HashMap Class:

```
package collection.com;


import java.util.LinkedHashMap;

import java.util.Map;


public class HashMapClass {

        public static void main(String args[]){


                LinkedHashMap<Integer,String> hm=new LinkedHashMap<Integer,String>();


                hm.put(100,"Amit");

                hm.put(101,"Vijay");

                hm.put(99,"Rahul");


               for(Map.Entry m:hm.entrySet()){

                 System.out.println(m.getKey()+" "+m.getValue());

                }

               }

}
```

Hash Set Class:

```
package collection.com;


import java.util.HashSet;

import java.util.Iterator;

import java.util.LinkedHashSet;
```

```
public class HashSetClass {

    public static void main(String args[]){

        HashSet<String> al=new HashSet<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());
        }

        System.out.println("linked hash set");
        // linked
        LinkedHashSet<String> al2=new LinkedHashSet<String>();
        al2.add("Bavi");
        al2.add("Cijay");
        al2.add("Bavi");
        al2.add("Ajay");

        Iterator<String> itr2=al2.iterator();
        while(itr2.hasNext()){
         System.out.println(itr2.next());
```

```
              }

             }



}
```

Linklist ArrayList :

```
package collection.com;

import java.util.*;

public class LinkListArrayList {
        public static void main(String args[]){
              ArrayList a;
              LinkedList ba;
               List<String> al=new ArrayList<String>();//creating arraylist
               al.add("Ravi");//adding object in arraylist
               al.add("Vijay");
               al.add("Ravi");
               al.add("Ajay");

               List<String> al2=new LinkedList<String>();//creating linkedlist
               al2.add("James");//adding object in linkedlist
               al2.add("Serena");
               al2.add("Swati");
               al2.add("Junaid");

               System.out.println("arraylist: "+al);
               System.out.println("linkedlist: "+al2);
               Queue<String> queue=new LinkedList<String>();
              }
}
```

MyQueue:

```
package collection.com;

import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Queue;

public class myQueue {

      public static void main(String[] args) {
             // TODO Auto-generated method stub
             System.out.println("basic queue");
 Queue<Integer> q= new LinkedList<Integer> ();
 q.add(15);
 q.add(11);
 q.add(10);
 q.add(14);
 System.out.println(q.poll());

 System.out.println("prority queue");
 PriorityQueue<Integer> qp= new PriorityQueue<Integer> ();
 qp.add(15);
 qp.add(11);
 qp.add(10);
 qp.add(14);
 System.out.println(qp.poll());
 System.out.println(qp.poll());

 System.out.println("prority queue for students");
 PriorityQueue<Student> pStudends= new PriorityQueue<Student> ();
 pStudends.add(new Student("Hussein", 27));
 pStudends.add(new Student("Jena", 2));
 pStudends.add(new Student("Laya", 1));
 System.out.println(pStudends.poll().name);
    }

}
```

Mystack:

```
package collection.com;

import java.util.Stack;

public class myStack {

    public static void main(String[] args) {

        Stack<String> st= new Stack<String>();
        st.push("Admins");
        st.push("Manager");
        st.push("Tester");
        System.out.println(st.pop());
    }

}
```

Student:

```
package collection.com;

public class Student  implements Comparable<Student> {
    String name;
    int age;
        public Student(String name,int age) {
            this.name=name;
            this.age=age;
        }

        @Override
        public int compareTo(Student o) {
               if(this.age==o.age)
                   return 0;
               else if(this.age> o.age)
                   return 1;
               else
                   return -1;
        }

}
```

Tree set example:

```
package collection.com;
```

```
import java.util.Iterator;
import java.util.PriorityQueue;
import java.util.TreeSet;

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

               TreeSet<String> al=new TreeSet<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());
               }

               // object
               System.out.println("*************** object");
               TreeSet<NodeS> s= new TreeSet<NodeS>();
                 s.add(new NodeS(61,null));
                    s.add(new NodeS(3,null));
                    s.add(new NodeS(6,null));
                    s.add(new NodeS(4,null));
                      Iterator<NodeS> itr2=s.iterator();
                      while(itr2.hasNext()){
                       System.out.println(itr2.next().cost);
                      }
              }
}


class NodeS implements Comparable<NodeS>{
        int cost;
        NodeS next;
        public NodeS(int cost, NodeS next){
               this.cost=cost;
               this.next=next;
        }
       @Override
       public int compareTo(NodeS o) {
             // TODO Auto-generated method stub
             if (this.cost>o.cost )
                    return 1;
             else
             return 0;
       }


}
```

