of range (index < 0 || index >= size()).

- Object set(int index, Object element): Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).

- int size(): Returns the number of elements in this list.

- Object[] toArray(): Returns an array containing all of the

elements in this list in the correctorder. Throws NullPointerException if the specified array is null.

- Object[] toArray(Object[] a): Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

- void trimToSize():Trims the capacity of this ArrayList instance to be the list's current size.

Example:

```
import java.util.*;
public class ArrayListDemo {
   public static void main(String args[]){
      List<Integer> li = new ArrayList<Integer>();
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter number of elements to store in list:");
      int n=sc.nextInt();
      System.out.println("Enter elements in list:");
     for(int i=0;i<n;i++){
         li.add(i,sc.nextInt());
     }
      System.out.println("size of list is:"+li.size());
      System.out.println("Elements in the list in forward direction
are:");
      ListIterator<Integer> itr = li.listIterator();
    while(itr.hasNext()){
         System.out.print(itr.next()+" ");
    }
      System.out.println();
      System.out.println("Elements in the list in backward direction
```

```
are:");
        while(itr.hasPrevious()){
            System.out.print(itr.previous()+" ");
       }
        System.out.println("Enter key to search:");
        int key=sc.nextInt();
        System.out.println("Index of key is:"+li.indexOf(key));
        System.out.println("last index of key is:"+li.lastIndexOf(key));
        ArrayList<Integer> l=new ArrayList<Integer>();
        l.add(100);
        l.add(200);
        l.add(300);
       li.addAll(l);
        System.out.println("List after adding new collection:"+li);
     li.set(0,0);
     System.out.println("list after setting new element is:"+li);
     List<Integer> l2=li.subList(3,7);
     System.out.println("Elements in the sublist are:");
    for(int i=0;i<l2.size();i++){
        System.out.print(l2.get(i)+" ");
    }
    }
}
```

LinkedList Class

- The LinkedList class extends AbstractSequentialList and implements the List interface.
- It provides a linked-list data structure.

Methods of LinkedList Class

- void add(int index, Object element): Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index > size()).

- boolean add(Object o): Appends the specified element to the end of this list.

- boolean addAll(Collection c): Appends 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. Throws NullPointerException if the specified collection is null.

- boolean addAll(int index, Collection c): Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.

- void addFirst(Object o): Inserts the given element at the beginning of this list.

- void addLast(Object o): Appends the given element to the end of this list.

- void clear(): Removes all of the elements from this list.

- Object clone(): Returns a shallow copy of this LinkedList.

- boolean contains(Object o): Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).

- Object get(int index):Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).

- Object getFirst():Returns the first element in this list. Throws NoSuchElementException if this list is empty.

- Object getLast(): Returns the last element in this list. Throws NoSuchElementException if this list is null.

- int indexOf(Object o): Returns the index in this list of the first occurrence of the specified element, or -1 if the list does not contain this element.

- int lastIndexOf(Object o): Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.

- ListIterator listIterator(int index): Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list. Throws IndexOutOfBoundsException if the

specified index is out of range (index < 0 || index >= size()).

- Object remove(int index): Removes the element at the specified position in this list. Throws NoSuchElementException if this list is empty.

- boolean remove(Object o):Removes the first occurrence of the specified element in this list. Throws NoSuchElementException if this list is empty. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).

- Object removeFirst(): Removes and returns the first element from this list. Throws NoSuchElementException if this list is empty.

- Object removeLast(): Removes and returns the last element from this list. Throws NoSuchElementException if this list is empty.

- Object set(int index, Object element): Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is out of range (index < 0 || index >= size()).

- int size():Returns the number of elements in this list.

- Object[] toArray(): Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.

- Object[] toArray(Object[] a): Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.

