```java
import java.util.ArrayList;
import java.util.Scanner;

public class GenericQueue<T> {
    private ArrayList<T> Queue;
    public GenericQueue(){
        Queue = new ArrayList<T>();
    }
    public void enqueue(T item){
        Queue.add(item);
    }
    public T dequeue(){
        if(isEmpty()){
            throw new RuntimeException("Queue Is Empty !!");
        }else{
            return Queue.remove(0);
        }
    }
    public void display(){
        if(isEmpty()) throw new RuntimeException("Queue is Empty!!");
        else{
            System.out.println("Queue Elements Are: ");
            for (T item:Queue) {
                System.out.print(item + " ");
            }
            System.out.println();
        }
    }
    public boolean isEmpty(){
        return Queue.isEmpty();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        GenericQueue<Integer> qui = new GenericQueue<>();
        for(int i=0;i<5;i++)    qui.enqueue(i);
        qui.display();
        System.out.println("1st popped ele:\n"+ qui.dequeue() +"\n2nd Popped ele:
\n"+qui.dequeue());
        qui.display();


        GenericQueue<String> qus = new GenericQueue<>();
        System.out.println("Enter 4 String Values: ");
        for (int i=0;i<4;i++)   qus.enqueue(sc.next());
        qus.display();
        System.out.println("1st popped ele:\n"+ qus.dequeue() +"\n2nd Popped ele:
\n"+qus.dequeue());
        qus.display();


    }
}




package ADSJ.src;


import java.util.LinkedList;


public class GenStack<T> {
    private LinkedList<T> newStack;


    public GenStack(){
        newStack = new LinkedList<>();
    }
```

```
public void push(T item){
    newStack.addFirst(item);
}
public T pop(){
    if(newStack.isEmpty())  throw new RuntimeException("Stack Is Empty!!");
    return newStack.removeFirst();
}

public void display(){
    if(newStack.isEmpty())  throw new RuntimeException("Stack Is Empty!!");
    for (T item: newStack){
        System.out.print(item+" ");
    }
    System.out.println();
}

public T peek(){
    if(newStack.isEmpty()) throw new RuntimeException("Stack Is Empty!!");
    return newStack.peek();
}



public static void main(String[] args) {
    GenStack<Integer> stack1 = new GenStack<>();

    for (int i = 0; i < 4; i++) {
        stack1.push(i);
    }
```

```
        stack1.display();

        System.out.println("1st popped ele:\n"+ stack1.pop() +"\n2nd Popped ele: \n"+stack1.pop());
        System.out.println("peek Element: "+stack1.peek());
        stack1.display();

    }
}



import java.util.LinkedList;
import java.util.Scanner;

public class GenLinkQueue <T>{
    private LinkedList<T> newQueue;
    public GenLinkQueue(){
        newQueue = new LinkedList<>();
    }
    public void enqueue(T item){
        newQueue.add(item);
    }
    public T dequeue(){
        if(newQueue.isEmpty())  throw new RuntimeException("Queue iS empty!!");
        return newQueue.removeFirst();
    }
    public void display(){
        if(newQueue.isEmpty()) throw new RuntimeException("Queue is empty !!");
        else{
            System.out.println("The Elements in the queue are: ");
            for(T item:newQueue){
                System.out.print(item+" ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Scanner sc =new Scanner(System.in);
        GenLinkQueue<Integer> queue1 = new GenLinkQueue<>();
        for (int i = 0; i < 5; i++) {
            queue1.enqueue(i);
        }
        queue1.display();
        System.out.println("1st popped ele:\n"+ queue1.dequeue() +"\n2nd Popped ele: \n"+
queue1.dequeue());
        queue1.display();

        GenLinkQueue<String> qus = new GenLinkQueue<>();
        System.out.println("Enter 4 String Values: ");
        for (int i=0;i<4;i++)   qus.enqueue(sc.next());
        qus.display();
        System.out.println("1st popped ele:\n"+ qus.dequeue() +"\n2nd Popped ele:
\n"+qus.dequeue());
        qus.display();
    }
}
```

```
import java.util.*;
public class IteratorDemo {
    public static void main(String[] args) {
        ArrayList<String> al = new ArrayList<String>();
        al.add("hello");
        al.add("hi");
        al.add("bye");
        Iterator<String> arrit = al.iterator();
        System.out.println("elements in array list is:");
        while(arrit.hasNext()){
            System.out.println(arrit.next());
        }
        LinkedList<String> li = new LinkedList<String>();
        li.add("cvr");
        li.add("college");
        li.add("engineering");
        Iterator<String> llit = li.iterator();
        System.out.println("elements in linked list is");
        while(llit.hasNext()){
            System.out.println(llit.next());
        }
    }
