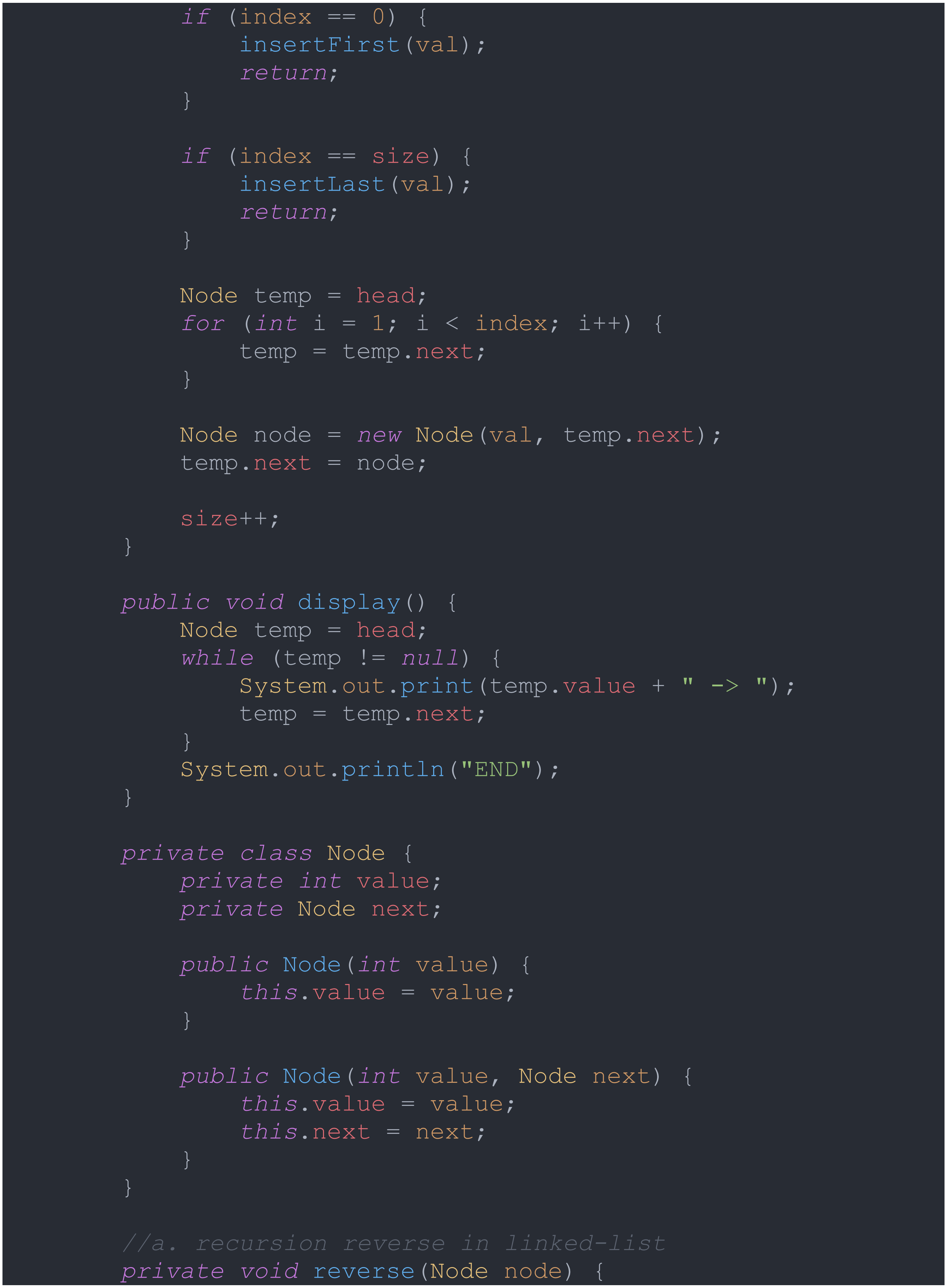
Name : Hardik Patel
Std Id : 202103032

Q.1 Write the code for the following problems through Recursion:

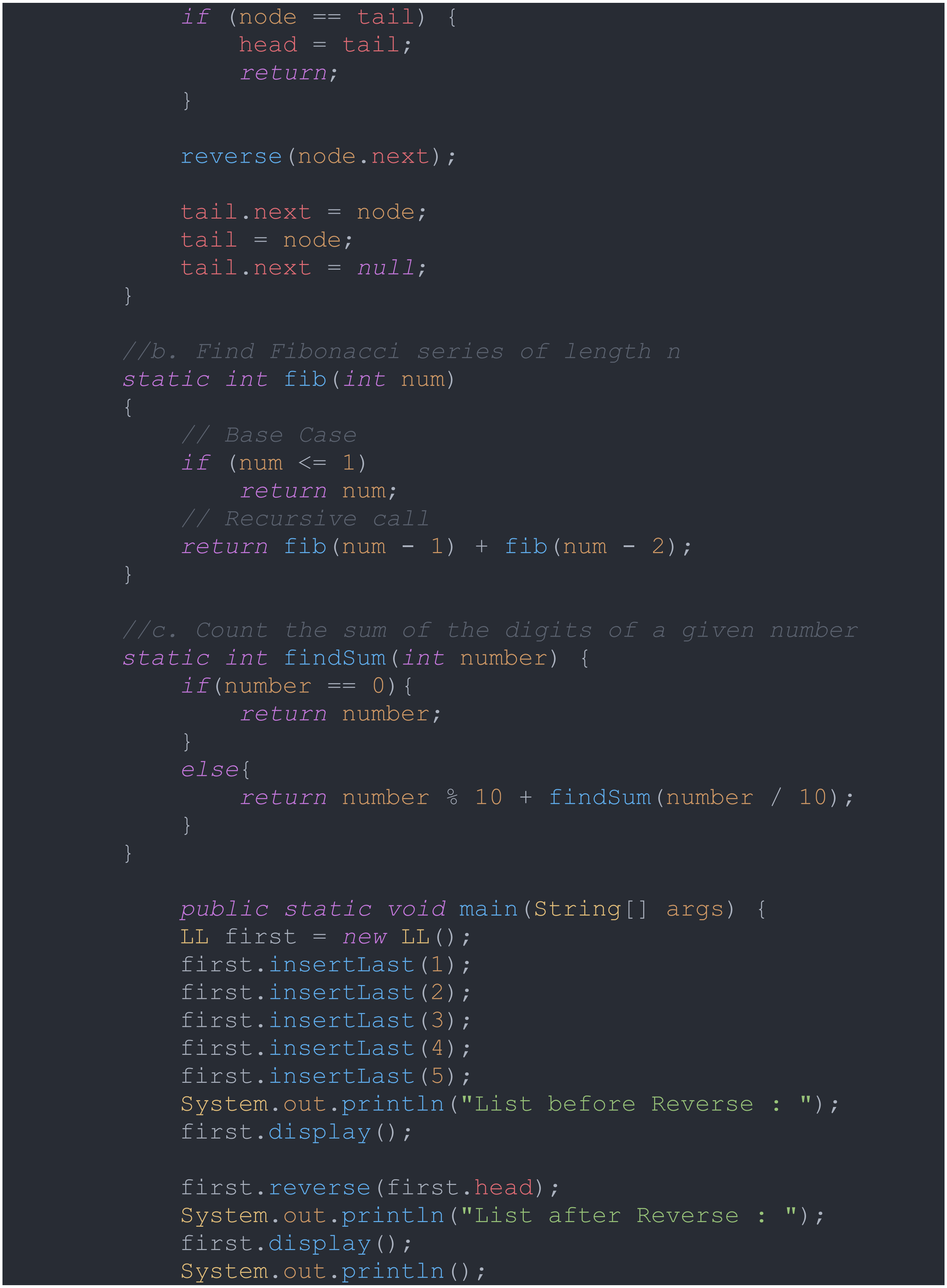
a. Reverse the linked list through the recursion.
b. Find Fibonacci series of length n
c. Count the sum of the digits of a given numb