Example:

```
import java.util.*;
public class LinkedListDemo {
  public static void main(String args[]) {
    LinkedList ll = new LinkedList();
    ll.add("F");
    ll.add("B");
    ll.add("D");
    ll.add("E");
    ll.add("C");
    ll.addLast("Z");
    ll.addFirst("A");
    ll.add(1, "A2");
    System.out.println("Original contents of ll: " + ll);
    ll.remove("F");
    ll.remove(2);
    System.out.println("Contents of ll after deletion: " + ll);
```

```
        ll.removeFirst();
        ll.removeLast();
        System.out.println("ll after deleting first and last: " + ll);
        Object val = ll.get(2);
        ll.set(2, (String) val + " Changed");
        System.out.println("ll after change: " + ll);
    }
}
```

Set Interface

- The set interface defines a set with unique elements.
- It extends the collection and declares the behaviour of a collection that does not allow duplicate elements.
- Set is a generic interface with declaration as
- Interface Set<E>
- E specifies the datatype of elements in set.
- Set is implemented by HashSet, TreeSet, LinkedHashSet classes.

Methods of Set Interface

add(element)	Adds a new value to the set
addAll(collection)	Adds all the elements of a collection to the specif
clear()	Clears all the values of a set (does not clear the se
contains(element)	Checks if the specified element is present in a set
containsAll(collection)	Checks if all the values of a collection are presen
hashCode()	Returns an integer hash code for the instance of a
isEmpty()	Checks if the set is empty
iterator()	Returns an iterator object to parse through the set
remove(element)	Removes a specified value from the set
removeAll(collection)	Removes the elements of a collection from the se
retainAll(collection)	Retains the values of a collection in a set
size()	Returns the size of the set
toArray()	Creates an array with the elements of a set

HashSet Class

- HashSet extends AbstractSet and implements the Set interface.
- It creates a collection that uses a hash table for storage.
- A hash table stores information by using a mechanism called hashing.
- In hashing, the informational content of a key is used to determine a unique value, called its hash code.
- The hash code is then used as the index at which the data associated with the key is stored.
- The transformation of the key into its hash code is performed automatically.

Methods of HashSet Class

- boolean add(Object o): Adds the specified element to this set if it is not already present.
- void clear(): Removes all of the elements from this set.
- Object clone(): Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
- boolean contains(Object o): Returns true if this set contains the specified element.

- boolean isEmpty(): Returns true if this set contains no elements.
- Iterator iterator(): Returns an iterator over the elements in this set.
- boolean remove(Object o): Removes the specified element from this set if it is present.
- int size(): Returns the number of elements in this set (its cardinality).

Example:

```
import java.util.*;
public class HashSetDemo {
   public static void main(String args[]){
      Set<Integer> li = new HashSet<Integer>();
      Scanner sc=new Scanner(System.in);
      System.out.println("Enter number of elements to store in set:");
      int n=sc.nextInt();
      System.out.println("Enter elements in set:");
     for(int i=0;i<n;i++){
         li.add(sc.nextInt());
     }
     System.out.println("size of set is:"+li.size());
     System.out.println("Elements in the set are:");
     Iterator<Integer> itr = li.iterator();
    while(itr.hasNext()){
         System.out.print(itr.next()+" ");
    }
     System.out.println();
```

```
 System.out.println("Enter key to search:");
 int key=sc.nextInt();
 if(li.contains(key)){
       System.out.println("Key found");
 }
 else{
      System.out.println("Key not found");
 }
 HashSet<Integer> l=new HashSet<Integer>();
 l.add(10);
 l.add(2);
 l.add(3);
li.addAll(l);
System.out.println("Set after adding new collection:"+li);
if(li.containsAll(l)){
    System.out.println("All elements in subset are present");
}
else{
    System.out.println("Some or no elements present in the set");
}
li.remove(100);
System.out.println("set after removing is:"+li);
li.retainAll(l);
System.out.println("set after retaining is:"+li);
l.removeAll(li);
System.out.println("set after removing is:"+l);
Object a[]=li.toArray();
System.out.println("set elements after converting to array are:");
for(int i=0;i<a.length;i++){
   System.out.print(a[i]+" ");
}
```

```
    }
}
```

LinkedHashSet Class

- This class extends HashSet, but adds no members of its own.
- LinkedHashSet maintains a linked list of the entries in the set, in the order in which they were inserted.
- This allows insertion-order iteration over the set.
- That is, when cycling through a LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.
- The hash code is then used as the index at which the data associated with the key is stored.
- The transformation of the key into its hash code is performed automatically.

