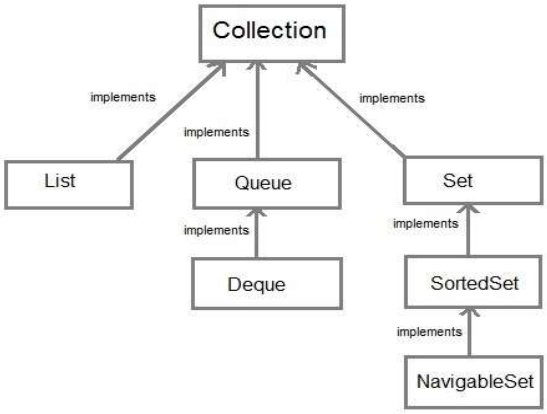
Collection Framework: The Collection framework in Java provides architecture to store many values or objects in a single unit. You can apply all sorts of operations required to be performed on data, such as adding, removing, altering, searching, and sorting using the Collection framework method. The Java Collection Framework provides the following interfaces**:**



ArrayList: ArrayList is a part of collection framework and is present in java.util package. It provides us with dynamic arrays in Java. Though, it may be slower than standard arrays but can be helpful in programs where lots of manipulation in the array is needed. This class is found in java.util package.

For Example:

```
Importjava.util.ArrayList;
Importjava.util.List;
Importjava.util.*;
class list1
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
System.out.println(l);
        }
}
```

List: List in Java provides the facility to maintain the ordered collection. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.The List interface is found in the java.util package and inherits the Collection interface. It is a factory of ListIterator interface. Through the ListIterator,we can iterate the list in forward and backward directions. The implementation classes of List interface are ArrayList, LinkedList, Stack and Vector. The ArrayList and LinkedList are widely used in Java programming. The Vector class is deprecated since Java.

- List is an index based data structure.
- List can store duplicate elements.
- List can store any numbers of null values.
- List follows the insertion order.
- We can iterate(get) the list elements by Iterate &ListIterator.

For Example:

```
importjava.util.ArrayList;
importjava.util.List;
importjava.util.*;
class nest
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
        Iterator itr=l.iterator();
while(itr.hasNext())
        {
System.out.println(itr.next());
        }
        }
}
```

Iterator(): Iterator in Java is an object used to cycle through arguments or elements present in a collection. It is derived from the technical term "iterating," which means looping through. Generally, an iterator in Java is used to loop through any collection of objects. To apply the iterator, all you need to do is import the java.util package and then use the iterator() method. You can then use the iterator to perform multiple operations in the collection.

For Example:

```
importjava.util.ArrayList;
importjava.util.List;
importjava.util.*;
class nest
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
        Iterator itr=l.iterator();
while(itr.hasNext())
        {
System.out.println(itr.next());
        }
        }
}
```

Program(In this program add the values and print after the update list).

For Example:

```
import java.util.ArrayList;
import java.util.List;
import java.util.*;
class list1
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
System.out.println(l);
System.out.println("----------------------");
System.out.println(l.get(0));
System.out.println(l.get(1));
System.out.println(l.get(2));
System.out.println("----------------------");
l.set(0,100);
l.set(1,"gagan");
l.set(2,"meena_kumari");
System.out.println(l);
System.out.println("----------------------");
        Iterator itr=l.iterator();
while(itr.hasNext())
        {
System.out.println(itr.next());
        }
    }
```

```
}
```

ListIterator(): ListIterator is an interface in Collection API. It extends Iterator interface. To support Forward and Backward Direction iteration and CRUD operations, it has the following methods. We can use this Iterator for all List implemented classes like ArrayList, CopyOnWriteArrayList, LinkedList, Stack, Vector, etc.