}



import java.util.Scanner;

public class LinearProbing {

    private int[] table;
    private int size;

    public LinearProbing(int size){
        this.size = size;
        table = new int[size];
        for(int i=0;i<size;i++) table[i] = -1;
```

```
}
public void insert(int key){
    int hash = key % size;
    int index = hash;

    while(table[index] != -1){
        index = (index+1) % size;
        if(index == hash){
            System.out.println("Hash Table is Full !!");
        }
    }
    table[index] = key;
    System.out.println("Inserted Key: "+key + "at index: "+index);
}

public int search(int key){
    int hash = key % size;
    int index = hash;
    while(table[index] != -1){
        if(table[index] == key){
            return index;
        }
        index = (index+1) % size;
        if(index == hash)   break;
    }
    return -1;
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int size = sc.nextInt();
```

```
        LinearProbing lp = new LinearProbing(size);

        while(true){
            System.out.println("Choose Any One Option:\n1.INSERT\n2.SEARCH\n3.EXIT:");
            int choice = sc.nextInt();
            switch (choice) {
                case 1 : {
                    System.out.println("Enter the Element You Wanted To insert: ");
                    int ele = sc.nextInt();
                    lp.insert(ele);
                    break;
                }
                case 2 : {
                    System.out.println("Enter The Element You Wanted To Search : ");
                    int searchElement = sc.nextInt();
                    int find = lp.search(searchElement);
                    if(find != -1){
                        System.out.println("Element is Found !!");
                    }else {
                        System.out.println("Element Is not Found !!");
                    }

                    break;
                }
                case 3 : System.exit(0);
                default : System.out.println("Invalid Choice !!");
            }
        }
    }
}
```

```
import java.util.LinkedList;
import java.util.Scanner;

class KeyValue<k,v>{
    private k key;
    private v value;

    public KeyValue(k key,v value){
        this.key = key;
        this.value = value;
    }
    public k getKey(){
        return key;
    }

    public v getValue() {
        return value;
    }
    public void setKey(k key) {
        this.key = key;
    }
    public void setValue(v value) {
        this.value = value;
    }
    public String toString(){
        return "("+key+","+value + ")";
    }
}

class CreateChainingTable<k,v>{
```

```
private LinkedList<KeyValue<k,v>> [] table;
private int size;
public CreateChainingTable(int size){
    table = new LinkedList[size];
    size = 0;


}


public int hashFunction(k key){
    return Math.abs(key.hashCode() % table.length);
}
public void insert(k key,v val){
    int hashVal = hashFunction(key);
    if(table[hashVal] == null){
        table[hashVal] = new LinkedList<>();
    }
    for(KeyValue<k,v> pair:table[hashVal]){
        if(pair.getKey().equals(key)){
            pair.setValue(val);
            return;
        }
    }
    table[hashVal].add(new KeyValue<>(key,val));
    size++;
}
public v search(k key){
    int hash = hashFunction(key);
    if(table[hash] != null){
        for(KeyValue<k,v> pair:table[hash]){
            if(pair.getKey().equals(key)){
                return pair.getValue();
                }
            }
        }
        return null;
    }
    public void delete(k key){
        int hash = hashFunction(key);
        if(table[hash]!=null){
            for(KeyValue<k,v> pair:table[hash]){
                if(pair.getKey().equals(key)){
                    table[hash].remove(pair);
                    size--;
                    return;
                }
            }
        }
    }
    public void display(){
        for(int i=0;i< table.length;i++ ){
            if(table[i]!=null){
                System.out.println("Index"+i+" ");
                for(KeyValue<k,v> pair:table[i]){
                    System.out.print(pair+"-->");
                }
                System.out.println();
            }
        }
    }

}
```

```
public class SeperateChaining {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Size: ");
        int size = sc.nextInt();
        CreateChainingTable<Integer,String> hashTable = new CreateChainingTable<>(size);
        while(true) {
            System.out.println("\n Seperate chaining ioperations \n");
            System.out.println("1.INSERT \n2.SEARCH\n3.DELETE\n4.DISPLAY\n5.EXIT: ");
            int choice = sc.nextInt();
            switch (choice) {