Example:

```
import java.util.*;
public class HashSetDemo {
```

```
    public static void main(String args[]) {
      LinkedHashSet hs = new LinkedHashSet();
      hs.add("B");
      hs.add("A");
      hs.add("D");
      hs.add("E");
      hs.add("C");
      hs.add("F");
      System.out.println(hs);
    }
}
```

Example2:

```
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 LinkedHashSetExample {
public static void main(String[] args) {
    LinkedHashSet<Book> hs=new LinkedHashSet<Book>();
    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);
    hs.add(b1);
    hs.add(b2);
    hs.add(b3);
        for(Book b:hs){
    System.out.println(b.id+" "+b.name+" "+b.author+"
"+b.publisher+" "+b.quantity);
```

```
    }
  }
}
```

TreeSet Class

- TreeSet provides an implementation of the Set interface that uses a tree for storage.
- Objects are stored in a sorted and ascending order.
- Access and retrieval times are quite fast, which makes TreeSet an excellent choice when storing large amounts of sorted information that must be found quickly.

Methods of TreeSet Class

- void add(Object o): Adds the specified element to this set if it is not already present.
- boolean addAll(Collection c): Adds all of the elements in the specified collection to this set.
- void clear(): Removes all of the elements from this set.
- Object clone(): Returns a shallow copy of this TreeSet instance.
- Comparator comparator(): Returns the comparator used to order

this sorted set, or null if this tree set uses its elements natural ordering.

- boolean contains(Object o):Returns true if this set contains the specified element.
- Object first(): Returns the first (lowest) element currently in this sorted set.
- SortedSet headSet(Object toElement): Returns a view of the portion of this set whose elements are strictly less than toElement.
- boolean isEmpty(): Returns true if this set contains no elements.
- Iterator iterator(): Returns an iterator over the elements in this set.
- Object last(): Returns the last (highest) element currently in this sorted set.
- boolean remove(Object o): Removes the specified element from this set if it is present.
- int size(): Returns the number of elements in this set (its cardinality).
- SortedSet subSet(Object fromElement, Object toElement):

Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.

- SortedSet tailSet(Object fromElement): Returns a view of the portion of this set whose elements are greater than or equal to fromElement.

- NavigableSet subSet(Object fromElement, boolean fromInclusive, Object toElement, boolean toInclusive): It returns a set of elements that lie between the given range.

- NavigableSet tailSet(Object fromElement, boolean inclusive): It returns a set of elements that are greater than or equal to (if, inclusive is true) the specified element.

- NavigableSet headSet(Object toElement, boolean inclusive): It returns the group of elements that are less than or equal to(if, inclusive is true) the specified element.

- NavigableSet descendingSet(): It returns the elements in reverse order.

- Iterator descendingIterator(): It is used to iterate the elements in descending order.

- Object ceiling(Object e): It returns the equal or closest greatest element of the specified element from the set, or null there is no such element.

- Object floor(Object e): It returns the equal or closest least element of the specified element from the set, or null there is no such element.
- Object higher(Object e): It returns the closest greatest element of the specified element from the set, or null there is no such element.
- Object lower(Object e): It returns the closest least element of the specified element from the set, or null there is no such element.
- Object pollFirst(): It is used to retrieve and remove the lowest(first) element.
- Object pollLast(): It is used to retrieve and remove the highest(last) element.
- boolean contains(Object o): It returns true if this set contains the specified element.

