**Lab Session1**

Date:

Problem 1

Problem Statement:

Due to the demonetization move, there is a long queue of people in front of ATMs. Due to withdrawal limit per person per day, people come in groups to withdraw money. Groups come one by one and line up behind the already present queue. The groups have a strange way of arranging themselves. In a particular group, the group members arrange themselves in increasing order of their height (not necessarily strictly increasing).

Swapy observes a long queue standing in front of curious kid, he wants to count the total number of groups present in the queue waiting to withdraw money. Since groups are standing behind each other, one cannot differentiate between different groups and the exact count cannot be given. Can you tell him the minimum number of groups that can be observed in the queue?

Source Code:

package test;

import java.util.Scanner;

public class GroupHeight {

    public static void main(String[] args) {

        Scanner h = new Scanner(System.in);

        int i, j, group=1, numOfPeople;

        System.out.println("Enter the number of people :");

        numOfPeople=h.nextInt();

        int height[] = new int[numOfPeople];

        System.out.println("Enter height");

            for(i=0;i<numOfPeople;i++) {

                height[i]=h.nextInt();

            }

            for(i=0;i<numOfPeople;i++) {

                j=i+1;

                    if(j==numOfPeople)

                        break;

                        if(height[i]>height[j]) {

                            group=group+1;

                        }

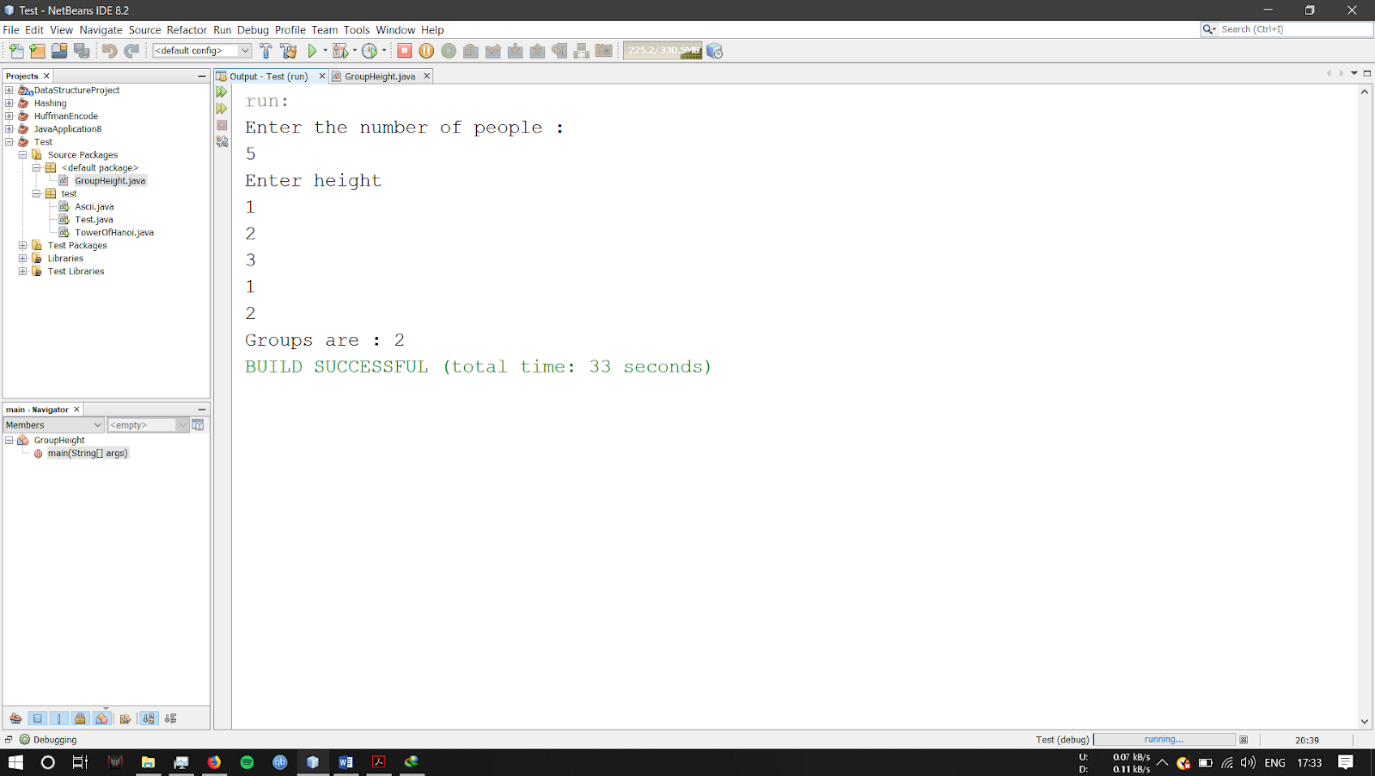
            }

        System.out.println("Groups are : "+group);

    }

}

Output:



Problem 2

You are given an array A of length N. You need to find maximum length of subsequence of A which contains anyone digit common in all the elements of that subsequence. You are not allowed to count leading zeroes.

Source Code:

import java.util.\*;

public class SubsequenceOfArray {

    public static void main(String[] args) {

        Scanner h=new Scanner(System.in);

        int i,n,t1,ans=0,st,t2;

        System.out.println("Number of elements:");

        n=h.nextInt();

        int num1[] = new int[n];

        int num2[][] = new int[10][n];

        System.out.println("Enter elements");

        for(i=0;i<n;i++) {

            num1[i] = h.nextInt();

        }

        for(i=0;i<10;i++) {

            for(int j=0;j<n;j++){

                t1=num1[j];

                while(t1>0) {

                    t2=t1%10;

                    num2[t2][j]=1;

                    t1=t1/10;

                }

            }

        }

        for(int k=0;k<10;k++) {

            for(int p=0;p<n;p++) {

                //System.out.println(num2[k][p]);

            }

        }

        for(int u=0;u<10;u++) {

            st=0;

            for(int v=0;v<n;v++) {

                if(num2[u][v]==1) {

                    st=st+1;

                }

            }

            ans=Math.max(st, ans);