                case 1:
                    System.out.println("Enter the Key: ");
                    int key = sc.nextInt();
                    sc.nextLine();
                    System.out.println("Enter Value:");
                    String val = sc.nextLine();
                    hashTable.insert(key, val);
                    break;
                case 2:
                    System.out.println("Enter Key TO search: ");
                    int searchKey = sc.nextInt();
                    String searchVal = hashTable.search(searchKey);
                    if (searchVal != null) System.out.println("Value For Key: " + searchKey + " is --> " +
searchVal);
                    else System.out.println("Element Not Found !!");
                    break;
                case 3:
                    System.out.println("Enter Key To delete: ");
                    int deleteKey = sc.nextInt();
                    hashTable.delete(deleteKey);
                    break;
                case 4:
                    System.out.println("The Following Elements are: ");
                    hashTable.display();
                    break;
                case 5:
                    System.exit(0);

            }
        }
    }
}


import java.util.LinkedList;
import java.util.Scanner;

class LinkedListClass{
    private LinkedList<Integer> sortedList;
    public LinkedListClass(){
        sortedList = new LinkedList<>();
    }
    public void insert(int item){
        if(sortedList.isEmpty()) sortedList.addFirst(item);
        else{
            int i=0;
            while(i < sortedList.size() && item >= sortedList.get(i) ) i++;
            sortedList.add(i,item);
        }
    }
    public void remove(int item){
        if(sortedList.isEmpty()){
            System.out.println("No Item To remove!! Linear List is Empty");
        }else{
            sortedList.remove(item);
            System.out.println("-------Element is deleted-----");
        }
    }
    public void display(){
        System.out.println("The Sorted Chain Elements Are: ");
        for(int i : sortedList){
            System.out.print(i+" ");
        }
    }
}


public class SortedChain {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinkedListClass sortedChain = new LinkedListClass();
        while(true){
            System.out.println("Enter The choices:\n1.INSERT\n2.REMOVE\n3.DISPLAY:");
            int choice = sc.nextInt();
            switch (choice){
                case 1:
                    System.out.println("Enter The Element You wanted To insert: ");
                    int el = sc.nextInt();
                    sortedChain.insert(el);
                    break;
                case 2:
                    System.out.println("Enter the element you wanted to delete: ");
                    int del = sc.nextInt();
                    sortedChain.remove(del);
                    break;
                case 3:
                    sortedChain.display();
                    break;
                default:return;
            }
        }
    }
}
```

```
import java.util.*;
public class KMPAlgorithm {
    private static int[] LPSArray(String pattern) {
        int[] lps = new int[pattern.length()];
        int i = 1, j = 0;
        while (i < pattern.length()) {
            if (pattern.charAt(i) == pattern.charAt(j)) {
                ps[i] = j + 1;
                i++;
                j++;
            } else {
                if (j != 0) {
                    j = lps[j - 1];
                } else {
```

```
                lps[i] = 0;
                i++;
            } // inner else closing
        } // outer else closing
    } // while closing
    return lps;
}// LPSArray closing

public static void KMPSearch(String text, String pattern) {
    int[] lps = LPSArray(pattern);
    int i = 0, j = 0;
    while (i < text.length()) {
        if (pattern.charAt(j) == text.charAt(i)) {
            i++;
            j++;
            if (j == pattern.length()) {
                System.out.println("Pattern found at index " + (i - j));
                j = lps[j - 1];
            }
        } else {
            if (j != 0) {
                j = lps[j - 1];
            } else {
                i++;
            }
        }
    }
}

public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
```

```
        System.out.println("enter Text:");
        String text = s.nextLine();
        System.out.println("enter Pattern:");
        String pattern = s.nextLine();
        KMPSearch(text, pattern);
    }
}
```

```
import java.util.Scanner;

class LinearListClass<T>{
    T[] list;
    int size = 0;
    public LinearListClass(int intialSize){
