1.Write a java program to demonstrate use of bounded type parameters and wild card arguments.

CODE:

```
import java.util.*;
public class BoundAndWild1
{
    public static <T extends Number> double Sum(List<T> number)
    {
        double sum = 0.0;
        for (T num : number)
            sum += num.doubleValue();
        return sum;
    }
    public static void printlist(List<?> list)
   {
        for (Object ele : list)
            System.out.println(ele + " ");
        System.out.println();
    }
    public static void main(String[] args)
   {
        ArrayList<Integer> l = new ArrayList<>();
        l.add(10);
        l.add(20);
        l.add(30);
        System.out.println(Sum(l));
        ArrayList<Double> d = new ArrayList<>();
        d.add(10.2);
        d.add(11.2);
        System.out.println(Sum(d));
        ArrayList<String> s = new ArrayList<>();
        s.add("HI");
        s.add("Hey");
        printlist(s);
    }
}
```

2.Write a java program to implement iterators on ArrayList and LinkedList.

CODE:

```
import java.util.*;
public class IteratorDemo2
{
    public static void main(String[] args)
    {
        ArrayList<String> s = new ArrayList<>();
        s.add("Hello");
        s.add("hi");
        Iterator<String> a = s.iterator();
        while (a.hasNext())
        {
            System.out.println(a.next());
        }
        LinkedList<String> ll = new LinkedList<>();
        ll.add("Hey");
        ll.add("there");
        Iterator<String> l = ll.iterator();
        while (l.hasNext())
        {
            System.out.println(l.next());
        }
    }
```

```
}
```

3.a)Implement a Generic stack to deal with Integer, Double and String data using user defined arrays and linked lists.

CODE:

```
import java.util.*;
class ArrayStack<T> {
    private T[] items;
    private int top;

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

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

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

    public T peek() {
        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 {
            int i;
            for (i = top; i >= 0; i--)
                System.out.println(items[i]);
        }
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public int size() {
        return top + 1;
    }
}

class LinkedStack<T> {
    private Node<T> top;
    private int size;

    public static class Node<T> {
```

```
        private T data;
        public Node<T> next;

        public Node(T data) {
            this.data = data;
            this.next = null;
        }
    }

    public LinkedStack() {
        top = null;
        size = 0;
    }

    public void push(T ele) {
        Node<T> n = new Node<T>(ele);
        n.next = top;
        top = n;
        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 T peek() {
        if (isEmpty()) {
            System.out.println("Stack is Empty..");
            return null;
        } else {
            return top.data;
        }
    }

    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 boolean isEmpty() {

        return size == 0;
    }

    public int size() {
        return size;
    }

}
```

```
public class GenStack3a {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter choice:");
        System.out.println("1.ArrayList\n2.LinkedList\n3.Exit");
        int ch = sc.nextInt();
        switch (ch) {
            case 1:
                System.out.println("Choose the data type on which you want to
perform operation:");
                System.out.println("1.Integer\n2.Float\n3.String");
                int datatype = sc.nextInt();
                switch (datatype) {
                    case 1:

                        ArrayStack<Integer> a = new ArrayStack<Integer>(10);
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch1 = sc.nextInt();
                            switch (ch1) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        a.push(sc.nextInt());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
a.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
a.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    a.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + a.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 2:

                        ArrayStack<Float> f = new ArrayStack<Float>(10);

                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch2 = sc.nextInt();

                            switch (ch2) {
                                case 1:
```

```
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        f.push(sc.nextFloat());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
f.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
f.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    f.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + f.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 3:
                        ArrayStack<String> s = new ArrayStack<String>(10);
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch3 = sc.nextInt();
                            switch (ch3) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        s.push(sc.next());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
s.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
s.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    s.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + s.size());
                                default:
                                    return;
                            }

                        }
                }
            case 2:
```

```
                System.out.println("Choose the data type on which you want to
perform operation:");
                System.out.println("1.Integer\n2.Float\n3.String");
                int datatypel = sc.nextInt();
                switch (datatypel) {
                    case 1:

                        LinkedStack<Integer> l = new LinkedStack<Integer>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch1 = sc.nextInt();
                            switch (ch1) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        l.push(sc.nextInt());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
l.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
l.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    l.display();

                                    break;
                                case 5:
                                    System.out.println("Size:" + l.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 2:
                        LinkedStack<Float> f = new LinkedStack<Float>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch2 = sc.nextInt();
                            switch (ch2) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        f.push(sc.nextFloat());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
f.pop());
                                    break;
                                case 3:
```

```
                                    System.out.println("Peeked element:" +
f.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    f.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + f.size());
                                default:

                                    return;

                            }
                        }
                    case 3:
                        LinkedStack<String> s = new LinkedStack<String>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch3 = sc.nextInt();

                            switch (ch3) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        s.push(sc.next());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
s.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
s.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    s.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + s.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                }
            default:
                break;
        }
    }
}
```

3.b)Implement a Generic queue to deal with Integer, Double and String data using user defined arrays and linked lists.