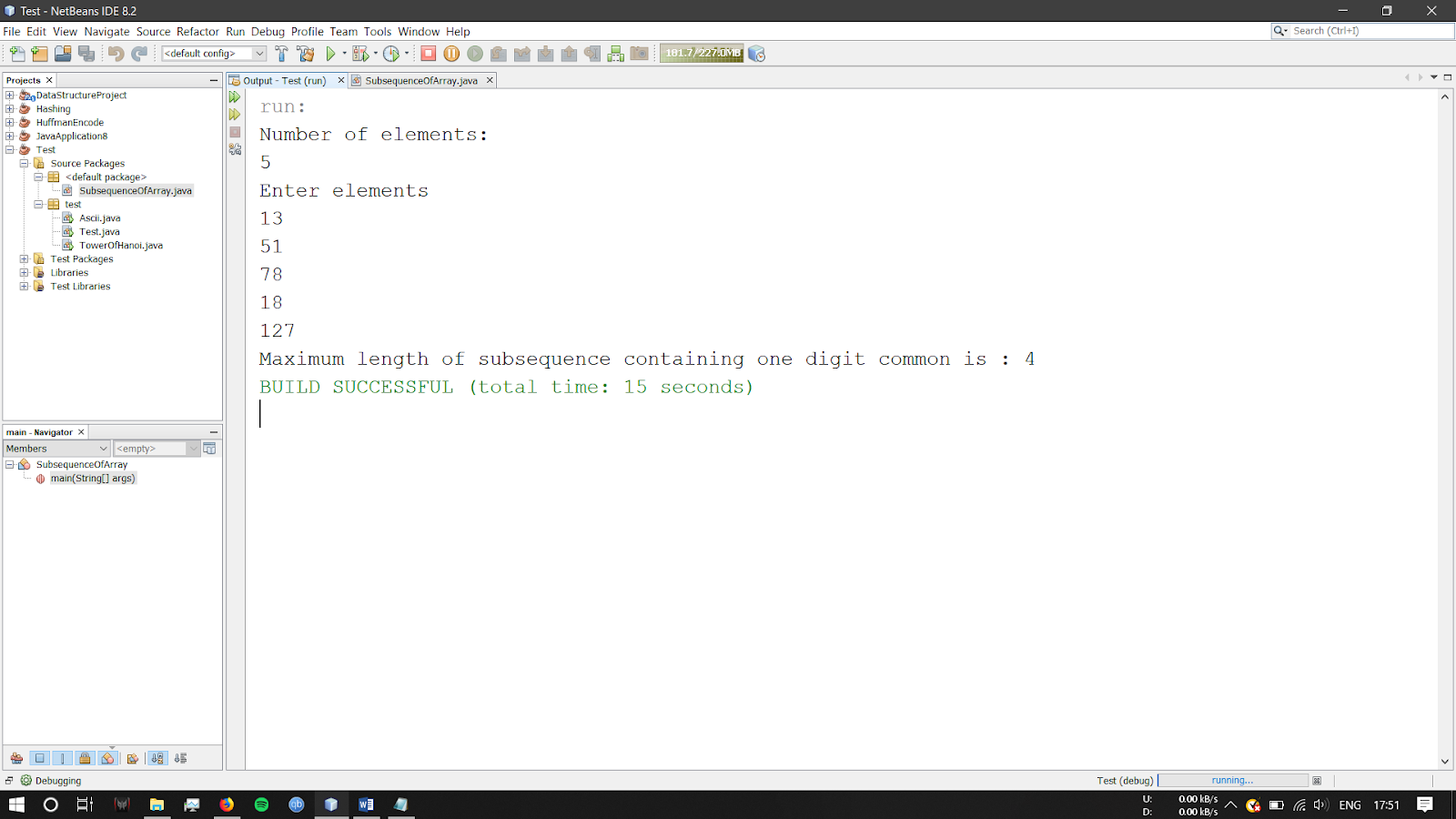
        }

        System.out.println("Maximum length of subsequence containing one digit common is : " + ans);

    }

}

Output:



**Lab Session 2**

Date:

Problem 1

Problem Statement:

Mr.X wants to change his profile picture on Instagram. Now Instagram has some restriction over the dimension of picture that we can upload.

Minimum dimension of the picture can be L x L, where L is the length of the side of square.

Now Roy has N photos of various dimensions.

Dimension of a photo is denoted as W x H

where W - width of the photo and H - Height of the photo

When any photo is uploaded following events may occur:

[1] If any of the width or height is less than L, user is prompted to upload another one. Print ‘UPLOAD ANOTHER’ in this case.

[2] If width and height, both are large enough and

(a) if the photo is already square then it is accepted. Print ‘ACCEPTED’ in this case.

(b) else user is prompted to crop it. Print &quot;CROP IT&quot; in this case.

Given L, N, W and H as input, print appropriate text as output.

Source Code:

import java.util.Scanner;

public class InstagramImageDimensions {

    public static void main(String[] args){

        Scanner in = new Scanner(System.in);

        int l, n, i, w, h;

        do {

            System.out.println("Enter the dimension of the image : ");

                while (!in.hasNextInt()) {

                System.out.println("Please Enter an Integer Value!");

                in.next();

                }

                l = in.nextInt();

        } while (l <= 0);

        do {

            System.out.println("Enter the number of pictures you want to upload :");

                while (!in.hasNextInt()) {

                System.out.println("Please Enter an Integer Value!");

                in.next();

                }

                n = in.nextInt();

        } while (l <= 0);

        for(i=0;i<n;i++){

            do {

                System.out.println("Enter the Width");

                    while (!in.hasNextInt()) {

                    System.out.println("Please Enter Integer Values!");

                    System.out.println("Enter the Width");

                    in.next();