        list = (T[]) new Object[intialSize];
    }
    public LinearListClass(){
        list = (T[]) new Object[10];
    }

    public void insert(T item){
        if(size == list.length) extend();
        list[size++] = item;
    }
    private void extend(){
        int extendSize = list.length * 3/2;
        T[] temp = (T[]) new Object[extendSize];
        for(int i=0;i< list.length;i++){
            temp[i] = list[i];
        }
        this.list = temp;
    }
    public void remove(T item){
        for(int i=0;i< list.length;i++){
            if(list[i]==item){
                for(;i< list.length-1;i++){
                    list[i] = list[i+1];
                }
                size--;
                System.out.println("Element Is deleted!!");
            }
        }
    }
    public void display(){
        System.out.println("The elements are: ");
        for (T item:list) {
            System.out.print(item + "-->");
        }
    }
}


public class LinearList {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        LinearListClass<Integer> linearList = new LinearListClass<>(5);
        while(true){
            System.out.println("Enter The choices:\n1.INSERT\n2.REMOVE\n3.DISPLAY:");
            int choice = sc.nextInt();
            switch(choice){
                case 1:
```

```
                System.out.println("Enter The Element You wanted To insert: ");
                int el = sc.nextInt();
                linearList.insert(el);
                break;
              case 2:
                System.out.println("Enter the element you wanted to delete: ");
                int del = sc.nextInt();
                linearList.remove(del);
                break;

              case 3:
                linearList.display();
                break;
              default:return;
            }

        }
    }
}



import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Scanner;

class Person{
    private final String name;
    private final float income;
    private final int age;
    public Person(String name,float income,int age){
        this.name = name;
        this.income = income;
        this.age = age;
    }

    public int getAge() {
        return age;
    }

    public float getIncome() {
        return income;
    }

    public String getName() {
        return name;
    }
}
public class SetOperation {
    public static void main(String[] args) {

        HashSet<Person> A = new HashSet<>();
        Scanner sc = new Scanner(System.in);
        for (int i=0;i<5;i++){
            System.out.println("Enter Name: ");
            String name = sc.next();
            System.out.println("Enter Income: ");
            float income = sc.nextFloat();
            System.out.println("Enter age: ");
            int age = sc.nextInt();
```

```
         Person p = new Person(name,income,age);
         A.add(p);
      }

      for (Person item: A) {
         System.out.println(item.getName());
      }




//       Person p1 = new Person("SaiRam",50000,65);
//       Person p2 = new Person("SanDeep",10000,59);
//       Person p3  = new Person("Sanjay",9000,79);
//       Person p4  = new Person("SriDhar",7000,68);
//       Person p5 = new Person("SaiNihal",4000,70);
//       Person p6 = new Person("Ram",9500,73);
//       Person p7 = new Person("Deepak",9200,62);
//       Person p8 = new Person("Hafeez",9700,67);
//       Person p9 = new Person("Muneeb",9999,99);
//       Person p10 = new Person("Dheeraj",11000,89);
//
//
//       HashSet<Person> A = new HashSet<>();
//
//       A.add(p1);A.add(p6);
//       A.add(p2);A.add(p7);
//       A.add(p3);A.add(p8);
//       A.add(p4);A.add(p9);
//       A.add(p5);A.add(p10);
//
```

```
//
//

        HashSet<Person> B = new HashSet<>();
        HashSet<Person> C = new HashSet<>();



        for (Person item: A) {
//            System.out.println(item.getName());
            if(item.getAge() >60){
                B.add(item);
            }
            if(item.getIncome()<10000.0){
                C.add(item);
            }
        }

        System.out.println("\nThe Persons Whose Age is greater Than 60: \n");
        System.out.print("\n\t NAME \t\t INCOME \t\t AGE \n");
        for (Person i: B) {
            System.out.printf("\t%5s\t\t%.2f\t\t%3d\n",i.getName(),i.getIncome(),i.getAge());
        }

        System.out.println("\nThe Persons Whose income is less Than 10000: \n");
        System.out.print("\n\t NAME \t\t INCOME \t\t AGE \n");
        for (Person i: C) {
            System.out.printf("\t%5s\t\t%.2f\t\t%3d\n",i.getName(),i.getIncome(),i.getAge());
        }

        HashSet<Person> intersection = new HashSet<>(B);
        intersection.retainAll(C);