CODE:

```
import java.util.*;
class ArrayQ<T> {
    private T[] items;
    private int rear, front;
    int size;

    public ArrayQ(int size) {
        items = (T[]) new Object[size];
        front = 0;
        rear = -1;
    }

    public void push(T item) {
        if (rear == items.length - 1) {
            System.out.println("Queue is Full..");
        } else {
            ++rear;
            items[rear] = item;
            size++;
        }
    }

    public T pop() {
        if (isEmpty()) {
            System.out.println("Queue is Empty..");
            return null;
        } else {
            T temp = items[front];
            front++;
            size--;
            return temp;
        }
    }

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

    public void display() {
        if (isEmpty()) {
            System.out.println("Queue is Empty..");
        } else {
            for (int i = front; i <= rear; i++)
                System.out.println(items[i]);
        }
    }

    public boolean isEmpty() {
        return size() == 0;
    }

    public int size() {
        return rear + 1;
    }
}

class LinkedQ<T> {
    private Node<T> front, rear, newnode;
```

```
private int size;

public static class Node<T> {
    private T data;
    public Node<T> next;

    public Node(T data) {
        this.data = data;
        this.next = null;
    }
}

public LinkedQ() {
    front = null;
    rear = null;
    size = 0;
}

public void push(T ele) {
    Node<T> n = new Node<T>(ele);
    if (isEmpty()) {
        front = rear = newnode;
    } else {
        rear.next = newnode;
        rear = newnode;
        size++;
    }
}

public T pop() {
    if (isEmpty()) {
        System.out.println("Queue is Empty..");
        return null;
    } else {
        T temp = front.data;
        front = front.next;
        size--;
        return temp;
    }
}

public T peek() {
    if (isEmpty()) {
        System.out.println("Stack is Empty..");
        return null;
    } else {
        return front.data;
    }
}

public void display() {
    if (isEmpty())
        System.out.println("Queue is Empty..");
    else {
        LinkedQ.Node<T> current;
        current = front;
        while (current != null) {
            System.out.println(current.data);
            current = current.next;
        }
    }
}

public boolean isEmpty() {
```

```
        return size == 0;
    }

    public int size() {
        return size;
    }
}

public class GenQueue3b {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter choice:");
        System.out.println("1.ArrayList\n2.LinkedList\n3.Exit");
        int ch = sc.nextInt();
        switch (ch) {
            case 1:
                System.out.println("Choose the data type on which you want to
perform operation:");
                System.out.println("1.Integer\n2.Float\n3.String");
                int datatype = sc.nextInt();
                switch (datatype) {
                    case 1:

                        ArrayQ<Integer> a = new ArrayQ<Integer>(10);
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch1 = sc.nextInt();
                            switch (ch1) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        a.push(sc.nextInt());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
a.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
a.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    a.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + a.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 2:

                        ArrayQ<Float> f = new ArrayQ<Float>(10);

                        while (true) {
```

```
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch2 = sc.nextInt();

                            switch (ch2) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        f.push(sc.nextFloat());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
f.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
f.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    f.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + f.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 3:
                        ArrayQ<String> s = new ArrayQ<String>(10);
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch3 = sc.nextInt();
                            switch (ch3) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        s.push(sc.next());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
s.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
s.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    s.display();
                                    break;
                                case 5:
```

```
                                System.out.println("Size:" + s.size());
                            default:
                                return;
                        }

                    }
                }
            case 2:
                System.out.println("Choose the data type on which you want to
perform operation:");
                System.out.println("1.Integer\n2.Float\n3.String");
                int datatypel = sc.nextInt();
                switch (datatypel) {
                    case 1:

                        LinkedQ<Integer> l = new LinkedQ<Integer>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch1 = sc.nextInt();
                            switch (ch1) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        l.push(sc.nextInt());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
l.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
l.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    l.display();

                                    break;
                                case 5:
                                    System.out.println("Size:" + l.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                    case 2:
                        LinkedQ<Float> f = new LinkedQ<Float>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch2 = sc.nextInt();
                            switch (ch2) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
```

```
                                    for (int i = 0; i < n; i++)
                                        f.push(sc.nextFloat());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
f.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
f.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    f.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + f.size());
                                default:

                                    return;

                            }
                        }
                    case 3:
                        LinkedQ<String> s = new LinkedQ<String>();
                        while (true) {
                            System.out.println("Enter Operation to perform on
stack:");
                            System.out.println("1.Push\n2.Pop\n3.Peek\
n4.Display\n5.Size");
                            int ch3 = sc.nextInt();

                            switch (ch3) {
                                case 1:
                                    System.out.println("Enter no:of elements to
push into a stack:");
                                    int n = sc.nextInt();
                                    for (int i = 0; i < n; i++)
                                        s.push(sc.next());
                                    break;
                                case 2:
                                    System.out.println("Popped Element:" +
s.pop());
                                    break;
                                case 3:
                                    System.out.println("Peeked element:" +
s.peek());
                                    break;
                                case 4:
                                    System.out.println("Elements in stack
are:");
                                    s.display();
                                    break;
                                case 5:
                                    System.out.println("Size:" + s.size());
                                    break;
                                default:
                                    return;
                            }
                        }
                }
            default:
                break;
```

```
        }
    }
}
```