                    }

                w=in.nextInt();

        } while (l <= 0);

        do {

        System.out.println("Enter the Height");

            while (!in.hasNextInt()) {

            System.out.println("Please Enter Integer Values!");

            System.out.println("Enter the Height");

            in.next();

            }

            h=in.nextInt();

        } while (l <= 0);

            if(w<l && h<l){

                System.out.println("Upload Another");

            }

            if(w>l && h>l){

                System.out.println("Crop it");

            }

            if(w==l && h==l){

                System.out.println("Accepted");

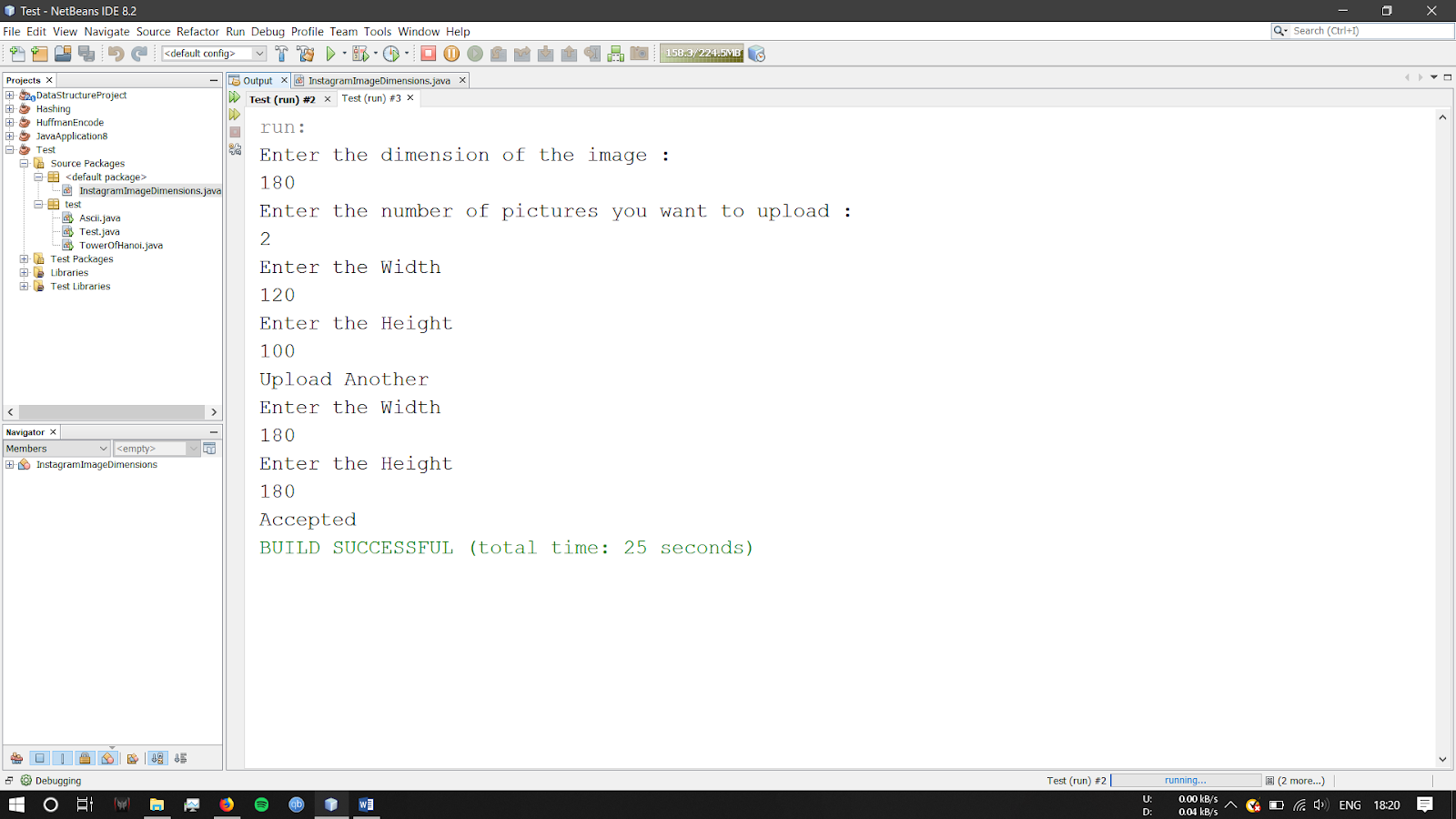
            }

        }

    }

}

Output:



Problem 2

Problem Statement

Given a 6\*6 2D Array

1 1 1 0 00

0 1 0 0 00

1 1 1 0 00

0 0 0 0 00

0 0 0 0 00

0 0 0 0 00

We define a pattern in 2D Array to be a subset of values with indices falling in this pattern in 's graphical representation:

a b c

  d

e f g

There are 16 patterns in 2D Array . Calculate the sum for every pattern in the 2D Array , then print the maximum patterns sum.

Source Code:

**import** java.util.Scanner;

**public** **class** Array3 {

/\*\*

\* **@param** args the command line arguments

\*/

**public** **static** **void** main(String[] args) {

// **TODO** code application logic here

**int**[][] a = **new** **int**[6][6];

**int** i, j, temp = 0, max = 0, min = Integer.***MIN\_VALUE***;

Scanner s = **new** Scanner(System.***in***);

System.***out***.println("Enter elements:");

**for** (i = 0; i < 6; i++)

**for** (j = 0; j < 6; j++) {

a[i][j] = s.nextInt();

}

/\*

\* min=a[1][1]+a[1][2]+a[1][3]+a[2][3]+a[3][1]+a[3][2]+a[3][3]; if(min<=0) {

\* max=min;

\*

\* }

\*/

**for** (i = 0; i < 4; i++) {

**for** (j = 0; j < 4; j++) {

temp = a[i][j] + a[i][j + 1] + a[i][j + 2] + a[i + 1][j + 1] + a[i + 2][j] + a[i + 2][j + 1]

+ a[i + 2][j + 2];

// System.out.println("ans"+temp);

min = Math.*max*(min, temp);

}

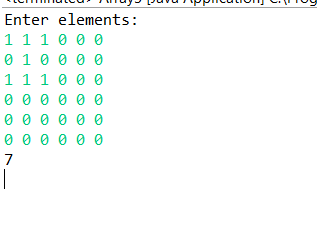
}

System.***out***.println(min);

}

}

Output



**Lab Session 4**

Date:

Problem 1

Problem Statement:

You have an empty sequence, and you will be given N queries. Each query is one of these three types:

1   -Push the element x into the stack.

2     -Delete the element present at the top of the stack.

3     -Print the maximum element in the stack.

Source Code:

import java.util.Scanner;

import java.util.Stack;

public class Stack5 {

public static void main(String[] args) {

// TODO Auto-generated method stub

int i;

int ch;

int number,sum1,sum2,sum3=0;

Stack<Integer> stk1 = new Stack();

Stack<Integer> stk2 = new Stack();

Scanner input = new Scanner(System.*in*);

System.*out*.println("Enter number of stacks");

number = input.nextInt();

System.*out*.println("Enter ");

for(i=0; i<number;i++)

{

sum2 = input.nextInt();

stk1.push(sum2);

}

for(i=0;i<number;i++)

{

sum1=stk1.pop();

System.*out*.println(""+sum1);

stk2.push(sum1);

}

int flag=0;

while(flag!=1)

{

System.*out*.println("Enter 1 to push, 2 to pop, 3 to stop");

ch = input.nextInt();

if(ch ==1)

{

sum2 = input.nextInt();

stk1.push(sum2);

System.*out*.println("the number pushed is"+sum2);

}

if(ch ==2)

{

int s= stk2.pop();

System.*out*.println("The popped is"+s);

}

if(stk2.isEmpty())

{

while(!stk1.isEmpty())