For Example:(In this program use the add method in listIterator())

```
importjava.util.*;
class list{
public static void main(String[] args)
    {
List l=new ArrayList();
l.add(10);
l.add(25);
System.out.println(l);
ArrayList l2=new ArrayList();
l2.add("java");
l2.add("html");
l2.add("css");
System.out.println(l2);
l.addAll(l2);
System.out.println(l);
    }
}
```

Sort() : There are various ways to sort the List, here we are going to use Collections.sort() method to sort the list element. The java.util package provides a utility class Collections which has the static method sort(). Using the Collections.sort() method, we can easily sort any List**.**

For Example:

```
importjava.util.List;
importjava.util.*;
class list1
{
public static void main(String[] args)
    {
        List<String> l=new ArrayList<String>();

l.add("deepak");
```

```
l.add("rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");

Collections.sort(l);
for(String hit:l)
System.out.println(hit);
System.out.println("----------------------");
    }
}
```

For Example:Note(in this program use the integer values).

```
importjava.util.List;
importjava.util.*;
class list2
{
public static void main(String[] args)
    {
        List<Integer> l=new ArrayList<Integer>();

l.add(23);
l.add(12);
l.add(56);
l.add(36);
l.add(12);
l.add(9);
l.add(75);
l.add(45);
l.add(33);
l.add(22);
Collections.sort(l);
for(inthit:l)
System.out.println(hit);
System.out.println("----------------------");
```

```
    }
}
```

Remove():remove(Object): This method is used to simply remove an object from the List. If there are multiple such objects, then the first occurrence of the object is removed.

remove(int index): Since a List is indexed, this method takes an integer value which simply removes the element present at that specific index in the List. After removing the element, all the elements are moved to the left to fill the space and the indices of the objects are updated.

For Example:

```
import java.util.ArrayList;
import java.util.List;
import java.util.*;
class list1
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
System.out.println(l);
System.out.println("----------------------");
System.out.println(l.get(0));
System.out.println(l.get(1));
System.out.println(l.get(2));
System.out.println("----------------------");
l.set(0,100);
```

```
l.set(1,"gagan");
l.set(2,"meena_kumari");
System.out.println(l);
System.out.println("----------------------");
        l.remove(2);
        l.remove(3);
        l.remove(4);
        l.remove(5);
        l.remove(7);

        System.out.println(l);
    }
}
```

Size(): In this program check the length of array.

For Example:

```
import java.util.ArrayList;
import java.util.List;
import java.util.*;
class list1
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
System.out.println(l);
```

```
int s=l.size();
        System.out.println("Lenght of array is   "+s);
System.out.println("---------------------");
        l.remove(2);
        l.remove(3);
        l.remove(4);
        l.remove(5);
        l.remove(7);
        System.out.println(l);
        System.out.println("---------------------");
        int s=l.size();
        System.out.println("Length of array is  "+s);
    }
}
```

Print the Previous value and next():

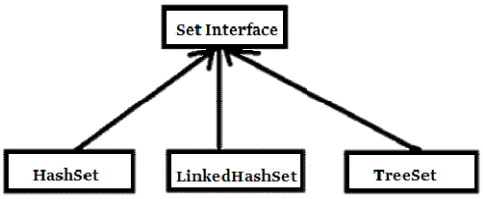
For Example:

```
import java.util.ArrayList;
import java.util.List;
import java.util.*;
class list1
{
public static void main(String[] args)
    {
        List l=new ArrayList();
l.add(10);
l.add("deepak");
l.add("Rahul");
l.add("java");
l.add("html");
l.add("css");
l.add("python");
l.add("java");
l.add("css");
l.add(10);
l.add(20);
l.add(20);
```

```
ListIterator itr=l.listIterator();
while(itr.hasNext())
        {
        System.out.println("index:  "+itr.nextIndex()+" value:   "+itr.next());
            }

int size=l.size();
System.out.println("Lenght of array is   "+size);
System.out.println("---------------------");


        while(itr.hasPrevious())
        {
        System.out.println("index:             "+itr.previousIndex()+"    value:
"+itr.previous());
            }
    System.out.println("end");

    }
}
```

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

- Set is not an index based data Structure. It stores the data according to the hashcode values.
- Set dose not allow to store duplicate elements.
- Set can store only one null values.
- Set does not follows the insertion order.
- We can iterate the Set elements by Iterator.