4.a)Write a java program to implement Generic stack using ArrayList collection class.

CODE:

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

class ArrGen<T> {
    private ArrayList<T> stack;

    public ArrGen() {
        stack = new ArrayList<T>();
    }

    public void push(T item) {
        stack.add(item);
    }

    public T pop() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return stack.remove(stack.size() - 1);
        }
    }

    public T peek() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return stack.get(stack.size() - 1);
        }
    }

    public void display() {
        if (isEmpty())
            throw new RuntimeException("Empty..");
        else {
            for (int i = stack.size() - 1; i >= 0; i--)
                System.out.println(stack.get(i));
        }
    }

    public boolean isEmpty() {

        return stack.size() == 0;
    }

    public int size() {
        return stack.size();
    }
}

public class CollectionDemo4a {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Choose the data type on which you want to perform
```

```
operation:");
        System.out.println("1.Integer\n2.Float\n3.String");
        int datatype = sc.nextInt();
        switch (datatype) {
            case 1:
                ArrayStack<Integer> a = new ArrayStack<Integer>(10);
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch1 = sc.nextInt();
                    switch (ch1) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                a.push(sc.nextInt());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + a.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + a.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            a.display();
                            break;
                        case 5:
                            System.out.println("Size:" + a.size());
                            break;
                        default:
                            return;
                    }
                }
            case 2:

                ArrayStack<Float> f = new ArrayStack<Float>(10);

                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch2 = sc.nextInt();

                    switch (ch2) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                f.push(sc.nextFloat());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + f.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + f.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            f.display();
                            break;
```

```
                        case 5:
                            System.out.println("Size:" + f.size());
                            break;
                        default:
                            return;
                    }
                }
            case 3:
                ArrayStack<String> s = new ArrayStack<String>(10);
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch3 = sc.nextInt();
                    switch (ch3) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                s.push(sc.next());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + s.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + s.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            s.display();
                            break;
                        case 5:
                            System.out.println("Size:" + s.size());
                        default:
                            return;
                    }

                }
        }
    }
}
```

4.b)Write a java program to implement Generic stack using LinkedList collection class.

CODE:

```
import java.util.*;

class LinkedGen<T> {
    private LinkedList<T> stack;

    public LinkedGen() {
        stack = new LinkedList<T>();
    }

    public void push(T item) {
        stack.addFirst(item);
    }

    public T pop() {
        if (isEmpty()) {
            System.out.println("Empty..");
```

```
            return null;
        } else {
            return stack.removeFirst();
        }
    }

    public T peek() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return stack.getFirst();
        }
    }

    public void display() {
        if (isEmpty())
            throw new RuntimeException("Empty..");
        else {
            for (int i = stack.size() - 1; i >= 0; i--)
                System.out.println(stack.get(i));
        }
    }

    public boolean isEmpty() {

        return stack.size() == 0;
    }

    public int size() {
        return stack.size();
    }
}

public class CollectionDemo4b {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter choice:");
        System.out.println("Choose the data type on which you want to perform
operation:");
        System.out.println("1.Integer\n2.Float\n3.String");
        int datatypel = sc.nextInt();
        switch (datatypel) {
            case 1:

                LinkedStack<Integer> l = new LinkedStack<Integer>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch1 = sc.nextInt();
                    switch (ch1) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                l.push(sc.nextInt());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + l.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + l.peek());
```

```
                        break;
                    case 4:
                        System.out.println("Elements in stack are:");
                        l.display();

                        break;
                    case 5:
                        System.out.println("Size:" + l.size());
                        break;
                    default:
                        return;
                }
            }
        case 2:
            LinkedStack<Float> f = new LinkedStack<Float>();
            while (true) {
                System.out.println("Enter Operation to perform on stack:");
                System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                int ch2 = sc.nextInt();
                switch (ch2) {
                    case 1:
                        System.out.println("Enter no:of elements to push
into a stack:");
                        int n = sc.nextInt();
                        for (int i = 0; i < n; i++)
                            f.push(sc.nextFloat());
                        break;
                    case 2:
                        System.out.println("Popped Element:" + f.pop());
                        break;
                    case 3:
                        System.out.println("Peeked element:" + f.peek());
                        break;
                    case 4:
                        System.out.println("Elements in stack are:");
                        f.display();
                        break;
                    case 5:
                        System.out.println("Size:" + f.size());
                    default:

                        return;

                }
            }
        case 3:
            LinkedStack<String> s = new LinkedStack<String>();
            while (true) {
                System.out.println("Enter Operation to perform on stack:");
                System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                int ch3 = sc.nextInt();

                switch (ch3) {
                    case 1:
                        System.out.println("Enter no:of elements to push
into a stack:");
                        int n = sc.nextInt();
                        for (int i = 0; i < n; i++)
                            s.push(sc.next());
                        break;
                    case 2:
                        System.out.println("Popped Element:" + s.pop());
```

```
                        break;
                    case 3:
                        System.out.println("Peeked element:" + s.peek());
                        break;
                    case 4:
                        System.out.println("Elements in stack are:");
                        s.display();
                        break;
                    case 5:
                        System.out.println("Size:" + s.size());
                        break;
                    default:
                        return;
                }
            }
        }
    }
}
```

5.a)Write a java program to implement Generic queue using ArrayList collection class.

CODE:

```
import java.util.*;
class ArrayQ<T> {
    private ArrayList<T> queue;

    public ArrayQ() {
        queue = new ArrayList<T>();
    }

    public void push(T item) {
        queue.add(item);
    }

    public T pop() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return queue.remove(0);
        }
    }

    public T peek() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return queue.get(0);
        }
    }

    public void display() {
        if (isEmpty())
            throw new RuntimeException("Empty..");
        else {
            for (int i = 0; i < queue.size(); i++)
                System.out.println(queue.get(i));
        }
    }

    public boolean isEmpty() {
```

```
        return queue.size() == 0;
    }

    public int size() {
        return queue.size();
    }

}

public class CollectionQA5a {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Choose the data type on which you want to perform
operation:");
        System.out.println("1.Integer\n2.Float\n3.String");
        int datatype = sc.nextInt();
        switch (datatype) {
            case 1:
                ArrayQ<Integer> a = new ArrayQ<Integer>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch1 = sc.nextInt();
                    switch (ch1) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                a.push(sc.nextInt());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + a.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + a.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            a.display();
                            break;
                        case 5:
                            System.out.println("Size:" + a.size());
                            break;
                        default:
                            return;
                    }
                }
            case 2:

                ArrayQ<Float> f = new ArrayQ<Float>();

                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch2 = sc.nextInt();

                    switch (ch2) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
```

```
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                f.push(sc.nextFloat());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + f.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + f.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            f.display();
                            break;
                        case 5:
                            System.out.println("Size:" + f.size());
                            break;
                        default:
                            return;
                    }
                }
            case 3:
                ArrayQ<String> s = new ArrayQ<String>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch3 = sc.nextInt();
                    switch (ch3) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                s.push(sc.next());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + s.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + s.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            s.display();
                            break;
                        case 5:
                            System.out.println("Size:" + s.size());
                        default:
                            return;
                    }

                }
        }
    }
}
```

5.b)Write a java program to implement Generic queue using LinkedList collection class.

CODE:

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