{

int s =stk1.pop();

stk2.push(s);

}

}

if(ch==3)

{

flag=1;

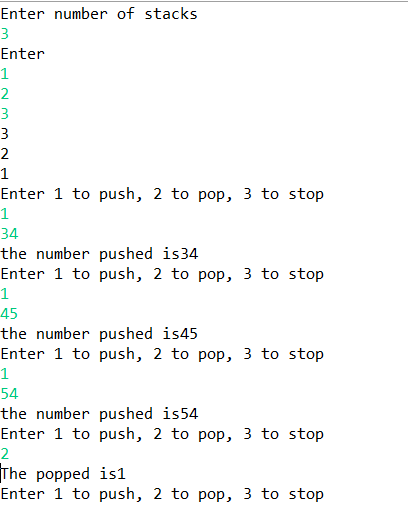
}

}

}

}

Output:



Problem 2

Problem Statement:

A bracket is considered to be any one of the following characters: (, ), {, }, [, or ].

Two brackets are considered to be a matched pair if the an opening bracket (i.e., (, [, or {) occurs to the left of a closing bracket (i.e., ), ], or }) of the exact same type. There are three types of matched pairs of brackets: [], {},and ().

A matching pair of brackets is not balanced if the set of brackets it encloses are not matched. For example, {[(])} is not balanced because the contents in between { and } are not balanced. The pair of square brackets encloses a single, unbalanced opening bracket, (, and the pair of parentheses encloses a single, unbalanced closing square bracket, ].

By this logic, we say a sequence of brackets is balanced if the following conditions are met:

It contains no unmatched brackets.

The subset of brackets enclosed within the confines of a matched pair of brackets is also a matched pair of brackets.

Given n strings of brackets, determine whether each sequence of

brackets is balanced. If a string is balanced, return YES. Otherwise, return NO.

Source Code:

import java.util.Scanner;

import java.util.Stack;

public class Stack3 {

public static void main(String[] args) {

int i;

Stack<Character> stk = new Stack();

Scanner input = new Scanner(System.*in*);

System.*out*.println("Enter n number of values");

int number = input.nextInt();

int num=number+1;

String a[]=new String[num];

for(i=0; i<num;i++)

{

a[i] = input.nextLine();

}

char ch;

for(i=0; i<num;i++)

{

String g=a[i];

OUTER\_LOOP:

for(int j=0;j<g.length();j++)

{

ch=g.charAt(j);

System.*out*.println("u"+ch);

if(ch=='{' || ch=='(' || ch=='[')

{

stk.push(ch);

}

else

{

if(ch==']' && stk.peek()=='[' || ch=='}' && stk.peek()=='{' || ch==')' && stk.peek()=='(')

{

stk.pop();

if(stk.isEmpty()) {

System.*out*.println("balanced");

}

}

else

{

stk.clear();

System.*out*.println("unbalanced");

break OUTER\_LOOP;

}

}

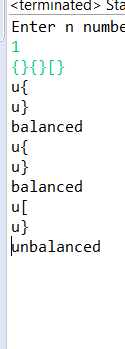
}

}

}

}

Output:



**Lab Session 5**

Date:24/7/19

Problem 1

Problem Statement:

A linked list is said to contain a cycle if any node is visited more than once while traversing the list.

Your function must return a boolean denoting whether or not there is a cycle in the list. If there is a cycle, return true; otherwise, return false.

Source Code:

import java.util.LinkedList;

import java.util.Scanner;

public class linkedlist {

Node head;// head of list

Node link;

/\* Node Class \*/

class Node

{

int data;

Node next;

// Constructor to create a new node

Node(int d) {data = d; next = null; }

}

public void insert (int a) {

Node ent = new Node(a);

ent.next = null;

if (head == null)

{

head = ent;

link = ent;

}

else {

link.next=ent;

link=ent;

}

}

public void loop() {

Node last=head;

while(last.next != null) {

last=last.next;

}

last.next=head.next;

}

public void loopdetect() {

Node temp1 = head;