For Example:

```
Importjava.util.*;
class set
{
public static void main(String[] args)
    {
        Set s=new HashSet();
```

```
s.add(100);
s.add(200);
s.add(300);
s.add(400);
s.add(500);
        //s.add(100);
        //s.add(200);
s.add(250);
System.out.println(s);
    }
}
```

For Example: Note(In this program used the Iterator () method in set).

```
importjava.util.*;
class set
{
public static void main(String[] args)
    {
        Set s=new HashSet();
s.add(100);
s.add(200);
s.add(300);
s.add(400);
s.add(500);
        //s.add(100);
        //s.add(200);
s.add(250);
 Iterator itr=s.iterator();
while(itr.hasNext())
         {
System.out.println(itr.next());
         }
    }
}
```

Add(): In this program use the add() and merge the two list in one list.

For Example:

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

```
class rat
{
    public static void main(String[] args) //throws Exception
    {
        Set h=new HashSet();
        h.add(100);
        h.add(200);
        h.add(300);
        h.add(400);
        System.out.println(h);
        Iterator itr=h.iterator();
        while(itr.hasNext())
         {
System.out.println(itr.next());
         }
         System.out.println("-------------");
         Set<String> g=new HashSet<String>();
         g.add("hello");
         g.add("World");
         System.out.println("Second set is     "+g);
         h.addAll(g);
         System.out.println("-----------------");
         System.out.println("All the content is merging  "+ h);

    }
}
```

Clear(): In this program used the clear() method then clear all the content in set.

For Example:

```
import java.util.*;
class rat
{
    public static void main(String[] args) //throws Exception
    {
        Set h=new HashSet();
        h.add(100);
        h.add(200);
```

```
        h.add(300);
        h.add(400);
        System.out.println(h);
        Iterator itr=h.iterator();
        while(itr.hasNext())
         {
System.out.println(itr.next());
         }
         System.out.println("-------------");
         Set<String> g=new HashSet<String>();
         g.add("hello");
         g.add("World");
         System.out.println("Second set is     "+g);
         h.addAll(g);
         System.out.println("-----------------");
         System.out.println("All the content is merging  "+ h);
         g.clear();
         System.out.println(g);

    }
}
```

Comparator: Java Comparator is an interface for sorting Java objects. Invoked by "java. util. comparator," Java Comparator compares two Java objects in a "compare(Object 01, Object 02)" format. Using configurable methods, Java Comparator can compare objects to return an integer based on a positive, equal or negative comparison.

For Example:

```
import java.util.*;
import java.util.io.*;
class Student{
int rollno;
String name;
int age;
Student(int rollno,String name,int age){
this.rollno=rollno;
this.name=name;
```

```
this.age=age;
}
}

class AgeComparator implements Comparator
{
public int compare(Object o1,Object o2){
Student s1=(Student)o1;
Student s2=(Student)o2;

if(s1.age==s2.age)
return 0;
else if(s1.age>s2.age)
return 1;
else
return -1;
}
}
class NameComparator implements Comparator{
    public int compare(Object o1,Object o2)
    {
    Student s1=(Student)o1;
    Student s2=(Student)o2;

    return s1.name.compareTo(s2.name);
    }
    }
    class Rollno implements Comparator
    {
        public int compare(Object o1, Object o2)
        {
        Student s1=(Student)o1;
        Student s2=(Student)o2;
         if(s1.rollno==s2.rollno)
         return 0;
         else if(s1.rollno>s2.rollno)
         return 1;
```

```
        else
        return -1;
    }
    }
    class Simple{
        public static void main(String args[])
        {

        ArrayList al=new ArrayList();
        al.add(new Student(101,"Vijay",23));
        al.add(new Student(106,"Ajay",27));
        al.add(new Student(105,"Jai",21));

        System.out.println("Sorting by Name");

        Collections.sort(al,new NameComparator());
        Iterator itr=al.iterator();
        while(itr.hasNext())
        {
        Student st=(Student)itr.next();
        System.out.println(st.rollno+"  "+st.name+"  "+st.age);
        }

        System.out.println("Sorting by age");
        System.out.println("------------------------");

        Collections.sort(al,new AgeComparator());
        Iterator itr2=al.iterator();
        while(itr2.hasNext())
        {
        Student st=(Student)itr2.next();
        System.out.println(st.rollno+" "+st.name+" "+st.age);
        }

         System.out.println("--------------");
        System.out.println("Sorting by rollno ");
        Collections.sort(al,new Rollno());
```

```
Iterator itr3=al.iterator();
while(itr3.hasNext())
{
    Student st=(Student)itr3.next();
    System.out.println(st.rollno+"  "+st.name+"  "+st.age);

}
}
}
```

Legacy Classes: Vector class is similar to the ArrayList class but there are certain differences. Vector is generally synchronized. It is used where the programmer doesn't really have knowledge about the length of the Array.