Code :

```
package com.DSA.LAB8;

public class LL {

    private Node head;
    private Node tail;
    private int size;

        public LL() {
            this.size = 0;
        }

        public void insertFirst(int val) {
            Node node = new Node(val);
            node.next = head;
            head = node;
            if (tail == null) {
                tail = head;
            }
            size += 1;
        }

        public void insertLast(int val) {
            if (tail == null) {
                insertFirst(val);
                return;
            }
            Node node = new Node(val);
            tail.next = node;
            tail = node;
            size++;
        }

        public void insert(int val, int index) {
```



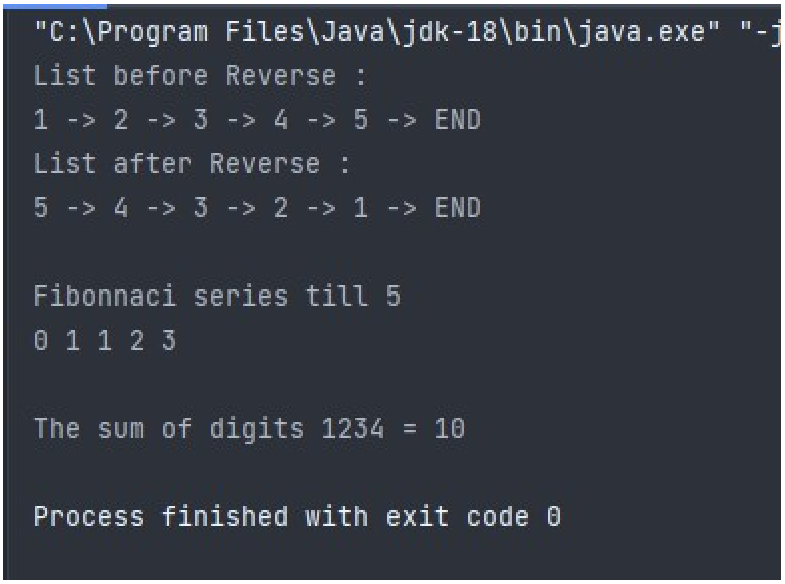
```java
        if (index == 0) {
            insertFirst(val);
            return;
        }

        if (index == size) {
            insertLast(val);
            return;
        }

        Node temp = head;
        for (int i = 1; i < index; i++) {
            temp = temp.next;
        }

        Node node = new Node(val, temp.next);
        temp.next = node;

        size++;
    }

    public void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.value + " -> ");
            temp = temp.next;
        }
        System.out.println("END");
    }

    private class Node {
        private int value;
        private Node next;

        public Node(int value) {
            this.value = value;
        }

        public Node(int value, Node next) {
            this.value = value;
            this.next = next;
        }
    }

    //a. recursion reverse in linked-list
    private void reverse(Node node) {
```



```
        if (node == tail) {
            head = tail;
            return;
        }

        reverse(node.next);

        tail.next = node;
        tail = node;
        tail.next = null;
    }

    //b. Find Fibonacci series of length n
    static int fib(int num)
    {
        // Base Case
        if (num <= 1)
            return num;
        // Recursive call
        return fib(num - 1) + fib(num - 2);
    }

    //c. Count the sum of the digits of a given number
    static int findSum(int number) {
        if(number == 0){
            return number;
        }
        else{
            return number % 10 + findSum(number / 10);
        }
    }

        public static void main(String[] args) {
        LL first = new LL();
        first.insertLast(1);
        first.insertLast(2);
        first.insertLast(3);
        first.insertLast(4);
        first.insertLast(5);
        System.out.println("List before Reverse : ");
        first.display();

        first.reverse(first.head);
        System.out.println("List after Reverse : ");
        first.display();
        System.out.println();
```

```
            int num = 5;
            System.out.println("Fibonnaci series till " + num);

            for (int i = 0; i < num; i++) {
                System.out.print(fib(i) + " ");
            }

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

            int number = 1234;
            System.out.println("The sum of digits "+ number + "
= " + findSum(number));

            }
    }
```

Output :

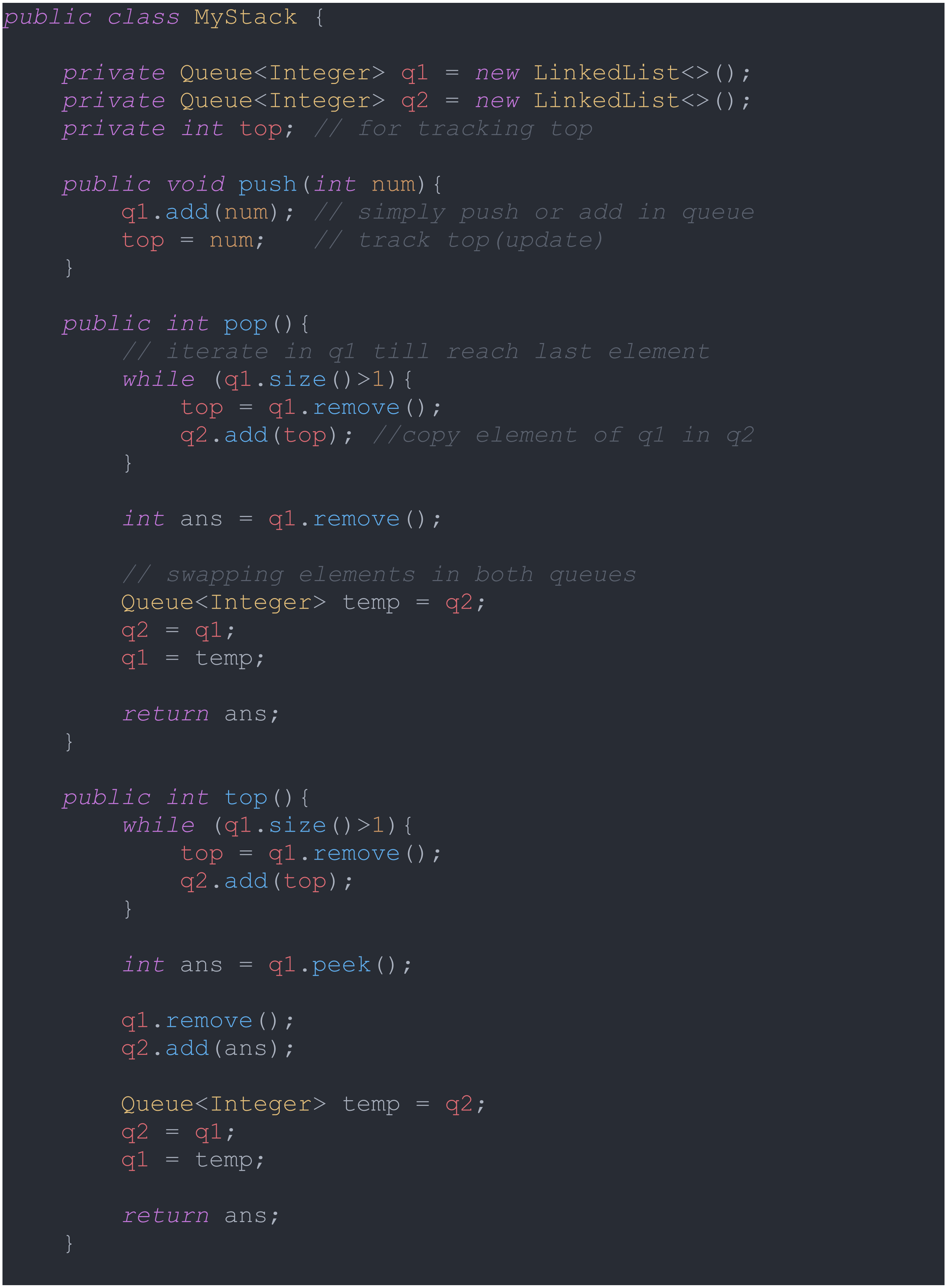


```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-j
List before Reverse :
1 -> 2 -> 3 -> 4 -> 5 -> END
List after Reverse :
5 -> 4 -> 3 -> 2 -> 1 -> END

Fibonnaci series till 5
0 1 1 2 3

The sum of digits 1234 = 10

Process finished with exit code 0
```