        System.out.println("\nThe InterSection Of B and C is: \n");
        System.out.println("\n\tNAME\t\tINCOME\t\tAGE");
        for (Person i: intersection) {
            System.out.printf("\t%4s\t\t%4.2f\t\t%3d\n",i.getName(),i.getIncome(),i.getAge());
        }

    }
}



import java.util.Scanner;


class ArrayStack<T>{
    private T[] items;
    private int top;


    public ArrayStack(int size){
        items = (T[]) new Object[size];
        top = -1;


    }


    public void push(T data){
        if (top==items.length-1) System.out.println("Stack is Full !!");
        else items[++top] = data;
    }


    public T pop(){
        if(isEmpty()){
            System.out.println("Stack IS Empty");
            return null;
        }
        else return items[top--];

    }

    public void display(){
        if(isEmpty()){
            System.out.println("Stack IS Empty");
        }
        else{
            for(int i=top;i>=0;i--) System.out.println(items[i]);
        }
    }

    public T peek(){
        if(isEmpty()){
            System.out.println("Stack IS Empty");
            return null;
        }
        else    return items[top];

    }
    public boolean isEmpty(){
        if(top==-1) return true;
        else return false;
    }
}

class LinkedStack<T>{
```

```
private Node<T> top;
private int size;
private static class Node<T> {
    private T data;
    private Node<T> next;
    public Node(T data){
        this.data = data;
        this.next = null;
    }
}

public LinkedStack(){
    top = null;
    size = 0;
}
public void push(T item){
    Node <T> node = new Node<>(item);
    node.next = top;
    top = node;
    size++;
}
public T pop(){
    if(isEmpty()){
        System.out.println("Stack Is Empty!!");
        return null;
    }else{
        T item = top.data;
        top = top.next;
        size--;
        return item;
    }
```

```
   }
   public void display(){
      if(isEmpty()){
         System.out.println("Stack Is Empty!!");
      }else {
         Node<T> current = top;
         while(current!=null){
            System.out.println(current.data);
            current = current.next;
         }
      }
   }
   public T peek(){
      if(isEmpty()){
         System.out.println("Stack Is Empty!!");
         return null;
      }else{
         return top.data;
      }
   }
   public boolean isEmpty(){
      if(size==0) return true;
      else return false;
   }
}
```

```
public class STacks {
   public static void main(String[] args) {
      int chooseType;
```

```
    Scanner sc = new Scanner(System.in);
    System.out.println("Choose The DataType You wanted To
insert:\n1.Integer\n2.Double\n3.String");
    chooseType = sc.nextInt();
    switch (chooseType){
      case 1:{
        ArrayStack<Integer> asi= new ArrayStack<>(25);
        int data ;
        System.out.println("Enter the elements you wanted to insert: ");
        int n = sc.nextInt();
        for(int i=0;i<n;i++){
          data = sc.nextInt();
          asi.push(data);
        }
        operations(asi);
        break;

      }
      case 2:{
        ArrayStack<Double> asd= new ArrayStack<>(25);
        Double data ;
        System.out.println("Enter the elements you wanted to insert: ");
        int n = sc.nextInt();
        for(int i=0;i<n;i++){
          data = sc.nextDouble();
          asd.push(data);
        }
        operations(asd);
        break;
      }
      case 3:{
```

```
                ArrayStack<String> ass= new ArrayStack<>(25);
                String data ;
                System.out.println("Enter the elements you wanted to insert: ");
                int n = sc.nextInt();
                for(int i=0;i<n;i++){
                    data = sc.next();
                    ass.push(data);
                }
                operations(ass);
                break;
            }
        }
    }
    public static <T> void operations(ArrayStack<T> o){
        int chooseOp;

        while(true) {
            System.out.println("Choose The Operations You wanted to be 
performed:\n1.pop\n2.peek\n3.display:");
            Scanner sc = new Scanner(System.in);
            chooseOp = sc.nextInt();
            switch (chooseOp) {
                case 1: {
                    T item = o.pop();
                    System.out.println(item);
                    break;
                }
                case 2: {
                    T top = o.peek();
                    System.out.println(top);
                    break;
                }
                case 3: {
                    o.display();
                    break;
                }
                default: return;
            }
        }
    }
}
```