- **Dictionary.**
- **Properties.**
- **HashTable.**
- **Vector.**
- **Stack.**

Method	Description
E elementAt(int index)	This method returns the element at the specified index
E firstElement()	It helps to return the first element in the Vector
Enumeration elements()	This helps to return an enumeration of the element in the vector
E lastElement()	Returns the last element in the Vector
void removeAllElements()	It helps in removing all the elements of the Vector

For Example:

```
class Test
{
public static void main(String[] args)
{
Vector ve = new Vector();
ve.add(1);
ve.add(2);
ve.add(3);
ve.add(4);
ve.add(5);
ve.add(6);
```

```
Enumeration en = ve.elements();
while(en.hasMoreElements())
{
System.out.println(en.nextElement());
}
}
}
```

Hashtable: Like HashMap, Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want linked to that key. The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

For Example:

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

class hashtb {
    public static void main(String args[])
    {

        Hashtable<Integer, String> ht1 = new Hashtable<>();

        Hashtable<Integer, String> ht2
            = new Hashtable<Integer, String>();

        ht1.put(1, "one");
        ht1.put(2, "two");
        ht1.put(3, "three");
        ht1.put(4,"Four");

        ht2.put(4, "four");
        ht2.put(5, "five");
        ht2.put(6, "six");


        System.out.println("Mappings of ht1 : " + ht1);
        System.out.println("Mappings of ht2 : " + ht2);
    }
```

```
}
```

Vector: Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index. They are very similar to ArrayList, but Vector is synchronized and has some legacy methods that the collection framework does not contain. Vector uses the List interface to create resizable arrays.

Syntax:

Vector<DataType> vector_name = new Vector<>();

- Vector tells our program we want to declare a vector.
- DataType is the type of data our vector will store.
- vector_name is the name of our vector.

new Vector<>(); creates a new vector and assigns it to the vector_name variable.For instance, suppose we wanted to declare a vector that stores all the colors of lamps our department store sells. We could use this code to declare the vector:

Vector<String> lamp_colors = new Vector>?();

For Example:

```
import java.util.Vector;
class domy
{
    public static void main(String[] args)
    {
        Vector<String> k=new Vector<>();
        k.add("Orange");
        k.add("Mango");
        k.add("Apple");
        k.add("Grapes");
        System.out.println("Vector is :  "+k);
    }
}
```

Changing Elements: After adding the elements if we wish to change the element, it can be done by again adding the element with the put() method. Since the elements in the hashtable are indexed using the keys, the value of the key can be changed by simply inserting the updated value for the key for which we wish to change.

For Example:

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

class hashtb {
    public static void main(String args[])
    {

        Hashtable<Integer, String> ht1 = new Hashtable<>(4);

        Hashtable<Integer, String> ht2
            = new Hashtable<Integer, String>();

        ht1.put(1, "one");
        ht1.put(2, "two");
        ht1.put(3, "three");
        ht1.put(4,"Four");

        ht2.put(4, "four");
        ht2.put(5, "five");
        ht2.put(6, "six");


        System.out.println("Mappings of ht1 : " + ht1);
        System.out.println("Mappings of ht2 : " + ht2);
        System.out.println("-----------");
        ht1.put(2,"Five");
        ht1.put(3,"Six");
        System.out.println("After updating ht1 is  : " + ht1);
    }
}
```

Remove(): In order to remove an element from the Map, we can use the remove() method. This method takes the key value and removes the mapping for a key from this map if it is present in the map.