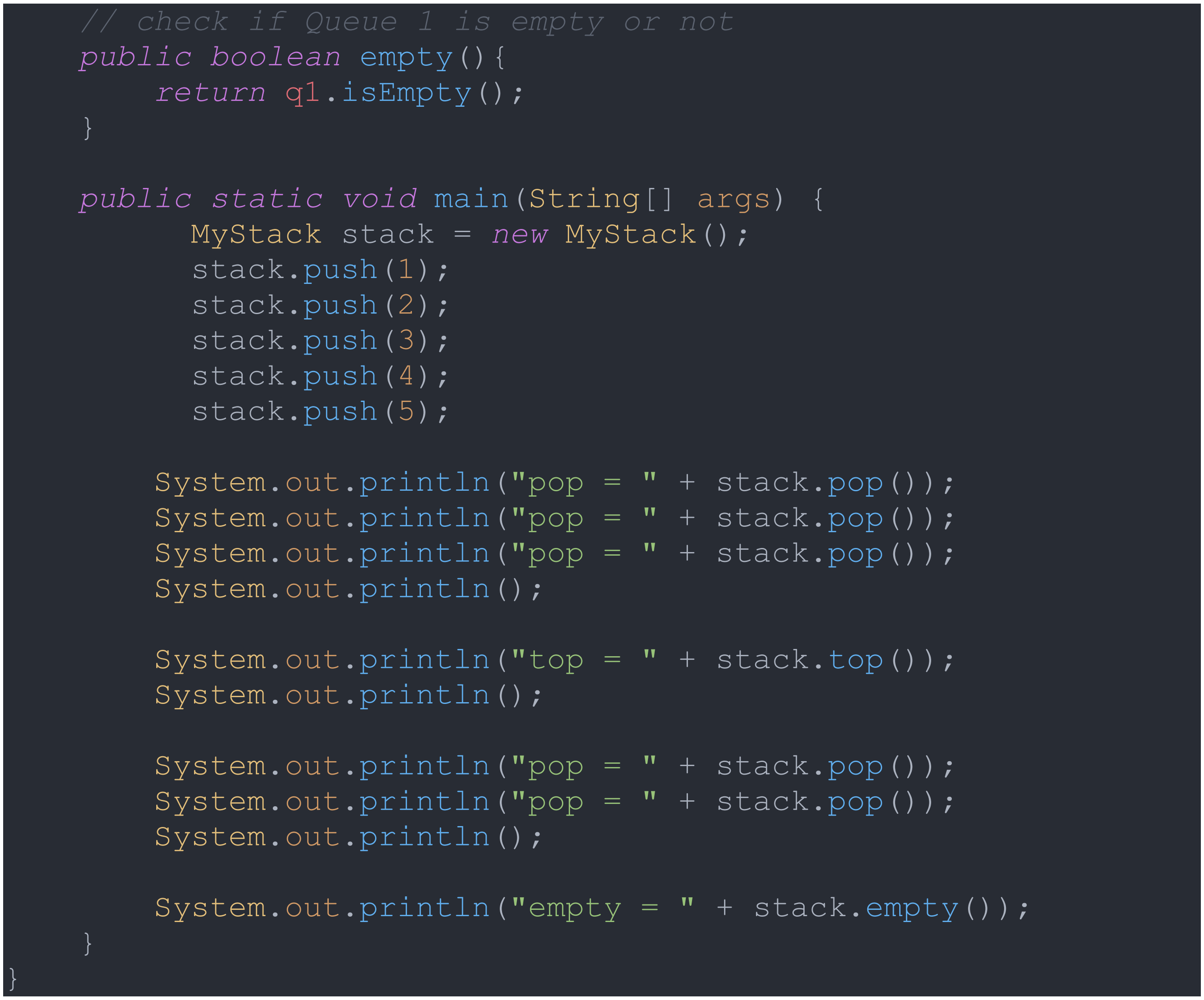
Q.2 Implement a stack using queues (only 2 queues). The implemented stack should support all the functions of a normal stack (push, top, pop, and empty)

Code :

```
package com.DSA.LAB8;

import java.util.LinkedList;
import java.util.Queue;
```

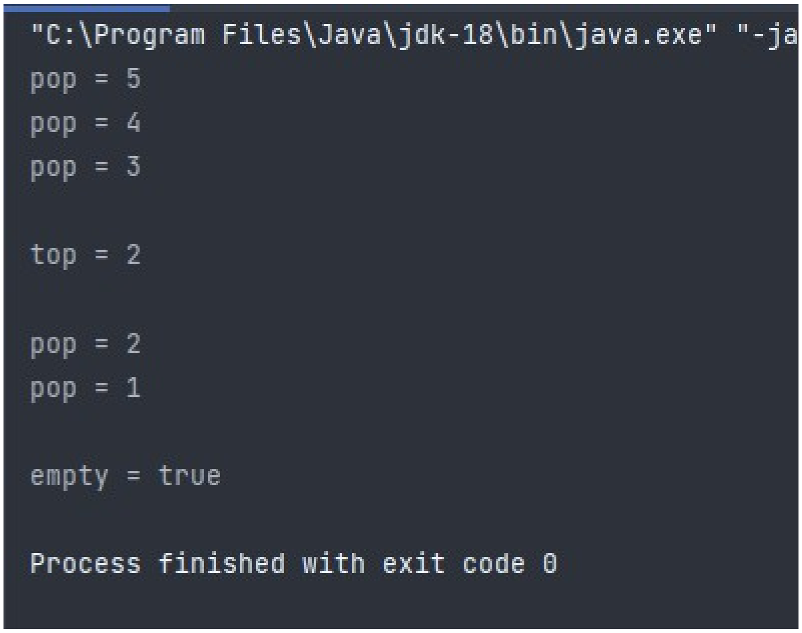


```java
public class MyStack {

    private Queue<Integer> q1 = new LinkedList<>();
    private Queue<Integer> q2 = new LinkedList<>();
    private int top; // for tracking top

    public void push(int num){
        q1.add(num); // simply push or add in queue
        top = num;   // track top(update)
    }

    public int pop(){
        // iterate in q1 till reach last element
        while (q1.size()>1){
            top = q1.remove();
            q2.add(top); //copy element of q1 in q2
        }

        int ans = q1.remove();

        // swapping elements in both queues
        Queue<Integer> temp = q2;
        q2 = q1;
        q1 = temp;

        return ans;
    }

    public int top(){
        while (q1.size()>1){
            top = q1.remove();
            q2.add(top);
        }

        int ans = q1.peek();

        q1.remove();
        q2.add(ans);

        Queue<Integer> temp = q2;
        q2 = q1;
        q1 = temp;

        return ans;
    }
```