BST

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

class BST{
    Node root;

    public BST(){
        root = null;
    }
    public void insert(int data){
        root = insertNode(root,data);
    }
    private Node insertNode(Node node,int data){
```

```
    if(node == null){
        node = new Node(data);
        return node;
    }
    if(data < node.data){
        node.left = insertNode(node.left,data);
    } else if (data > node.data) {
        node.right = insertNode(node.right,data);
    }
    return node;
}
public void search(int data){
    Node check = toFind(root,data);
    if(check == null){
        System.out.println("Element Not Found");
    } else {
        System.out.println(data + "Element Is Found!!");
    }
}

private Node toFind(Node node,int data){
    if(node == null || node.data == data){
        return node;
    }
    if(data < node.data){
        return toFind(node.left,data);
    } else {
        return toFind(node.right,data);
    }


}
```

```
public void remove(int data){
    root = toDelete(root,data);
}

private Node toDelete(Node node,int data){
    if(node == null){
        return null;
    } else if (data < node.data) {
        node.left = toDelete(node.left,data);
    } else if (data > node.data) {
        node.right = toDelete(node.right,data);
    } else {
        if(node.left == null){
            return node.right;
        } else if (node.right == null) {
            return node.left;
        }
        Node replaceNode = minFromRight(node.right);
        node.data = replaceNode.data;
        node.right = toDelete(node.right , replaceNode.data);
    }
    return node;
}


private Node minFromRight(Node node){
    Node temp = node;
    while(temp.left != null){
        temp = temp.left;
    }
    return temp;
```

```
}


public void inOrder(){
   System.out.println("\nInOrder\n");
   toFindInorder(root);
}
private void toFindInorder(Node node){
   if(node != null){
      toFindInorder(node.left);
      System.out.print(node.data + "-->");
      toFindInorder(node.right);
   }
}
public void preOrder(){
   System.out.println("\nPreOrder\n");
   toFindPreOrder(root);
}
private void toFindPreOrder(Node node){
   if(node != null){
      System.out.print(node.data + "-->");
      toFindPreOrder(node.left);
      toFindPreOrder(node.right);
   }
}
public void postOrder(){
   System.out.println("\nPostOrder\n");
   toFindPostOrder(root);
}

private void toFindPostOrder(Node node){
   if(node != null) {
```

```
            toFindPostOrder(node.left);
            toFindPostOrder(node.right);
            System.out.print(node.data + "-->");
        }
    }
}


public class BinarySearchTree {
    public static void main(String[] args) {
        BST tree = new BST();
        tree.insert(10);
        tree.insert(9);
        tree.insert(1);
        tree.insert(3);
        tree.insert(13);
        tree.insert(15);
        tree.insert(11);
        tree.insert(12);
        tree.inOrder();
        tree.postOrder();
        tree.preOrder();
        tree.remove(13);
        tree.inOrder();
        System.out.println();
        tree.search(11);

    }
}
```