Example:

```
import java.util.*;
public class TreeSetDemo {
  public static void main(String args[]) {
   TreeSet ts = new TreeSet();
    ts.add("C");
```

```
        ts.add("A");
        ts.add("B");
        ts.add("E");
        ts.add("F");
        ts.add("D");
        System.out.println(ts);
    }
}
```

Example2:

```
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;
}
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>();
    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);
```

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

Map Interface

- The Map interface in Java is a collection that maps keys to values.
- A map cannot contain duplicate keys; each key can only be mapped to one value.
- The Map interface is implemented by the HashMap, TreeMap, and LinkedHashMap classes.

Methods of Map Interface

- void clear( ): Removes all key/value pairs from the invoking map.

- boolean containsKey(Object k): Returns true if the invoking map contains k as a key. Otherwise, returns false.
- boolean containsValue(Object v): Returns true if the map contains v as a value. Otherwise, returns false.
- Set entrySet( ): Returns a Set that contains the entries in the map. The set contains objects of type Map.Entry. This method provides a set-view of the invoking map.
- boolean equals(Object obj): Returns true if obj is a Map and contains the same entries. Otherwise, returns false.
- Object get(Object k): Returns the value associated with the key k.
- int hashCode( ): Returns the hash code for the invoking map.
- boolean isEmpty( ): Returns true if the invoking map is empty. Otherwise, returns false.
- Set keySet( ): Returns a Set that contains the keys in the invoking map. This method provides a set-view of the keys in the invoking map.
- Object put(Object k, Object v): Puts an entry in the invoking map, overwriting any previous value associated with the key. The key and value are k and v, respectively. Returns null if the

key did not already exist. Otherwise, the previous value linked to the key is returned.

- void putAll(Map m):Puts all the entries from m into this map.
- Object remove(Object k): Removes the entry whose key equals k.
- int size( ): Returns the number of key/value pairs in the map.
- Collection values( ): Returns a collection containing the values in the map. This method provides a collection-view of the values in the map.

HashMap Class

- The HashMap class uses a hashtable to implement the Map interface.
- This allows the execution time of basic operations, such as get( ) and put( ), to remain constant even for large sets.
- HashMap is not synchronized.
- It allows us to store the null elements as well, but there should be only one null key.

Methods of HashMap Class

void clear()	It is used to remove all of the mappings from thi
boolean isEmpty()	It is used to return true if this map contains no k
Object clone()	It is used to return a shallow copy of this HashM and values themselves are not cloned.
Set entrySet()	It is used to return a collection view of the mapp map.
Set keySet()	It is used to return a set view of the keys contain
V put(Object key, Object value)	It is used to insert an entry in the map.
void putAll(Map map)	It is used to insert the specified map in the map.
V putIfAbsent(K key, V value)	It inserts the specified value with the specified k not already specified.
V remove(Object key)	It is used to delete an entry for the specified key.
boolean remove(Object key, Object value)	It removes the specified values with the associated map.
V compute(K key,	It is used to compute a mapping for the specified k

BiFunction<? super K,? super V,? extends V> remappingFunction)	mapped value (or null if there is no current mapping).
V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)	It is used to compute its value using the given map specified key is not already associated with a value (or is ma it into this map unless null.
V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)	It is used to compute a new mapping given the key value if the value for the specified key is present and nor
boolean containsValue(Object value)	This method returns true if some value equal to the map, else return false.
boolean containsKey(Object key)	This method returns true if some key equal to the k map, else return false.
boolean equals(Object o)	It is used to compare the specified Object with the
void	It performs the given action for each entry in the m

forEach(BiConsumer<? super K,? super V> action)	been processed or the action throws an exception.
V get(Object key)	This method returns the object that contains the va key.
V getOrDefault(Object key, V defaultValue)	It returns the value to which the specified key is m if the map contains no mapping for the key.
boolean isEmpty()	This method returns true if the map is empty; retur least one key.
V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)	If the specified key is not already associated with a with null, associates it with the given non-null value.
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a s
void replaceAll(BiFunction<	It replaces each entry's value with the result of invo on that