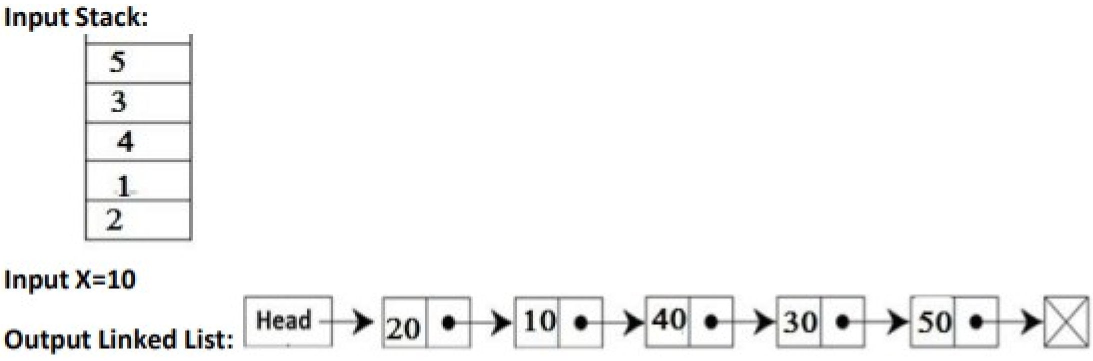
```
    // check if Queue 1 is empty or not
    public boolean empty(){
        return q1.isEmpty();
    }

    public static void main(String[] args) {
          MyStack stack = new MyStack();
          stack.push(1);
          stack.push(2);
          stack.push(3);
          stack.push(4);
          stack.push(5);

        System.out.println("pop = " + stack.pop());
        System.out.println("pop = " + stack.pop());
        System.out.println("pop = " + stack.pop());
        System.out.println();

        System.out.println("top = " + stack.top());
        System.out.println();

        System.out.println("pop = " + stack.pop());
        System.out.println("pop = " + stack.pop());
        System.out.println();

        System.out.println("empty = " + stack.empty());
    }
}
```

Output :

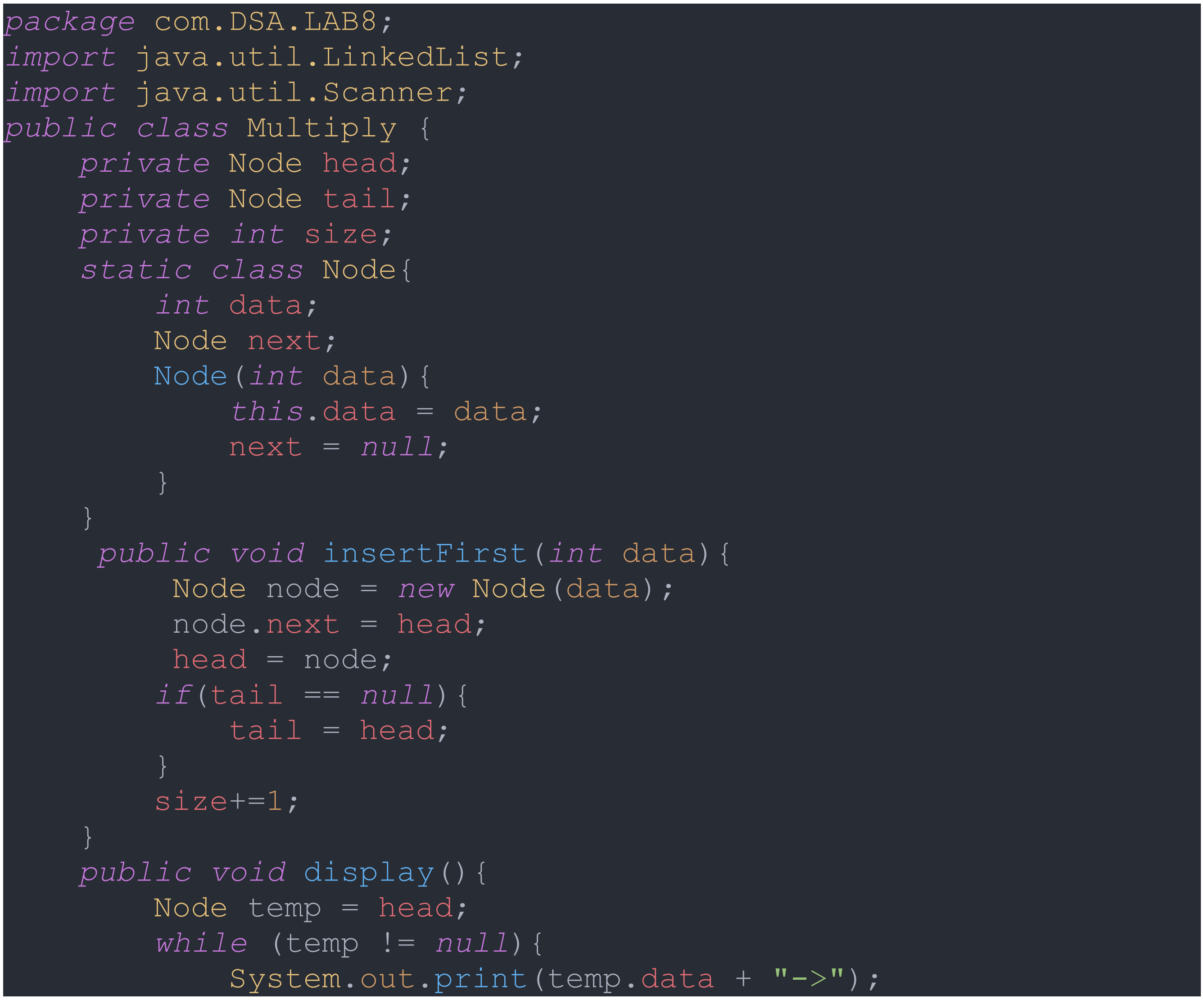


```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-ja
pop = 5
pop = 4
pop = 3

top = 2

pop = 2
pop = 1

empty = true

Process finished with exit code 0
```