```
class LinkedQ<T> {
    private LinkedList<T> q;

    public LinkedQ() {
        q = new LinkedList<T>();
    }

    public void push(T item) {
        q.addFirst(item);
    }

    public T pop() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return q.removeLast();
        }
    }

    public T peek() {
        if (isEmpty()) {
            System.out.println("Empty..");
            return null;
        } else {
            return q.getLast();
        }
    }

    public void display() {
        if (isEmpty())
            throw new RuntimeException("Empty..");
        else {
            for (int i = q.size() - 1; i >= 0; i--)
                System.out.println(q.get(i));
        }
    }

    public boolean isEmpty() {

        return q.size() == 0;
    }

    public int size() {
        return q.size();
    }
}

public class CollectionQL5b {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter choice:");
        System.out.println("Choose the data type on which you want to perform
operation:");
        System.out.println("1.Integer\n2.Float\n3.String");
        int datatypel = sc.nextInt();
        switch (datatypel) {
            case 1:

                LinkedQ<Integer> l = new LinkedQ<Integer>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
```

```
                    int ch1 = sc.nextInt();
                    switch (ch1) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                l.push(sc.nextInt());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + l.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + l.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            l.display();

                            break;
                        case 5:
                            System.out.println("Size:" + l.size());
                            break;
                        default:
                            return;
                    }
                }
            case 2:
                LinkedQ<Float> f = new LinkedQ<Float>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch2 = sc.nextInt();
                    switch (ch2) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                f.push(sc.nextFloat());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + f.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + f.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            f.display();
                            break;
                        case 5:
                            System.out.println("Size:" + f.size());
                        default:

                            return;

                    }
                }
            case 3:
                LinkedQ<String> s = new LinkedQ<String>();
                while (true) {
                    System.out.println("Enter Operation to perform on stack:");
```

```
                    System.out.println("1.Push\n2.Pop\n3.Peek\n4.Display\
n5.Size");
                    int ch3 = sc.nextInt();

                    switch (ch3) {
                        case 1:
                            System.out.println("Enter no:of elements to push
into a stack:");
                            int n = sc.nextInt();
                            for (int i = 0; i < n; i++)
                                s.push(sc.next());
                            break;
                        case 2:
                            System.out.println("Popped Element:" + s.pop());
                            break;
                        case 3:
                            System.out.println("Peeked element:" + s.peek());
                            break;
                        case 4:
                            System.out.println("Elements in stack are:");
                            s.display();
                            break;
                        case 5:
                            System.out.println("Size:" + s.size());
                            break;
                        default:
                            return;
                    }
                }
        }
    }
}
```

6.Write a java program to demonstrate the use of following collection classes

a)HashSet

CODE:

```
import java.util.*;
public class Hash6a {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<Integer>();
        set.add(1);
        set.add(2);
        set.add(0);
        set.add(0);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        set.remove(1);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        System.out.println("Size:" + set.size());
        Iterator<Integer> h = set.iterator();
        while (h.hasNext())
            System.out.println(h.next());
        System.out.println("Empty or not:" + set.isEmpty());
        set.clear();
        System.out.println("Empty or not:" + set.isEmpty());
    }
}
```

b)LinkedHashSet

CODE:

```
import java.util.*;
public class LinkedHash6b {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<Integer>();
        set.add(1);
        set.add(2);
        set.add(0);
        set.add(0);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        set.remove(1);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        System.out.println("Size:" + set.size());
        Iterator<Integer> h = set.iterator();
        while (h.hasNext())
            System.out.println(h.next());
        System.out.println("Empty or not:" + set.isEmpty());
        set.clear();
        System.out.println("Empty or not:" + set.isEmpty());
    }
}
```

c)TreeSet

CODE:

```
import java.util.*;
public class Tree6c {
    public static void main(String[] args) {
        HashSet<Integer> set = new HashSet<Integer>();
        set.add(1);
        set.add(2);
        set.add(0);
        set.add(0);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        set.remove(1);
        System.out.println("Checking 1 is present in set:" + set.contains(1));
        System.out.println("Elements in set:" + set);
        System.out.println("Size:" + set.size());
        Iterator<Integer> h = set.iterator();
        while (h.hasNext())
            System.out.println(h.next());
        System.out.println("Empty or not:" + set.isEmpty());
        set.clear();
        System.out.println("Empty or not:" + set.isEmpty());
    }
}
```

7.Write a java program to create a class called Person with income, age and name as its members.

CODE:

```
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);
        System.out.println("Enter no:of Person:");
        int n = sc.nextInt();
        for (int i = 0; i < n; 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());
        }

    }
}
```

8.Write a java program to demonstrate the use of following collection classes.

a)HashMap

CODE:

```
import java.util.*;
public class HashM8a {
    public static void main(String[] args) {
        HashMap<Integer, String> hm = new HashMap<Integer, String>();
        System.out.println("Is Empty or not:" + hm.isEmpty());
        hm.put(1, "A");
        hm.put(2, "B");
        hm.put(3, "C");
        System.out.println(hm);
        hm.remove(1, "D");
        System.out.println("Elements :" + hm);
        hm.replace(1, "D");
        System.out.println("Elements :" + hm);
        System.out.println(hm.get(2));
```

```
        hm.clear();
        System.out.println("Elements :" + hm);
        System.out.println("size:" + hm.size());
    }
}
```

b)LinkedHashMap

CODE:

```
import java.util.*;
public class LinkedM8b {
    public static void main(String[] args) {
            HashMap<Integer,String> hm=new HashMap<Integer,String>();
            System.out.println("Is Empty or not:"+hm.isEmpty());
            hm.put(1,"A");
            hm.put(2,"B");
            hm.put(3,"C");
            System.out.println(hm);
            hm.remove(1,"D");
            System.out.println("Elements :"+hm);
            hm.replace(1,"D");
            System.out.println("Elements :"+hm);
            System.out.println(hm.get(2));
            hm.clear();
            System.out.println("Elements :"+hm);
            System.out.println("size:"+hm.size());
      }
 }
```

c)TreeMap

CODE:

```
import java.util.*;
public class Tree8c {
    public static void main(String[] args) {
        HashMap<Integer, String> hm = new HashMap<Integer, String>();
        System.out.println("Is Empty or not:" + hm.isEmpty());
        hm.put(1, "A");
        hm.put(2, "B");
        hm.put(3, "C");
        System.out.println(hm);
        hm.remove(1, "D");
        System.out.println("Elements :" + hm);
        hm.replace(1, "D");
        System.out.println("Elements :" + hm);
        System.out.println(hm.get(2));
        hm.clear();
        System.out.println("Elements :" + hm);
        System.out.println("size:" + hm.size());
    }
}
```

9.Write a java program to implement Sorted Chain.

CODE:

```
import java.util.Scanner;
class NodeClass{
    NodeClass next;
    int data;
    NodeClass(int data){
```

```
        this.data = data;
        this.next = null;
    }
}
class SortedChainLinkedList{
    NodeClass head;
    public SortedChainLinkedList(){
        this.head = null;
    }
    public void insert(int data){
        NodeClass newNode = new NodeClass(data);
        if(head == null || data < head.data){
            newNode.next = head;
            head = newNode;
        } else {
            NodeClass current = head;
            while(current.next !=null && current.next.data < data){
                current = current.next;
            }
            newNode.next = current.next;
            current.next = newNode;
        }
    }
    public void display(){
        NodeClass temp = head;
        System.out.println("The Elements are: ");
        while(temp != null){
            System.out.print(temp.data + "-->");
            temp  = temp.next;
        }
        System.out.println("END");
    }
}
public class SortedChainDemo {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        SortedChainLinkedList list = new SortedChainLinkedList();
        System.out.println("Enter no:of elements to insert:");
        int n=sc.nextInt();
        System.out.println("Enter The Element You wanted To insert: ");
        int el = sc.nextInt();
        list.insert(el);
        System.out.println("Elements are:");
        list.display();
    }
}
```

10.Write a java program to implement Seperate Chaining.

CODE:

```
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);

            }
        }
    }
}
```

11.Write a java program to implement Linear Probing.

CODE:

```
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);
        System.out.println("Enter size:");
        int size = sc.nextInt();
        LinearProbing lp = new LinearProbing(size);
        System.out.println("1.INSERT\n2.SEARCH\n3.EXIT:");
        while (true) {
            System.out.println("Enter choice:");
            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 !!");
            }
        }
    }
}
```

12.Implement BST using collection API.

CODE:

```
import java.util.*;

class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}

class BST {
    Node root;

    public BST() {
        root = null;
    }

    public void insert(int key) {
        root = insertRecursive(root, key);
    }

    private Node insertRecursive(Node root, int key) {
        if (root == null) {
            root = new Node(key);
            return root;
        }

        if (key < root.key)
            root.left = insertRecursive(root.left, key);
        else if (key > root.key)
            root.right = insertRecursive(root.right, key);
```

```
    return root;
}

Node search(Node root, int key) {
    if (root == null || root.key == key)
        return root;
    if (key < root.key)
        return search(root.left, key);
    else
        return search(root.right, key);
}