? super K,? super V,? extends V> function)	entry until all entries have been processed or the fu exception.
Collection<V> values()	It returns a collection view of the values contained
int size()	This method returns the number of entries in the m

Example:

```
import java.util.*;
public class HashMapDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Map<String, Integer> map = new HashMap<String,Integer>();
        System.out.println("Enter the number of key-value pairs:");
        int n = sc.nextInt();
        for (int i = 0; i < n; i++) {
            System.out.println("Enter the key:");
            String key = sc.next();
            System.out.println("Enter the value:");
            int value = sc.nextInt();
            map.put(key, value);
        }
        System.out.println("The map is:");
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
            System.out.println(entry.getKey() + " = " + entry.getValue());
        }
        System.out.println("Enter the key to serch:");
        String key=sc.next();
        if(map.containsKey(key)){
```

```
            System.out.println("Value of associated key
is:"+map.get(key));
        }
        else{
            System.out.println("key not found");
        }
        System.out.println("Enter the value to serch:");
        int value=sc.nextInt();
        if(map.containsValue(value)){
            System.out.println("Value is found");
        }
        else{
            System.out.println("value not found");
        }
        System.out.println("current size of the map is:"+map.size());
        Map<String,Integer> m=new HashMap<String,Integer>();
        m.put("ME",6);
        m.put("CE",7);
        m.put("EEE",8);
        map.putAll(m);
        System.out.println("After adding new pairs map is:");
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
           System.out.println(entry.getKey() + " = " + entry.getValue());
        }
        System.out.println("After adding new pairs size of the map
is:"+map.size());
        map.remove("CSE");
        System.out.println("After removing specified key map is:");
        for (Map.Entry<String, Integer> entry : map.entrySet()) {
           System.out.println(entry.getKey() + " = " + entry.getValue());
        }
```

```
        Set<String> keys = map.keySet();
        System.out.println("The keys of the map are:");
        for (String k : keys) {
            System.out.print(k+" ");
        }
        System.out.println();
        Collection values = map.values();
        System.out.println("The values of the map are:");
        for (Object v : values) {
            System.out.print(v+" ");
        }
    }
}
```

LinkedHashMap Class

- This class extends HashMap and maintains a linked list of the entries in the map, in the order in which they were inserted.
- This allows insertion-order iteration over the map.
- That is, when iterating a LinkedHashMap, the elements will be returned in the order in which they were inserted.
- You can also create a LinkedHashMap that returns its elements in the order in which they were last accessed.

Methods of LinkedHashMap Class

V get(Object key)	It returns the value to which the spe

	mapped.
void clear()	It removes all the key-value pairs fr
boolean containsValue(Object value)	It returns true if the map maps one specified value.
Set<Map.Entry<K,V>> entrySet()	It returns a Set view of the mapping map.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for eac until all entries have been processed or the action t
V getOrDefault(Object key, V defaultValue)	It returns the value to which the spe mapped or defaultValue if this map contains n key.
Set<K> keySet()	It returns a Set view of the keys conta
protected boolean removeEldestEntry(Map.Entry<K,V> eldest)	It returns true on removing its eldest
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the the given

	function on that entry until all entries processed or the function throws an exception.
Collection<V> values()	It returns a Collection view of the val map.

Example:

```
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 MapExample {

public static void main(String[] args) {

   Map<Integer,Book> map=new
LinkedHashMap<Integer,Book>();

   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);

   map.put(2,b2);

   map.put(1,b1);

   map.put(3,b3);

   for(Map.Entry<Integer, Book> entry:map.entrySet()){

      int key=entry.getKey();

      Book b=entry.getValue();

      System.out.println(key+" Details:");

      System.out.println(b.id+" "+b.name+" "+b.author+"
"+b.publisher+" "+b.quantity);

   }

}
```

}

TreeMap Class

- The TreeMap class implements the Map interface by using a tree.
- A TreeMap provides an efficient means of storing key/value pairs in sorted order, and allows rapid retrieval.
- You should note that, unlike a hash map, a tree map guarantees that its elements will be sorted in an ascending key order.
- Java TreeMap cannot have a null key but can have multiple null values.
- TreeMap is synchronized.