For Example:

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

```
class hashtb {
    public static void main(String args[])
    {
        Hashtable<Integer, String> ht1 = new Hashtable<>(4);
        Hashtable<Integer, String> ht2
            = new Hashtable<Integer, String>();
 ht1.put(1, "one");
 ht1.put(2, "two");
 ht1.put(3, "three");
 ht1.put(4,"Four");
 ht2.put(4, "four");
 ht2.put(5, "five");
 ht2.put(6, "six");
 System.out.println("Mappings of ht1 : " + ht1);
 System.out.println("Mappings of ht2 : " + ht2);
 System.out.println("-----------");
        ht1.put(2,"Five");
        ht1.put(3,"Six");
        System.out.println("After updating ht1 is  : " + ht1);
        ht1.remove(2);
        ht1.remove(4);
        System.out.println("After removing the updating ht1 is  : " + ht1);
    }
}
```

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.

For Example:

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

```
class hash{
 public static void main(String args[]){
 HashSet<String> set=new HashSet<String>();
           set.add("Ravi");
           set.add("Vijay");
           set.add("Arun");
           set.add("Sumit");
           System.out.println("An initial list of elements: "+set);
           //Removing specific element from HashSet
           set.remove("Ravi");
           System.out.println("After invoking remove(object) method: "+set);
           HashSet<String> set1=new HashSet<String>();
           set1.add("Ajay");
           set1.add("Gaurav");
           set.addAll(set1);
           System.out.println("Updated List: "+set);
           //Removing all the new elements from HashSet
           set.removeAll(set1);
           System.out.println("After invoking removeAll() method: "+set);
           //Removing elements on the basis of specified condition
           set.removeIf(str->str.contains("Vijay"));
           System.out.println("After invoking removeIf() method: "+set);
           //Removing all the elements available in the set
           set.clear();
           System.out.println("After invoking clear() method: "+set);
 }
}
```

Stack: The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

For Example:

```
import java.util.*;
class hash{
  public static void main(String args[]){
    Stack<String> stack = new Stack<String>();
    stack.push("Ayush");
```

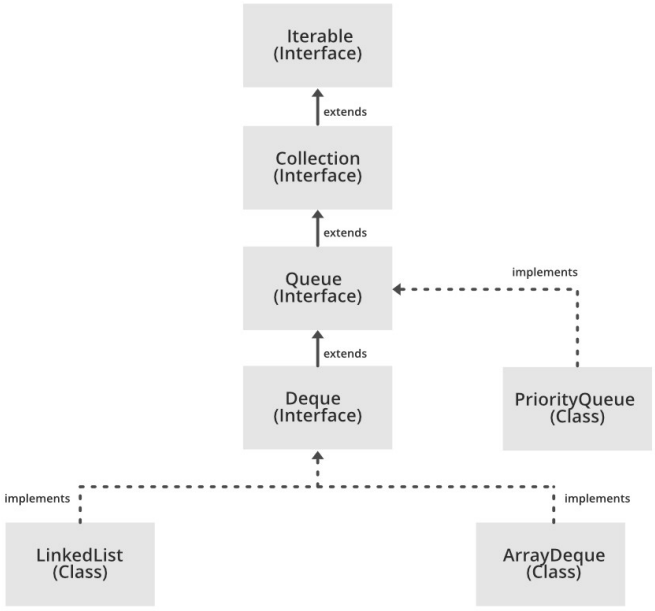
```
stack.push("Garvit");
stack.push("Amit");
stack.push("Ashish");
stack.push("Garima");
stack.pop();
stack.pop();
Iterator<String> itr=stack.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

For Exmple: In this program use the stack and enumeration elements().

```
class Stack{
public static void main(String args[]) {
Stack st = new Stack();
st.push(1);
st.push(2);
st.push(3);
st.push(4);
st.push(5);
Enumeration e1 = st.elements();
while(e1.hasMoreElements())
System.out.print(e1.nextElement()+" ");
st.pop();
st.pop();
System.out.println("nAfter popping out one element”);
Enumeration e2 = st.elements();
while(e2.hasMoreElements())
System.out.print(e2.nextElement()+" ");
}
}
```

Queue: The Queue interface is present in java.util package and extends the Col006Cection interface is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects

with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the FIFO or the First-In-First-Out principle.



PriorityQueue: The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

For Example:

```
import java.util.*;
class hash{
  public static void main(String args[]){
    PriorityQueue<String> queue=new PriorityQueue<String>();
    queue.add("Amit Sharma");
    queue.add("Vijay Raj");
    queue.add("JaiShankar");
    queue.add("Raj");
    System.out.println("head:"+queue.element());
    System.out.println("head:"+queue.peek());
    System.out.println("iterating the queue elements:");
    Iterator itr=queue.iterator();
    while(itr.hasNext()){
    System.out.println("*  " + itr.next());
    }
    System.out.println(queue);
    queue.remove();
    System.out.println(queue);
    queue.poll();
    System.out.println(queue);
    Iterator<String> itr2=queue.iterator();
    while(itr2.hasNext()){
    System.out.println(itr2.next());
    }
    }
```

```
}
```

Deque Interface: Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

ArrayDeque: ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends. ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

For Example:

```
import java.util.*;
class hash{

    public static void main(String[] args)
    {
        Deque<String> deque
            = new LinkedList<String>();

        // We can add elements to the queue
        // in various ways

        // Add at the last
        deque.add("Element 1 (Tail)");

        // Add at the first
        deque.addFirst("Element 2 (Head)");

        // Add at the last
        deque.addLast("Element 3 (Tail)");

        // Add at the first
        deque.push("Element 4 (Head)");

        // Add at the last
        deque.offer("Element 5 (Tail)");

        // Add at the first
```

```
        deque.offerFirst("Element 6 (Head)");

        System.out.println(deque + "\n");

        // We can remove the first element
        // or the last element.
        deque.removeFirst();
        deque.removeLast();
        System.out.println("Deque after removing " + "first and last: "+
deque);
    }
  }
```

Adding Elements: In order to add an element in a deque, we can use the add() method. The difference between a queue and a deque is that in deque, the addition is possible from any direction. Therefore, there are other two methods available named addFirst() and addLast() which are used to add the elements at either end.

For Example:

```
import java.util.*;
class hash{

 public static void main(String[] args)
  {
      // Initializing an deque
      Deque<String> dq
          = new ArrayDeque<String>();

      // add() method to insert
      dq.add("For");
      dq.addFirst("Advance Java ");
      dq.addLast(" Creat a Core-Java ");
      System.out.println(dq);
  }
}
```

Removing Elements: In order to remove an element from a deque, there are various methods available. Since we can also remove from both the ends, the deque interface provides us with removeFirst(),

removeLast() methods. Apart from that, this interface also provides us with the poll(), pop(), pollFirst(), pollLast() methods where pop() is used to remove and return the head of the deque. However, poll() is used because this offers the same functionality as pop() and doesnt return an exception when the deque is empty.

For Example:

```
import java.util.*;
class hash{
  public static void main(String[] args)
  {
      // Initializing an deque
      Deque<String> dq
          = new ArrayDeque<String>();
      // add() method to insert
      dq.add("Core");
      dq.addFirst("Hello ");
   dq.addLast("Java");
   System.out.println(dq);
      System.out.println(dq.pop());
      System.out.println(dq.poll());
      System.out.println(dq.pollFirst());
      System.out.println(dq.pollLast());
  }
}
```

TreeSet: Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

For Example:

```
import java.util.*;
class hash{
  public static void main(String[] args)
  {
    TreeSet<String> set=new TreeSet<String>();
    set.add("Kamal");
    set.add("Vijay");
    set.add("Ravi");
```

```
    set.add("Ajay");
    //traversing elements
    Iterator<String> itr=set.iterator();
    while(itr.hasNext()){
    System.out.println(itr.next());
    }
  }
}
```

For Example: In this program using the Navigable Set.

```
import java.util.NavigableSet;
import java.util.TreeSet;
class Main {
    public static void main(String[] args) {
        // Creating NavigableSet using the TreeSet
        NavigableSet<Integer> numbers = new TreeSet<>();
        // Insert elements to the set
        numbers.add(1);
        numbers.add(2);
        numbers.add(3);
        System.out.println("NavigableSet: " + numbers);
        // Access the first element
        int firstElement = numbers.first();
        System.out.println("First Number: " + firstElement);
        // Access the last element
        int lastElement = numbers.last();
        System.out.println("Last Element: " + lastElement);
        // Remove the first element
        int number1 = numbers.pollFirst();
        System.out.println("Removed First Element: " + number1);
        // Remove the last element
        int number2 = numbers.pollLast();
        System.out.println("Removed Last Element: " + number2);
    }
}
```