Vector example:

```
package collection.com;

import java.util.*;
```

```
public class VectorExample {

   public static void main(String args[]) {
      /* Vector of initial capacity(size) of 2 */
      Vector<String> vec = new Vector<String>(2);

      /* Adding elements to a vector*/
      vec.addElement("Apple");
      vec.addElement("Orange");
      vec.addElement("Mango");
      vec.addElement("Fig");

      /* check size and capacityIncrement*/
      System.out.println("Size is: "+vec.size());
      System.out.println("Default capacity increment is: "+vec.capacity());

      vec.addElement("fruit1");
      vec.addElement("fruit2");
      vec.addElement("fruit3");

      /*size and capacityIncrement after two insertions*/
      System.out.println("Size after addition: "+vec.size());
      System.out.println("Capacity after increment is: "+vec.capacity());

      /*Display Vector elements*/
      Enumeration en = vec.elements();
      System.out.println("\nElements are:");
      while(en.hasMoreElements())
         System.out.print(en.nextElement() + " ");
   }
   Vector<String> vectask = new Vector<String>(2);


}
```

Searching:
Binary Search:

```
package com.search;

public class BinarySearch {

       public static void main(String[] args) {
             DataSet data= new DataSet(1000000);
             int Search=999999;
             Boolean IsFound=false;
             int low=0;
             int high=data.getSize()-1;
             int mid=0;
             while( !IsFound){
                    if(low>high){
                           System.out.println("number isnot found");
                           break;
                    }
                    mid=low+((high-low)/2);
                    data.NumberTry++;
                    if(data.data[mid]==Search){
                           System.out.println("number is found after "+
            data.NumberTry + "try");
                break;
            }
            if( data.data[mid]< Search)
                low=mid+1;
            if( data.data[mid]> Search)
                high=mid-1;

        }
    }

}
```

Data set:

```
package com.search;

public class DataSet {
int[] data;
int NumberTry;
    public DataSet(int size) {
        data= new int[size];
        for(int i=1;i<=size;i++)
            data[i-1]=i;
        NumberTry=0;
    }
    public int getSize(){
        return data.length;
    }

}
```

Interpolation search:

```
package com.search;

public class InterploationSearch {

    public static void main(String[] args) {
        DataSet data= new DataSet(1000000);
        int Search=999999;
        Boolean IsFound=false;
        int low=0;
        int high=data.getSize()-1;
        int mid=0;
        while( !IsFound){
            if(low>high){
                System.out.println("number isnot found");
                break;
            }
            mid=low+((high-low)/(data.data[high]-
                    data.data[low]))*(Search-data.data[low]);
            data.NumberTry++;
            if(data.data[mid]==Search){
                System.out.println("number is found after "+
            data.NumberTry + "try");
```

```
                break;
            }
            if( data.data[mid]< Search)
                low=mid+1;
            if( data.data[mid]> Search)
                high=mid-1;

        }
    }

}
```

Linear search:

```
package com.search;

public class LinearSearch {

    public static void main(String[] args) {
        DataSet data= new DataSet(1000000);
        int Search=999999;
        Boolean IsFound=false;
        for(int i=0;i< data.getSize();i++){
            data.NumberTry++;
            if(data.data[i]==Search){
                System.out.println("Element is found after "
                        + data.NumberTry + "  try");
                IsFound=true;
                break;
            }
        }
        if(IsFound==false){
            System.out.println("number isnot found");
        }

    }

}
```

Bubble sorting:

```
package com.sort;

public class BubbleSorting {

    static void BubbleSort(int[] arr){
        int n= arr.length;
        int temp=0;
        for(int i=0;i<n;i++)
            for(int j=1;j<(n-i);j++){

                if(arr[j-1]>arr[j]){
                    temp=arr[j-1];
                    arr[j-1]=arr[j];
                    arr[j]= temp;
                }
```

```
                    }
        }


        public static void main(String[] args) {
         int[] arr={1,50,30,10,60,80};
         System.out.println("Before Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
         BubbleSort(arr);
         System.out.println("\nAfter Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
        }

}
```

Heap sort:

```
package com.sort;

public class HeapSort {
static int total;
static void swap(Comparable[] arr, int a, int b){
        Comparable temp= arr[a];
        arr[a]= arr[b];
        arr[b]= temp;
}
static void heapify(Comparable[] arr, int i){
        int lft= i*2;
        int rgt=i*2+1;
        int grt=i;
        if( lft<= total && arr[lft].compareTo(arr[grt])>=0)
                grt=lft;
        if( rgt<= total && arr[rgt].compareTo(arr[grt])>=0)
                grt=rgt;
        if( grt!=i){
                swap(arr,i,grt);
                heapify(arr, grt);
        }
}

static void sort( Comparable[] arr){
        total=arr.length-1;
        for(int i=total/2;i>=0;i--)
                heapify(arr, i);
        for(int i=total;i>0;i--){
                swap(arr,0,i);
                total--;
                heapify(arr, 0);
        }
}
public static void main(String[] args) {
         Integer[] arr={1,50,30,10,60,80};
         System.out.println("Before Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
         sort(arr);
         System.out.println("\nAfter Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
        }

}
```