Node delNode(Node root, int key) {
    if (root == null)
        return root;
    if (key < root.key)
        root.left = delNode(root.left, key);
    else if (key > root.key)
        root.right = delNode(root.right, key);
    else {
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;
        Node minValueNode = minValueNode(root.right);
        root.key = minValueNode.key;
        root.right = delNode(root.right, minValueNode.key);
    }
    return root;
}

Node minValueNode(Node root) {
    Node current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current;
}

public void inOrderTraversal() {
    inOrderRecursive(root);
}

private void inOrderRecursive(Node root) {
    if (root != null) {
        inOrderRecursive(root.left);
        System.out.print(root.key + " ");
        inOrderRecursive(root.right);
    }
}

public void preOrderTraversal() {
    preOrderRecursive(root);
}

private void preOrderRecursive(Node root) {
    if (root != null) {
        System.out.print(root.key + " ");
        preOrderRecursive(root.left);
        preOrderRecursive(root.right);
    }
}
```

```
    public void postOrderTraversal() {
        postOrderRecursive(root);
    }

    private void postOrderRecursive(Node root) {
        if (root != null) {
            postOrderRecursive(root.left);
            postOrderRecursive(root.right);
            System.out.print(root.key + " ");
        }
    }
}

public class MainBST {
    public static void main(String[] args) {
        BST bst = new BST();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter no:of elements:");
        int n = sc.nextInt();
        System.out.println("Enter Elements:");
        for (int i = 0; i < n; i++) {
            bst.insert(sc.nextInt());
        }
        System.out.println("Element to Search:");
        int eleS = sc.nextInt();
        Node find = bst.search(bst.root, eleS);
        if (find != null)
            System.out.println("Element" + eleS + " found");
        else
            System.out.println("Element" + eleS + " not found");
        System.out.println("Enter element to delete:");
        int eleD = sc.nextInt();
        bst.root = bst.delNode(bst.root, eleD);
        System.out.print("In-order traversal: ");
        bst.inOrderTraversal();
        System.out.print("\nPre-order traversal: ");
        bst.preOrderTraversal();
        System.out.print("\nPost-order traversal: ");
        bst.postOrderTraversal();
    }
}
```

13.Implement AVL tree using Collection API.

CODE:

```
import java.util.*;
public class AVLtree {
    TreeSet<Integer> t = new TreeSet<Integer>();

    public void insert(int ele) {
        t.add(ele);
    }

    public void delete(int ele) {
        if (t.contains(ele)) {
            t.remove(ele);
        } else {
            System.out.println("ELement not found..");
        }
    }

    public void search(int ele) {
```

```
        if (t.contains(ele)) {
            System.out.println("Element Found..");
        } else {
            System.out.println("ELement not found..");
        }
    }

    public void display() {
        System.out.println(t);
    }

    public static void main(String[] args) {
        AVLtree a = new AVLtree();
        Scanner sc = new Scanner(System.in);
        System.out.println("1.Insert\n2.Delete\n3.Search\n4.Display\n5.Exit");
        while (true) {
            System.out.println("Enter Choice:");
            int ch = sc.nextInt();
            switch (ch) {
                case 1:
                    System.out.println("Enter element to insert:");
                    int eleI = sc.nextInt();
                    a.insert(eleI);
                    break;
                case 2:
                    System.out.println("Enter element to delete:");
                    int eleD = sc.nextInt();
                    a.delete(eleD);
                    break;
                case 3:
                    System.out.println("Enter element to Search:");
                    int eleS = sc.nextInt();
                    a.search(eleS);
                    break;
                case 4:
                    System.out.println("Elements in Tree are:");
                    a.display();
                    break;
                case 5:
                    System.exit(0);
                default:
                    System.out.println("Wrong Choice..");
            }
        }
    }
}
```

14.Implement priority queues with max heap tree using collection API.

CODE:

```
import java.util.*;

public class MaxHeapPriorityQueue {
    private PriorityQueue<Integer> pq;

    public MaxHeapPriorityQueue() {
        pq = new PriorityQueue<>(Collections.reverseOrder());
    }

    public void enqueue(int value) {
        pq.add(value);
    }
```

```
    public int dequeue() {
        if (pq.isEmpty()) {
            System.out.println("priority queue is empty");
        }
        return pq.remove();
    }

    public void printHeap() {
        System.out.println(pq);
    }

    public boolean isEmpty() {
        return pq.isEmpty();
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        MaxHeapPriorityQueue mp = new MaxHeapPriorityQueue();
        while (true) {
            System.out.println("1.Insert 2.Delete 3.Display 4.Exit\n");
            System.out.println("enter choice:\n");
            int ch = sc.nextInt();
            switch (ch) {
                case 1:
                    System.out.println("enter no of elements:");
                    int numElements = sc.nextInt();
                    System.out.println("enter elements:");
                    for (int i = 0; i < numElements; i++) {
                        int value = sc.nextInt();
                        mp.enqueue(value);
                    }
                    break;
                case 2:
                    System.out.println("deleted max element:" + mp.dequeue());
                    break;
                case 3:
                    System.out.println("elements in queue are:");
                    mp.printHeap();
                    break;
                case 4:
                    System.exit(0);
                default:
                    System.out.println("enter valid choice");
            }
        }
    }
}
```

15.Implement Heap Sort with max Heap tree using Collection API.

CODE:

```
import java.util.*;

public class HeapSort {
    private PriorityQueue<Integer> pq;
    public static ArrayList<Integer> l = new ArrayList<Integer>();

    public HeapSort() {
        pq = new PriorityQueue<>(Collections.reverseOrder());
    }

    public void enqueue(int value) {
```

```
        pq.add(value);
    }

    public void dequeue() {
        while (!pq.isEmpty()) {
            l.add(pq.remove());
        }
        System.out.println("sorted heap=" + l);
    }

    public boolean isEmpty() {
        return pq.isEmpty();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        HeapSort obj = new HeapSort();
        System.out.println("enter no.of elemnets to insert");
        int n = sc.nextInt();
        System.out.println("enter elements");
        for (int i = 0; i < n; i++) {
            obj.enqueue(sc.nextInt());
        }
        obj.dequeue();
    }
}
```

16.Implement Boyer Moor algorithm.

CODE:

```
import java.util.*;
public class BoyerMoore{
 public static void main(String args[]) {
 Scanner s=new Scanner(System.in);
 System.out.println("enter text");
String text=s.next();
System.out.println("enter Pattern");
String pattern=s.next();
test(text, pattern);
 }
 public static void test(String text, String word) {
 char[] textC = text.toCharArray();
 char[] wordC = word.toCharArray();
 List<Integer> positions = bm(textC, wordC);
 if (!positions.isEmpty()) {
 System.out.println("Pattern Found at Positions: ");
 for (int position : positions) {
 System.out.println(position);
 }
 } else {
 System.out.println("Pattern Not Found");
 System.out.println("\ttext: " + text);
 System.out.println("\tword: " + word);
 }
 }
 public static List<Integer> bm(char[] string, char[] pat) {
 List<Integer> positions = new ArrayList<>();
 int[] d1 = makeD1(pat);
 int[] d2 = makeD2(pat);
 int i = pat.length - 1;
 int j = pat.length - 1;
 while (i < string.length) {
 if (string[i] == pat[j]) {
```

```
 if (j == 0) {
 positions.add(i);
 i += pat.length;
 j = pat.length - 1;
 } else {
 i--;
 j--;
 }
 } else {
 int x = d1[string[i]];
 int y = d2[j];
 i += Math.max(x, y);
 j = pat.length - 1;
 }
 }
 return positions;
 }
 public static int[] makeD1(char[] pat) {
 int[] table = new int[255];
 for (int i = 0; i < 255; i++) {
 table[i] = pat.length;
 }
 for (int i = 0; i < pat.length - 1; i++) {
 table[pat[i]] = pat.length - 1 - i;
 }
 return table;
 }
 public static boolean isPrefix(char[] word, int pos) {
 int suffixlen = word.length - pos;
 for (int i = 0; i < suffixlen; i++) {
 if (word[i] != word[pos + i]) {
 return false;
 }
 }
 return true;
 }

public static int suffix_length(char[] word, int pos) {
 int i;
 for (i = 0; ((word[pos - i] == word[word.length - 1 - i]) & (i < pos)); i++) {
 }
 return i;
 }
 public static int[] makeD2(char[] pat) {
 int[] delta2 = new int[pat.length];
 int p;
 int last_prefix_index = pat.length - 1;
 for (p = pat.length - 1; p >= 0; p--) {
 if (isPrefix(pat, p + 1))
 last_prefix_index = p + 1;
 delta2[p] = last_prefix_index + (pat.length - 1 - p);
 }
 for (p = 0; p < pat.length - 1; p++) {
 int slen = suffix_length(pat, p);
 if (pat[p - slen] != pat[pat.length - 1 - slen])
 delta2[pat.length - 1 - slen] = pat.length - 1 - p + slen;
 }
 return delta2;
 }
}
```

17.Implement Knuth Morris Pratt algorithm.

CODE:

```
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);
 }
}
```