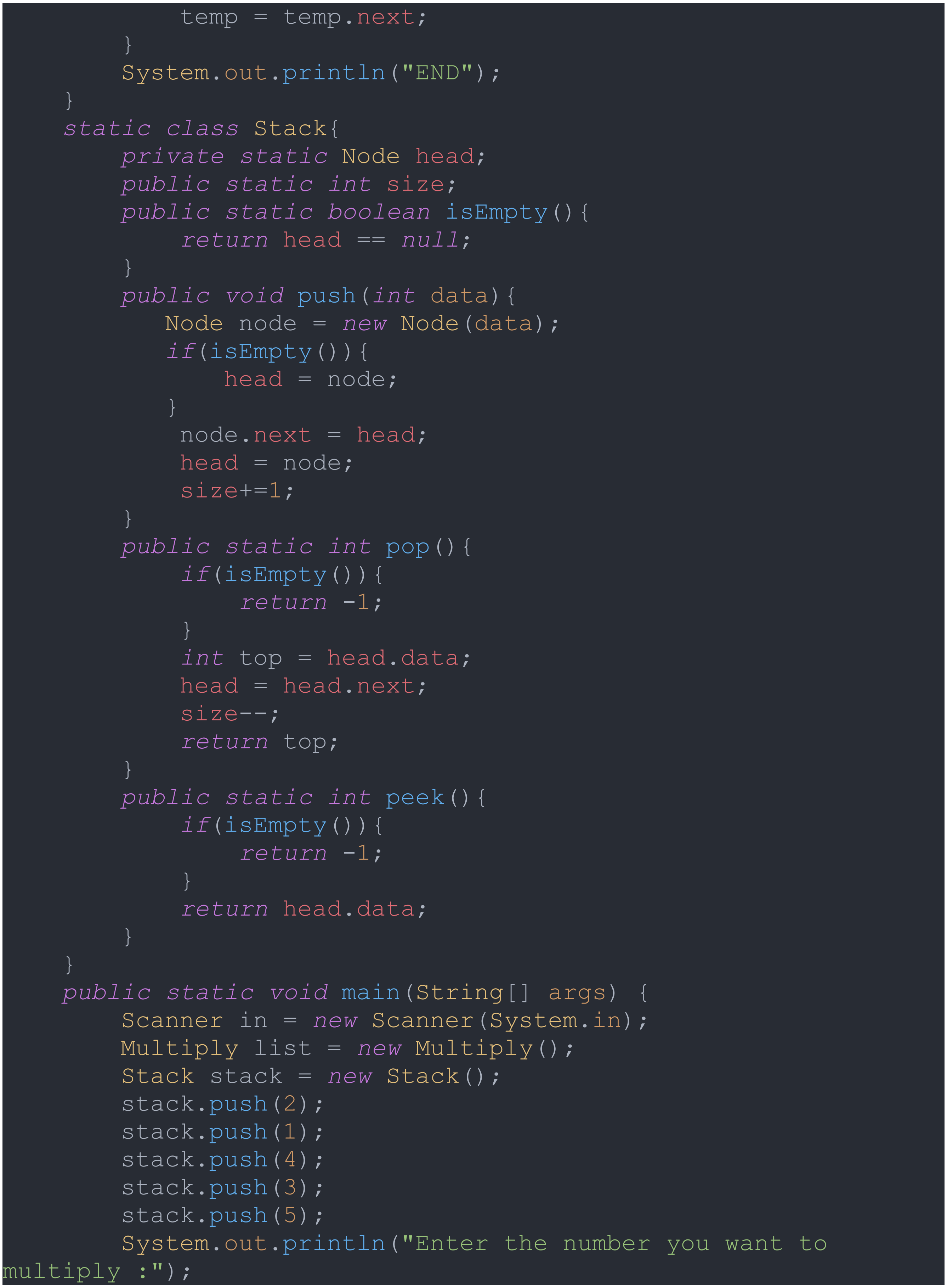
Q.3 Write function(s) to multiply each data of the stack with X and store the elements in a linked list (insert at head) as shown in figures bwlow.



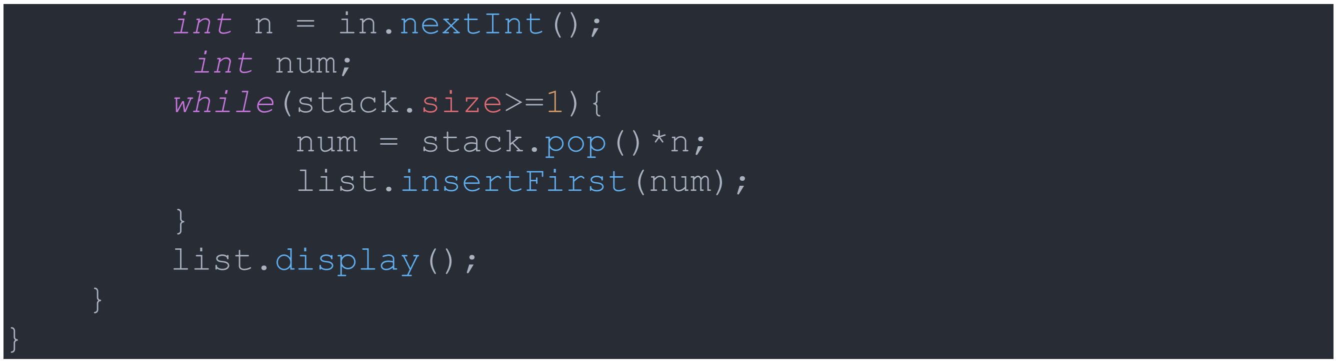
Code :



```
package com.DSA.LAB8;
import java.util.LinkedList;
import java.util.Scanner;
public class Multiply {
    private Node head;
    private Node tail;
    private int size;
    static class Node{
        int data;
        Node next;
        Node(int data){
            this.data = data;
            next = null;
        }
    }
     public void insertFirst(int data){
         Node node = new Node(data);
         node.next = head;
         head = node;
        if(tail == null){
            tail = head;
        }
        size+=1;
    }
    public void display(){
        Node temp = head;
        while (temp != null){
            System.out.print(temp.data + "->");
```

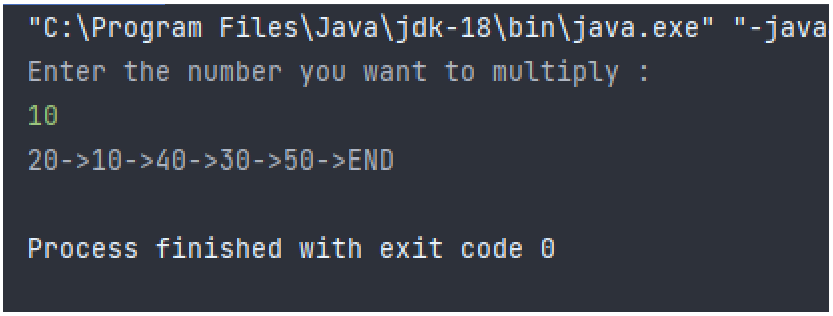


```
            temp = temp.next;
        }
        System.out.println("END");
    }
    static class Stack{
        private static Node head;
        public static int size;
        public static boolean isEmpty(){
            return head == null;
        }
        public void push(int data){
           Node node = new Node(data);
           if(isEmpty()){
               head = node;
           }
            node.next = head;
            head = node;
            size+=1;
        }
        public static int pop(){
            if(isEmpty()){
                return -1;
            }
            int top = head.data;
            head = head.next;
            size--;
            return top;
        }
        public static int peek(){
            if(isEmpty()){
                return -1;
            }
            return head.data;
        }
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Multiply list = new Multiply();
        Stack stack = new Stack();
        stack.push(2);
        stack.push(1);
        stack.push(4);
        stack.push(3);
        stack.push(5);
        System.out.println("Enter the number you want to
multiply :");
```



```
        int n = in.nextInt();
         int num;
        while(stack.size>=1){
              num = stack.pop()*n;
              list.insertFirst(num);
        }
        list.display();
    }
}
```

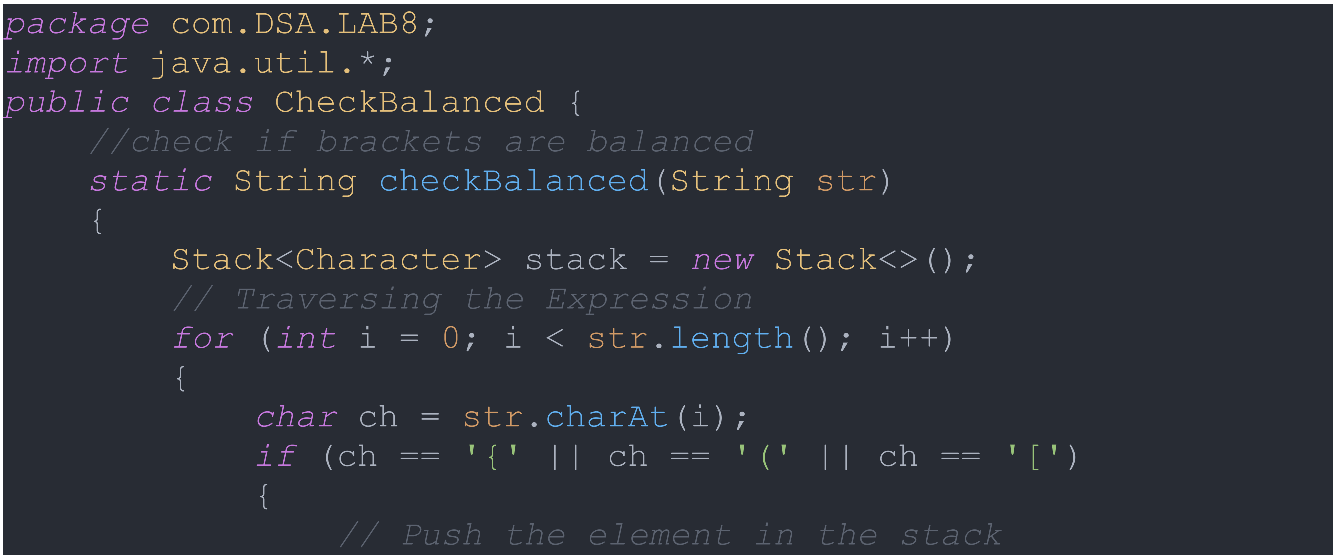
Output :



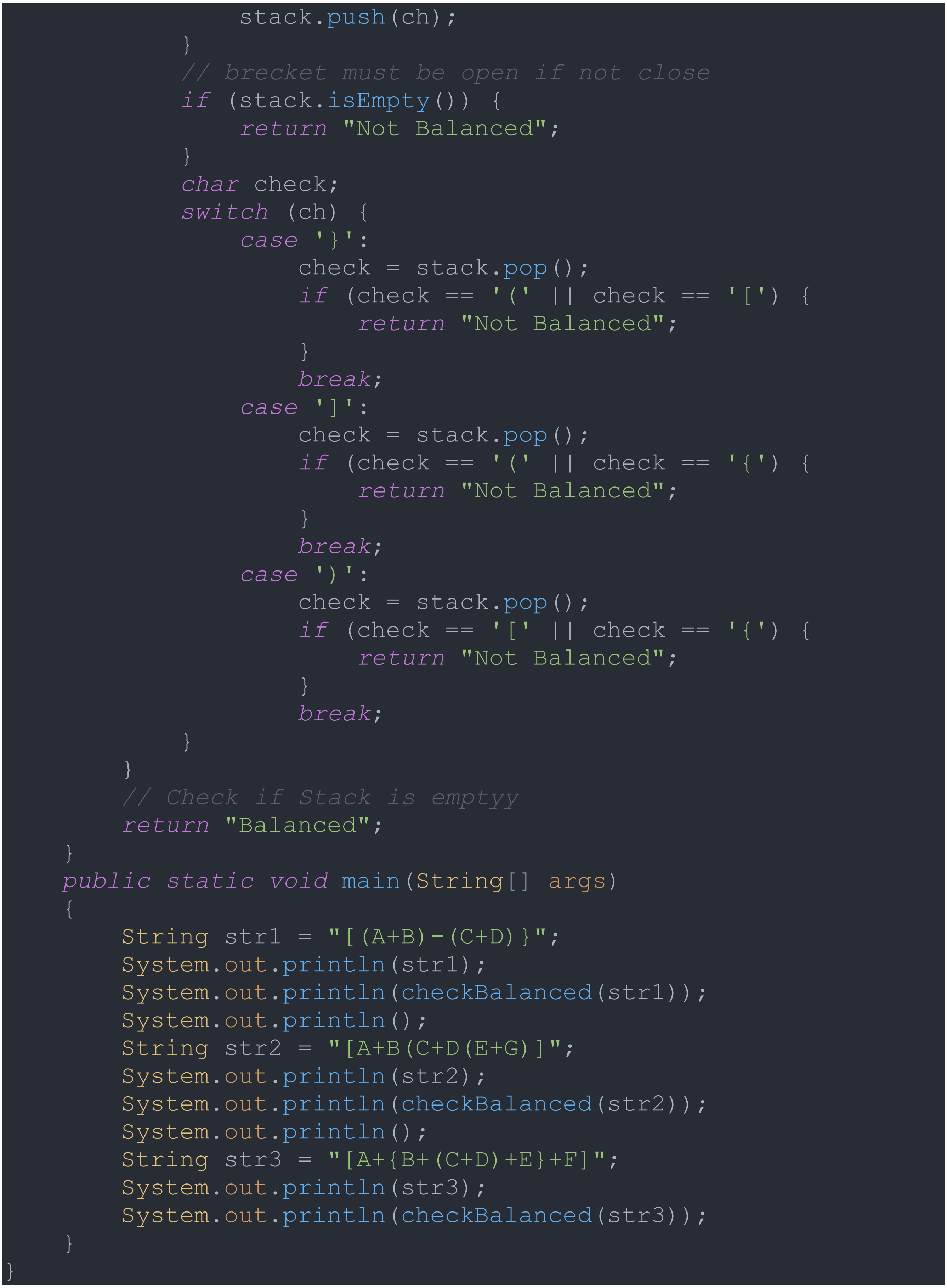
```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-java
Enter the number you want to multiply :
10
20->10->40->30->50->END

Process finished with exit code 0
```

4. For any mathematical equation to run successfully, the parenthesis plays a vital role in its solution. For long equations, there are the chances of the misplaced brackets or missing brackets. Being a futuristic potential engineer, design a parenthesis balance checking program. For eg.: [(A+B)-(C+D)} ◊ Unbalanced [A+B(C+D(E+G)] ◊ Unbalanced [A+{B+(C+D)+E}+F]