Methods of TreeMap Class

Map.Entry<K,V> ceilingEntry(K key)	It returns the key-value pair having the least key, specified key, or null if there is no such key.
K ceilingKey(K key)	It returns the least key, greater than the specified key.
void clear()	It removes all the key-value pairs from a map.
Object clone()	It returns a shallow copy of TreeMap instance.

Comparator<? super K> comparator()	It returns the comparator that arranges the key in the natural ordering.
NavigableSet<K> descendingKeySet()	It returns a reverse order NavigableSet view of t
NavigableMap<K,V> descendingMap()	It returns the specified key-value pairs in descen
Map.Entry firstEntry()	It returns the key-value pair having the least key.
Map.Entry<K,V> floorEntry(K key)	It returns the greatest key, less than or equal to th is no such key.
void forEach(BiConsumer<? super K,? super V> action)	It performs the given action for each entry in the m processed or the action throws an exception.
SortedMap<K,V> headMap(K toKey)	It returns the key-value pairs whose keys are strictl
NavigableMap<K,V> headMap(K toKey, boolean inclusive)	It returns the key-value pairs whose keys are less t true) toKey.

Map.Entry<K,V> higherEntry(K key)	It returns the least key strictly greater than the give key.
K higherKey(K key)	It is used to return true if this map contains a mapp
Set keySet()	It returns the collection of keys exist in the map.
Map.Entry<K,V> lastEntry()	It returns the key-value pair having the greatest key
Map.Entry<K,V> lowerEntry(K key)	It returns a key-value mapping associated with the the given key, or null if there is no such key.
K lowerKey(K key)	It returns the greatest key strictly less than the give key.
NavigableSet<K> navigableKeySet()	It returns a NavigableSet view of the keys containe
Map.Entry<K,V> pollFirstEntry()	It removes and returns a key-value mapping associ map, or null if the map is empty.
Map.Entry<K,V> pollLastEntry()	It removes and returns a key-value mapping associ this map, or null if the map is empty.

V put(K key, V value)	It inserts the specified value with the specified key
void putAll(Map<? extends K,? extends V> map)	It is used to copy all the key-value pair from one m
V replace(K key, V value)	It replaces the specified value for a specified key.
boolean replace(K key, V oldValue, V newValue)	It replaces the old value with the new value for a sp
void replaceAll(BiFunction<? super K,? super V,? extends V> function)	It replaces each entry's value with the result of invo entry until all entries have been processed or the fu
NavigableMap<K,V> subMap(K fromKey, boolean fromInclusive, K toKey, boolean toInclusive)	It returns key-value pairs whose keys range from fi
SortedMap<K,V> subMap(K fromKey, K toKey)	It returns key-value pairs whose keys range from fi exclusive.
SortedMap<K,V>	It returns key-value pairs whose keys are greater th

tailMap(K fromKey)	
NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)	It returns key-value pairs whose keys are greater th true) fromKey.
boolean containsKey(Object key)	It returns true if the map contains a mapping for th
boolean containsValue(Object value)	It returns true if the map maps one or more keys to
K firstKey()	It is used to return the first (lowest) key currently i
V get(Object key)	It is used to return the value to which the map map
K lastKey()	It is used to return the last (highest) key currently i
V remove(Object key)	It removes the key-value pair of the specified key f
Set<Map.Entry<K,V>> entrySet()	It returns a set view of the mappings contained in t
int size()	It returns the number of key-value pairs exists in th
Collection values()	It returns a collection view of the values contained