Merge sort:

```
package com.sort;

public class MergeSort {
        int[] array;
        int[] TempArray;
        public static void main(String[] args) {
                    // TODO Auto-generated method stub
                    int[] arr={1,50,30,10,60,80};
                    System.out.println("Before sorting");
                    for(int i=0;i<arr.length;i++)
                           System.out.print(arr[i]+ "\t");

                    new MergeSort().PrepareForSort(arr);
                    System.out.println("\nAfter sorting");
                    for(int i=0;i< arr.length;i++)
                           System.out.print( arr[i]+ "\t");
             }
             void PrepareForSort(int[] arr){
                    // prepare for sort
                    this.array=arr;
                    this.TempArray=new int[arr.length];
                    doMergeSort(0,arr.length-1);
             }

             void doMergeSort(int LowerIndex, int HigherIndex){

                    if(LowerIndex< HigherIndex ){
                    int middle=LowerIndex+ (HigherIndex-LowerIndex)/2;
                    doMergeSort(LowerIndex,middle); //ex.(1-5)
                    doMergeSort(middle+1,HigherIndex);//ex.(6,10)
                    MergePart(LowerIndex,middle,HigherIndex);
                    }


             }

             // merge small problems and sort
             void MergePart(int LowerIndex,int middle,int HigherIndex ){
              for(int i=LowerIndex;i<=HigherIndex;i++)
                     TempArray[i]= array[i];
              int i=LowerIndex;
              int j=middle+1;
              int  k=LowerIndex;
              while(i<=middle && j<= HigherIndex){
                     if( TempArray[i]<= TempArray[j]){
                            array[k] =TempArray[i];
                            i++;
                      }else{
                            array[k] =TempArray[j];
                           j++;
                     }
                     k++;
              }
              while(i<=middle){
                     array[k] =TempArray[i];
                     k++;
                     i++;
              }
             }
}
```

Quick sorting:

```
package com.sort;

public class QuickSorting {

       static void QuickSort(int[] arr, int low, int high){

             if(low>high) return;
             int mid= low+(high-low)/2;
             int pivot= arr[mid];
             int i=low;
             int j=high;
             while(i<=j){
                    while(arr[i]<pivot)
                           i++;
                    while(arr[j]>pivot)
                           j--;
                    if(i<=j){
                           int temp= arr[i];
                           arr[i]=arr[j];
                           arr[j]=temp;
                           i++;
                           j--;
                    }

             }
             if(low<j)
                    QuickSort(arr, low, j);
             if(high>i)
                    QuickSort(arr, i, high);

       }



       public static void main(String[] args) {
        int[] arr={1,50,30,10,60,80};
        System.out.println("Before Sort");
        for(int i=0;i<arr.length;i++)
        System.out.print(arr[i] +"\t");
        QuickSort(arr, 0, arr.length-1);
        System.out.println("\nAfter Sort");
        for(int i=0;i<arr.length;i++)
        System.out.print(arr[i] +"\t");
       }
}
```

Selection sorting:

```
package com.sort;

public class SelectionSorting {
static void SelectionSort(int[] arr){
        for(int i=0;i<arr.length-1;i++){
                int index=i;
                for(int j=i+1;j<arr.length;j++){
                        if( arr[j]<arr[index])
                                index=j;
                }
                if(index!=i){
                        int temp= arr[index];
                        arr[index]=arr[i];
                        arr[i]=temp;
                }
        }
}

public static void main(String[] args) {
         int[] arr={1,50,30,10,60,80};
         System.out.println("Before Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
         SelectionSort(arr);
         System.out.println("\nAfter Sort");
         for(int i=0;i<arr.length;i++)
         System.out.print(arr[i] +"\t");
        }


}
```

BST:

```
package com.tree;

public class BST {
Node root;
        public BST() {
                root=null;
        }

        public Node NodeCreate(int value){
                return new Node(value, null, null);
        }
        public void add(Node start, Node newNode){
                if(root==null){
                        root=newNode;
                        return;
                }
        if(newNode.value> start.value){
            if( start.right==null)
                start.right=newNode;
            add(start.right,newNode);
        }
        if(newNode.value< start.value){
            if( start.left==null)
                start.left=newNode;
            add(start.left,newNode);
        }
    }

public void Search(int value, Node start){

    if(start==null){
        System.out.println("node isnot found");
        return;
    }
    if(start.value==value)
    {
        System.out.println("node is found");
        return;
    }
    if( value>start.value){
        Search(value, start.right);
    }
    if( value<start.value){
        Search(value, start.left);
    }
}

}
```

BST Demo:

```
package com.tree;

public class BSTDemo {

    public static void main(String[] args) {
        BST bt= new BST();

        bt.add(bt.root, bt.NodeCreate(10));
        bt.add(bt.root, bt.NodeCreate(12));
        bt.add(bt.root, bt.NodeCreate(11));
        bt.add(bt.root, bt.NodeCreate(13));
        bt.add(bt.root, bt.NodeCreate(6));
        bt.Search(111, bt.root);
    }

}
```

Node:

```
package com.tree;
```

```
public class Node {
int value;
Node left;
Node right;
       public Node(int value,Node left,
                    Node right) {
       this.value=value;
       this.left=left;
       this.right=right;
       }

}
```