Code :

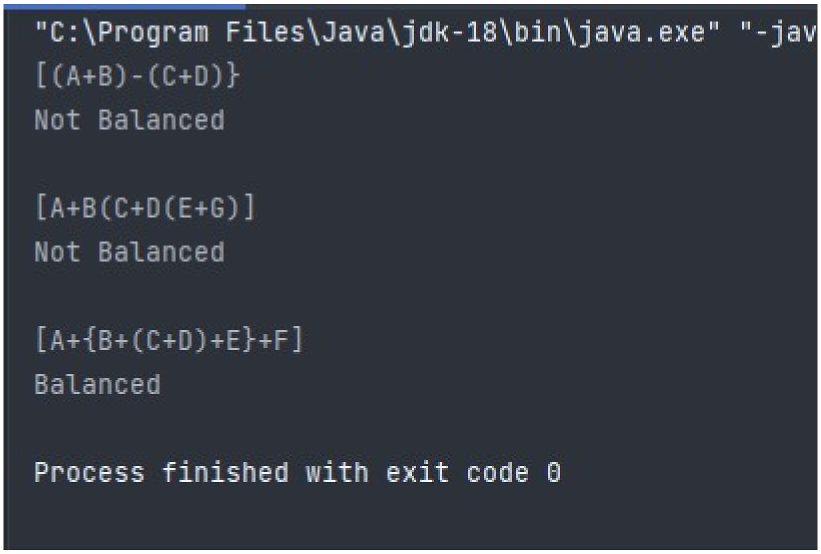


```
package com.DSA.LAB8;
import java.util.*;
public class CheckBalanced {
    //check if brackets are balanced
    static String checkBalanced(String str)
    {
        Stack<Character> stack = new Stack<>();
        // Traversing the Expression
        for (int i = 0; i < str.length(); i++)
        {
            char ch = str.charAt(i);
            if (ch == '{' || ch == '(' || ch == '[')
            {
                // Push the element in the stack
```



```
                stack.push(ch);
            }
            // brecket must be open if not close
            if (stack.isEmpty()) {
                return "Not Balanced";
            }
            char check;
            switch (ch) {
                case '}':
                    check = stack.pop();
                    if (check == '(' || check == '[') {
                        return "Not Balanced";
                    }
                    break;
                case ']':
                    check = stack.pop();
                    if (check == '(' || check == '{') {
                        return "Not Balanced";
                    }
                    break;
                case ')':
                    check = stack.pop();
                    if (check == '[' || check == '{') {
                        return "Not Balanced";
                    }
                    break;
            }
        }
        // Check if Stack is emptyy
        return "Balanced";
    }
    public static void main(String[] args)
    {
        String str1 = "[(A+B)-(C+D)}";
        System.out.println(str1);
        System.out.println(checkBalanced(str1));
        System.out.println();
        String str2 = "[A+B(C+D(E+G)]";
        System.out.println(str2);
        System.out.println(checkBalanced(str2));
        System.out.println();
        String str3 = "[A+{B+(C+D)+E}+F]";
        System.out.println(str3);
        System.out.println(checkBalanced(str3));
    }
}
```

Output :



```
"C:\Program Files\Java\jdk-18\bin\java.exe" "-jav
[(A+B)-(C+D)}
Not Balanced

[A+B(C+D(E+G)]
Not Balanced

[A+{B+(C+D)+E}+F]
Balanced

Process finished with exit code 0
```

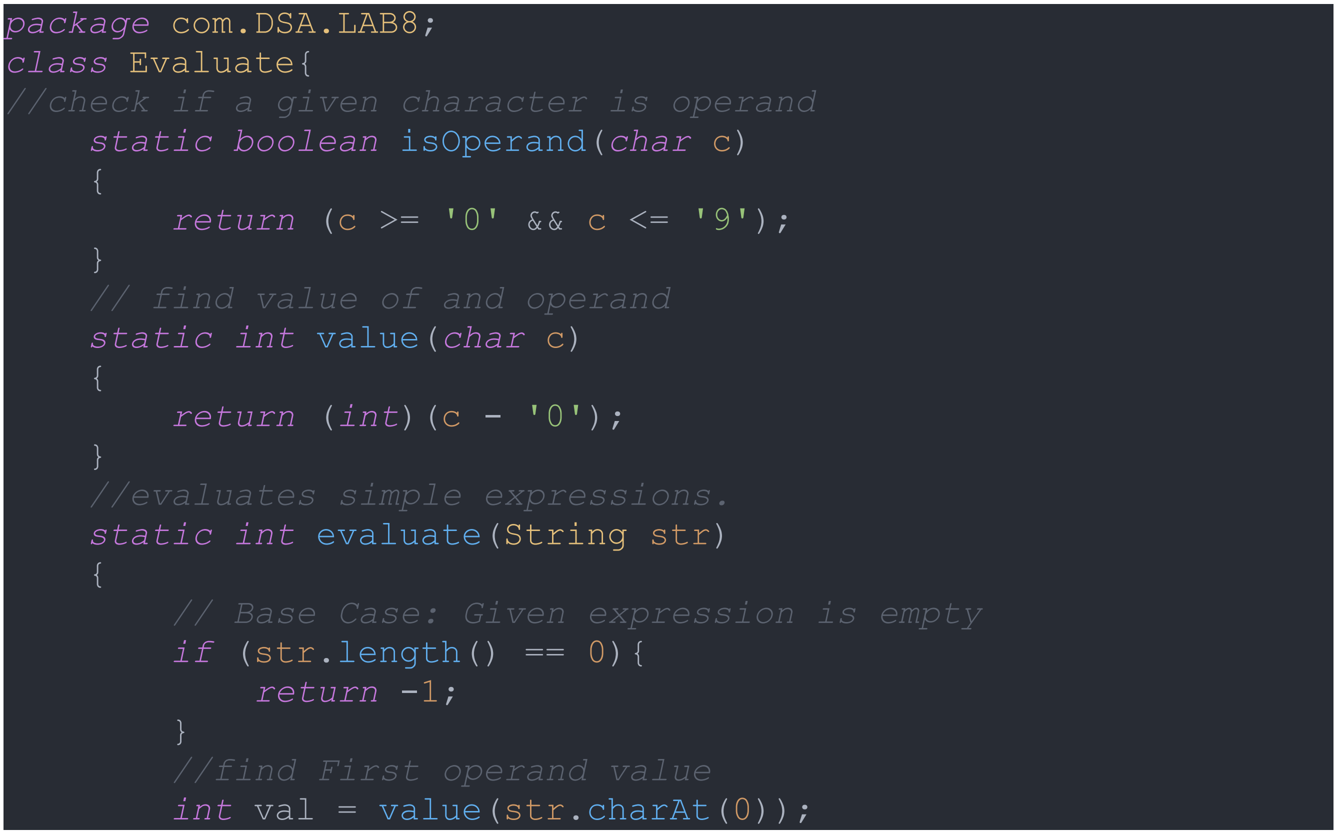
Q.5 Accept the evaluation formula string from user and evaluate the formula using stack.

For eg.: Input: (1+(2*3)-5)

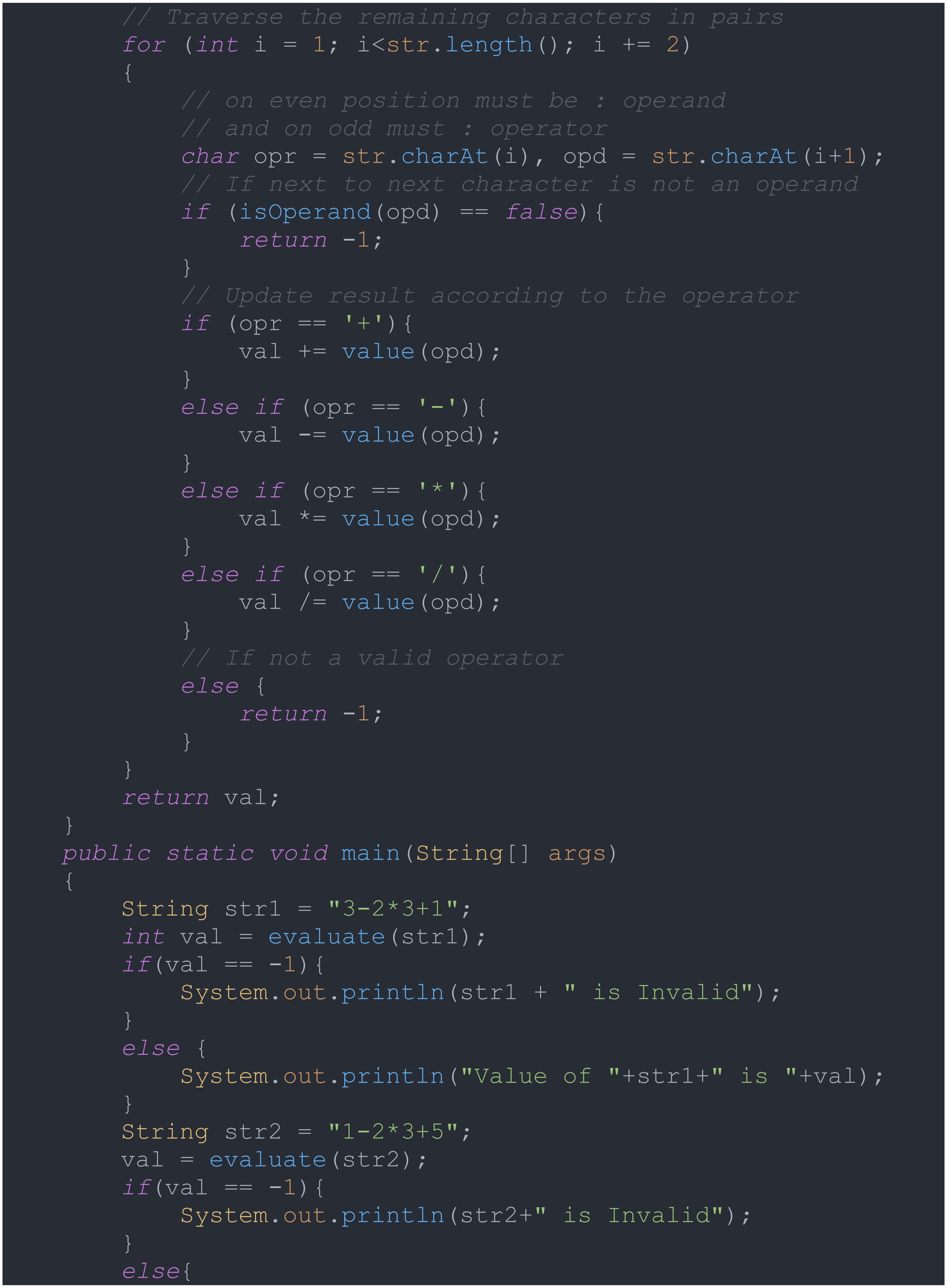
Output: 2

Hint: You may use infix or postfix expression for the solution

Code :

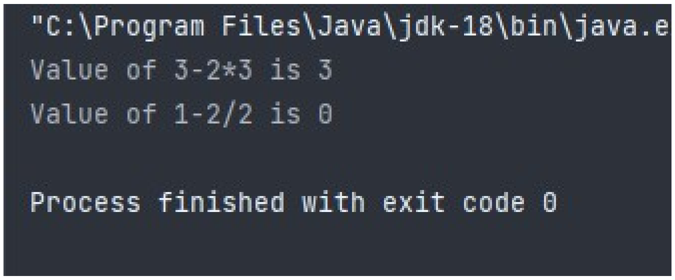


```java
package com.DSA.LAB8;
class Evaluate{
//check if a given character is operand
    static boolean isOperand(char c)
    {
        return (c >= '0' && c <= '9');
    }
    // find value of and operand
    static int value(char c)
    {
        return (int)(c - '0');
    }
    //evaluates simple expressions.
    static int evaluate(String str)
    {
        // Base Case: Given expression is empty
        if (str.length() == 0){
            return -1;
        }
        //find First operand value
        int val = value(str.charAt(0));
```

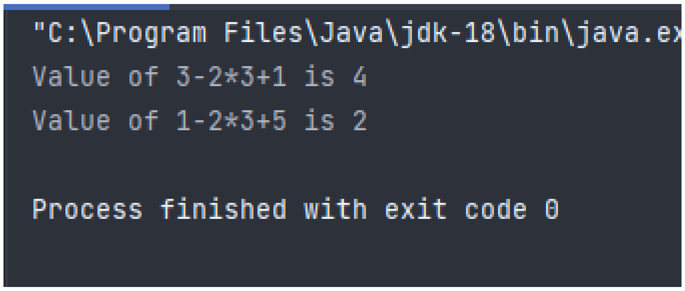


```
        // Traverse the remaining characters in pairs
        for (int i = 1; i<str.length(); i += 2)
        {
            // on even position must be : operand
            // and on odd must : operator
            char opr = str.charAt(i), opd = str.charAt(i+1);
            // If next to next character is not an operand
            if (isOperand(opd) == false){
                return -1;
            }
            // Update result according to the operator
            if (opr == '+'){
                val += value(opd);
            }
            else if (opr == '-'){
                val -= value(opd);
            }
            else if (opr == '*'){
                val *= value(opd);
            }
            else if (opr == '/'){
                val /= value(opd);
            }
            // If not a valid operator
            else {
                return -1;
            }
        }
        return val;
    }
    public static void main(String[] args)
    {
        String str1 = "3-2*3+1";
        int val = evaluate(str1);
        if(val == -1){
            System.out.println(str1 + " is Invalid");
        }
        else {
            System.out.println("Value of "+str1+" is "+val);
        }
        String str2 = "1-2*3+5";
        val = evaluate(str2);
        if(val == -1){
            System.out.println(str2+" is Invalid");
        }
        else{
            System.out.println("Value of " + str2+" is "+val);
        }
    }
}
```

Output :



```
"C:\Program Files\Java\jdk-18\bin\java.e
Value of 3-2*3 is 3
Value of 1-2/2 is 0

Process finished with exit code 0
```



```
"C:\Program Files\Java\jdk-18\bin\java.ex
Value of 3-2*3+1 is 4
Value of 1-2*3+5 is 2

Process finished with exit code 0
```