Example:

```
import java.util.*;
class NavigableMapExample{
 public static void main(String args[]){
   NavigableMap<Integer,String> map=new
TreeMap<Integer,String>();
      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      System.out.println("descendingMap: "+map.descendingMap());
      System.out.println("headMap: "+map.headMap(102,true));
      System.out.println("tailMap: "+map.tailMap(102,true));
      System.out.println("subMap: "+map.subMap(100, false, 102,
true));
 }
}
```

Example2:

```
import java.util.*;
```

```
class SortedMapExample{
 public static void main(String args[]){
   SortedMap<Integer,String> map=new
TreeMap<Integer,String>();
      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      System.out.println("headMap: "+map.headMap(102));
      System.out.println("tailMap: "+map.tailMap(102));
      System.out.println("subMap: "+map.subMap(100, 102));
   }
}
```

Queue Interface

- The Queue interface is provided in java.util package and it implements the Collection interface.
- The queue implements FIFO i.e. First In First Out.
- This means that the elements entered first are the ones that are deleted first.

- A queue is generally used to hold elements before processing them.
- Once an element is processed then it is removed from the queue and next item is picked for processing.
- Queue is implemented by PriorityQueue class and sometime using LinkedList also.

Methods of Queue Interface

Sr.No.	Method & Description
1	boolean add(E e)This method inserts the specified element into th queue if it is possible to do so immediately without violating capa restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
2	E element()This method retrieves, but does not remove, the head this queue.
3	boolean offer(E e)This method inserts the specified element into t queue if it is possible to do so immediately without violating capa restrictions.
4	E peek()This method retrieves, but does not remove, the head of t queue, or returns null if this queue is empty.
5	E poll()This method retrieves and removes the head of this queue

returns null if this queue is

empty.

6 E remove()This method retrieves and removes the head of this qu

PriorityQueue Class

- The java.util.PriorityQueue class is an unbounded priority queue based on a priority heap.Following are the important points about PriorityQueue −
- The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.
- A priority queue does not permit null elements.
- A priority queue relying on natural ordering also does not permit insertion of non-comparable objects.

Methods of PriorityQueue Class

- boolean add(E e): This method inserts the specified element into this priority queue.
- void clear(): This method removes all of the elements from this priority queue.
- Comparator<? super E> comparator(): This method returns the

comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements.

- boolean contains(Object o): This method returns true if this queue contains the specified element.
- Iterator<E> iterator(): This method returns an iterator over the elements in this queue.
- boolean offer(E e): This method inserts the specified element into this priority queue.
- E peek():This method retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
- E poll(): This method retrieves and removes the head of this queue, or returns null if this queue is empty.
- boolean remove(Object o): This method removes a single instance of the specified element from this queue, if it is present.
- int size(): This method returns the number of elements in this collection.
- Object[] toArray(): This method returns an array containing all of the elements in this queue.

- <T> T[] toArray(T[] a): This method returns an array containing all of the elements in this queue; the runtime type of the returned array is that of the specified array.

Example:

```
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;

}

public int compareTo(Book b) {

    if(id>b.id){

        return 1;

    }else if(id<b.id){
```

```
        return -1;
    }else{
    return 0;
    }
}
}
public class LinkedListExample {
public static void main(String[] args) {
    Queue<Book> queue=new PriorityQueue<Book>();
    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);
    queue.add(b1);
    queue.add(b2);
    queue.add(b3);
    System.out.println("Traversing the queue elements:");
    for(Book b:queue){
    System.out.println(b.id+" "+b.name+" "+b.author+"
```

```
"+b.publisher+" "+b.quantity);
    }
    queue.remove();
    System.out.println("After removing one book record:");
    for(Book b:queue){
        System.out.println(b.id+" "+b.name+" "+b.author+"
"+b.publisher+" "+b.quantity);
        }
  }
}
```

----------Endof4th Unit Part-2----------