Node temp2 = head;

while(temp2.next.next != null) {

temp1=temp1.next;

temp2=temp2.next.next;

if(temp1==temp2){

System.*out*.println("loop detected");

break;

}

}

}

public void removeloop() {

Node temp1 = head;

Node temp2 = head;

while(temp2.next.next != null) {

temp1=temp1.next;

temp2=temp2.next.next;

if(temp1==temp2){

System.*out*.println("loop detected");

break;

}

}

temp2 = head.next;

int loopLength = 1;

while(temp2 != temp1) {

temp2=temp2.next;

loopLength++;

}

System.*out*.println("Loop length is " + loopLength);

temp2 = head.next;

while(temp2.next != temp1) {

temp2=temp2.next;

}

temp2.next=null;

System.*out*.println("Loop breaks");

}

public static void main(String[] args) {

linkedlist list = new linkedlist();

int n,i,m,c=0;

Scanner input = new Scanner(System.*in*);

System.*out*.println("how many numbers you want to insert in linked list");

n=input.nextInt();;

System.*out*.println("Enter numricals in linked list ");

for (i=0;i<n;i++)

{

m = input.nextInt();

list.insert (m);

}

list.loop();

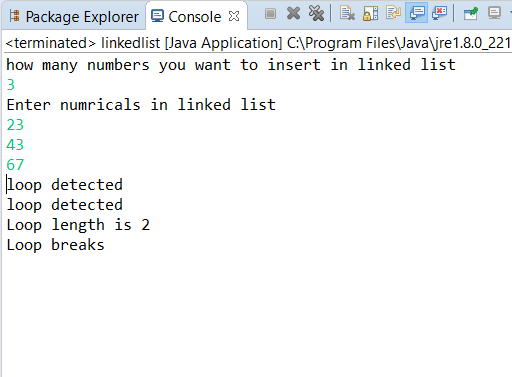
list.loopdetect();

list.removeloop();

}

}

Ouput:



Problem 2

Problem Statement:

Given a linked list, rotate the list to the right by k places, where k is non-negative.

Source Code:

package lb4;

//Program to rotate the nodes from last to first

import java.util.InputMismatchException;

import java.util.Scanner;

public class Rotatelist {

Node head;// head of list

Node link;

/\* Node Class \*/

class Node {

int data;

Node next;

// Constructor to create a new node

Node(int d) {

data = d;

next = null;

}

}

public void insert(int a) {

Node ent = new Node(a);

ent.next = null;

if (head == null) {

head = ent;

link = ent;

} else {

link.next = ent;

link = ent;

}

}

public void rotate(int b) {

Node a = head;

Node x = null;

for (int i = 0; i < b; i++) {

while (a.next != null) {

x = a;

a = a.next;

}

x.next = null;

a.next = head;

head = a;

}

}

public void print() {

Node curr = head;

while (curr != null) {

System.out.println("node:" + curr.data);

curr = curr.next;

}

}

public static void main(String[] args) {

// TODO Auto-generated method stub

Rotatelist list = new Rotatelist();

int n, i, m, c = 0;

boolean st = false;

Scanner input = new Scanner(System.in);

while (!st) {

try {

System.out.println("Press : 1 :: To insert in list");

System.out.println("Press : 2 :: To rotate list");

System.out.println("Press : 3 :: To print list");

System.out.println("Press : 4 :: To stop");

n = input.nextInt();

switch (n) {

case 1:

System.out.println("Enter numricals in linked list ");

m = input.nextInt();

list.insert(m);

break;

case 2:

System.out.println("Enter number of rotations");

m = input.nextInt();

list.rotate(m);

break;

case 3:

System.out.println("Printing....");

list.print();

break;

case 4:

System.out.println("Stoping.......");

System.out.println("THANK YOU!!!!!");

st = true;

break;

default:

System.out.println("Choose correct number");

break;

}

} catch (InputMismatchException e) {

System.out.println("Enter numericals");

}

}

}

}

Output:

