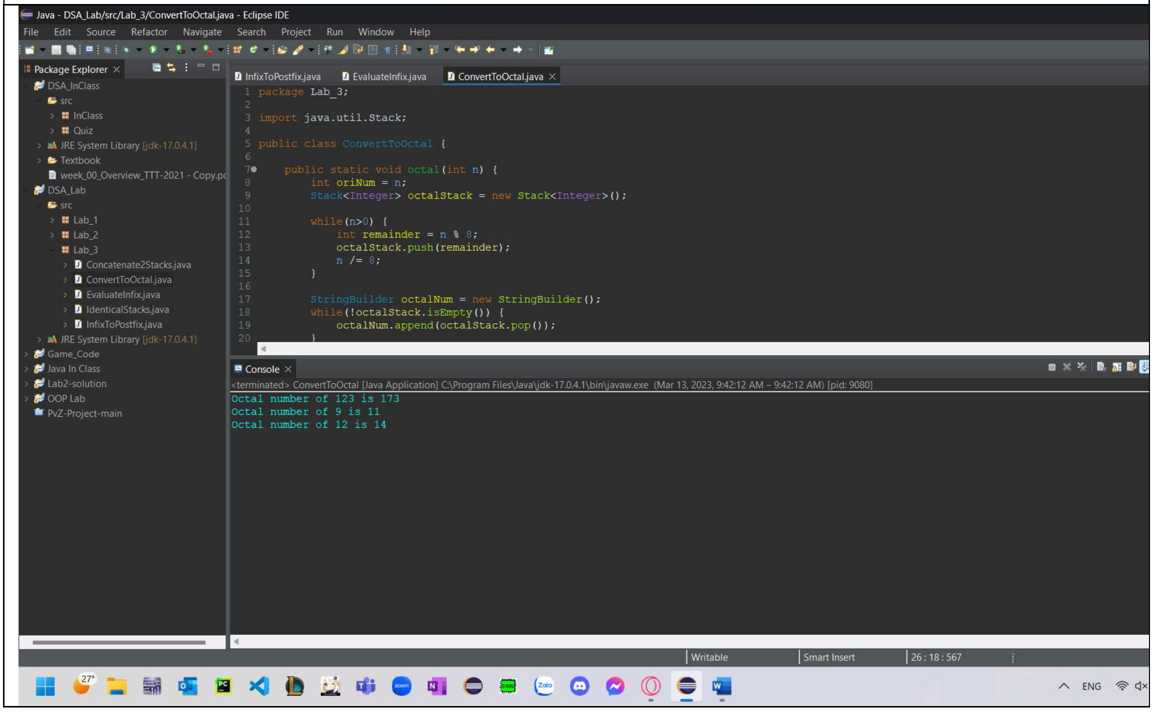
NGUYỄN MINH ĐỨC – ITITIU21045

Problem 1

i. Convert a decimal number to its octal form.

```java
package Lab_3;

import java.util.Stack;

public class ConvertToOctal {

	public static void octal(int n) {
		int oriNum = n;
		Stack<Integer> octalStack = new Stack<Integer>();
		
		while(n>0) {
			int remainder = n % 8;
			octalStack.push(remainder);
			n /= 8;
		}
		
		StringBuilder octalNum = new StringBuilder();
		while(!octalStack.isEmpty()) {
			octalNum.append(octalStack.pop());
		}
		
		System.out.println("Octal number of "+oriNum+" is 
"+octalNum.toString());
	}
	public static void main(String[] args) {
		octal(123);
		octal(9);
		octal(12);
		
	}

}
```

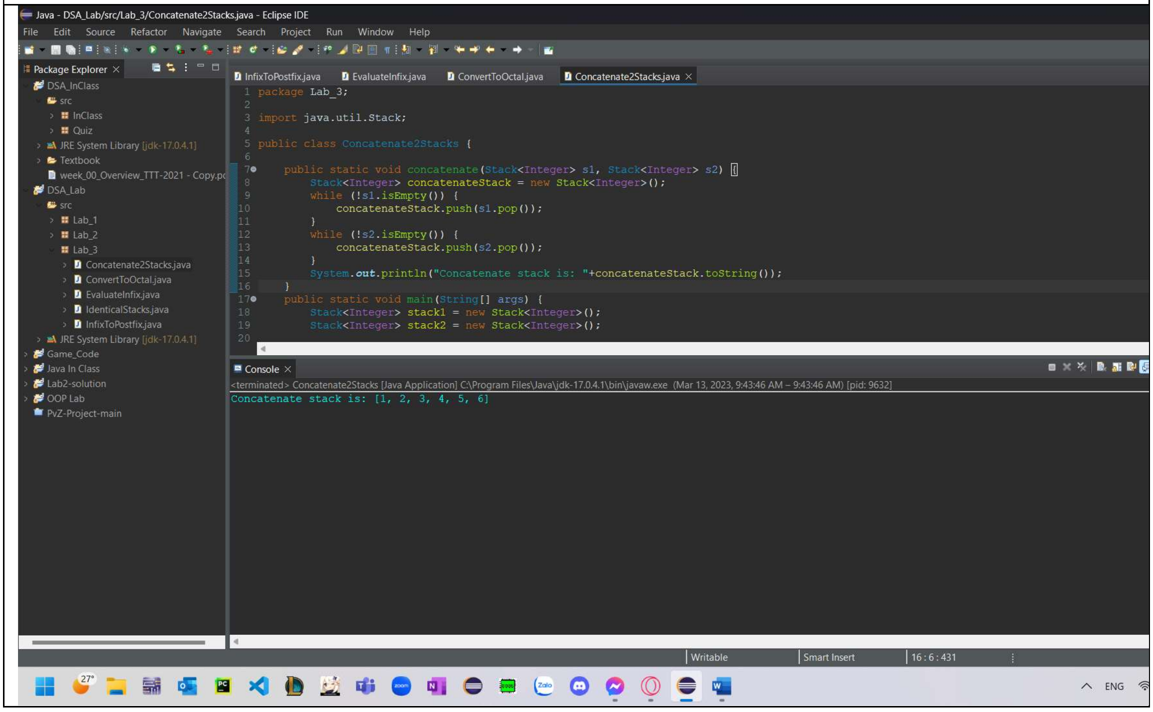

Java - DSA_Lab/src/Lab_3/ConvertToOctal.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
InfixToPostfix.java
EvaluateInfix.java
ConvertToOctal.java
Console
Writable
Smart Insert
26 : 18 : 567

```java
package Lab_3;

import java.util.Stack;

public class ConvertToOctal {

    public static void octal(int n) {
        int oriNum = n;
        Stack<Integer> octalStack = new Stack<Integer>();

        while(n>0) {
            int remainder = n % 8;
            octalStack.push(remainder);
            n /= 8;
        }

        StringBuilder octalNum = new StringBuilder();
        while(!octalStack.isEmpty()) {
            octalNum.append(octalStack.pop());
        }
```

```
<terminated> ConvertToOctal [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe  (Mar 13, 2023, 9:42:12 AM – 9:42:12 AM) [pid: 9080]
Octal number of 123 is 173
Octal number of 9 is 11
Octal number of 12 is 14
```

ii. Concatenate two stacks.

```
package Lab_3;

import java.util.Stack;

public class Concatenate2Stacks {

	public static void concatenate(Stack<Integer> s1, 
Stack<Integer> s2) {
		Stack<Integer> concatenateStack = new Stack<Integer>();
		while (!s1.isEmpty()) {
			concatenateStack.push(s1.pop());
		}
		while (!s2.isEmpty()) {
			concatenateStack.push(s2.pop());
		}
		System.out.println("Concatenate stack is: 
"+concatenateStack.toString());
	}
	public static void main(String[] args) {
		Stack<Integer> stack1 = new Stack<Integer>();
		Stack<Integer> stack2 = new Stack<Integer>();

		stack1.push(3);
		stack1.push(2);
		stack1.push(1);

		stack2.push(6);
		stack2.push(5);
		stack2.push(4);

		concatenate(stack1, stack2);
	}

}
```

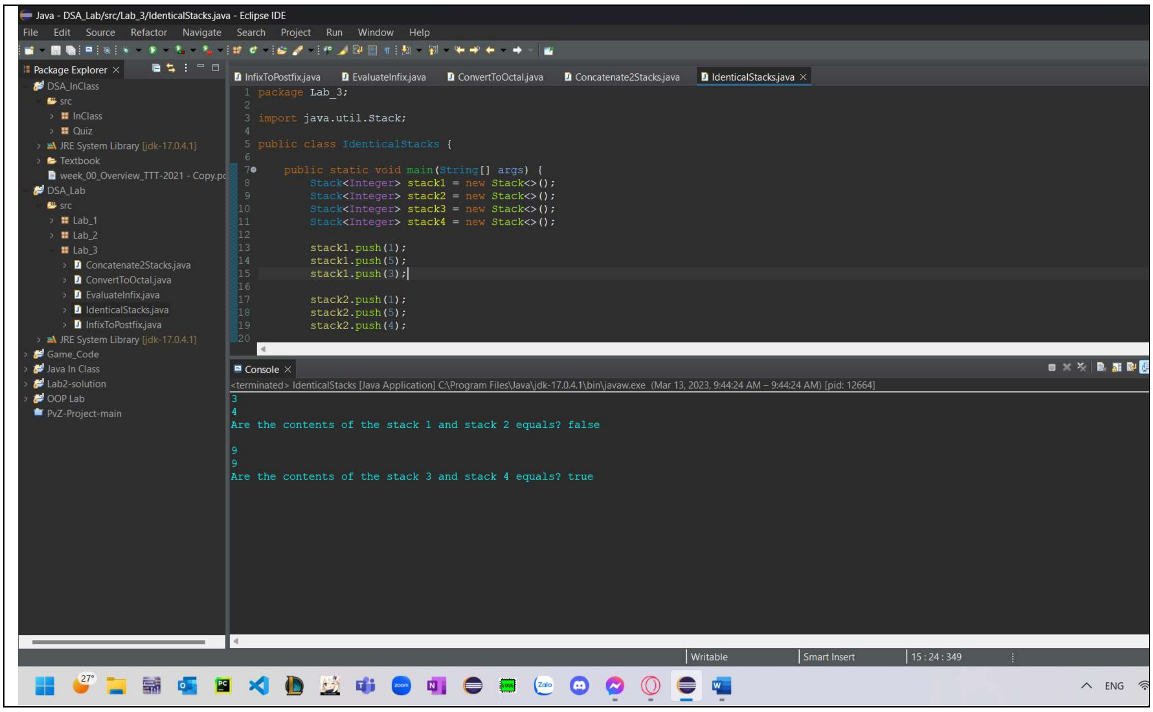


```
package Lab_3;

import java.util.Stack;

public class Concatenate2Stacks {

    public static void concatenate(Stack<Integer> s1, Stack<Integer> s2) {
        Stack<Integer> concatenateStack = new Stack<Integer>();
        while (!s1.isEmpty()) {
            concatenateStack.push(s1.pop());
        }
        while (!s2.isEmpty()) {
            concatenateStack.push(s2.pop());
        }
        System.out.println("Concatenate stack is: "+concatenateStack.toString());
    }
    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<Integer>();
        Stack<Integer> stack2 = new Stack<Integer>();

```

```
<terminated> Concatenate2Stacks [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe  (Mar 13, 2023, 9:43:46 AM – 9:43:46 AM) [pid: 9632]
Concatenate stack is: [1, 2, 3, 4, 5, 6]
```

iii. Determine if the contents of two stack are identical.

```
package Lab_3;

import java.util.Stack;

public class IdenticalStacks {

        public static void main(String[] args) {
                Stack<Integer> stack1 = new Stack<>();
                Stack<Integer> stack2 = new Stack<>();
                Stack<Integer> stack3 = new Stack<>();
                Stack<Integer> stack4 = new Stack<>();

                stack1.push(1);
                stack1.push(5);
                stack1.push(3);

                stack2.push(1);
                stack2.push(5);
                stack2.push(4);

                stack3.push(2);
                stack3.push(6);
                stack3.push(9);

                stack4.push(2);
                stack4.push(6);
                stack4.push(9);

                boolean equal1 = stack1.equals(stack2);
                boolean equal2 = stack3.equals(stack4);

                System.out.println(stack1.peek());
                System.out.println(stack2.peek());
                System.out.println("Are the contents of the stack 1 and
stack 2 equals? "+equal1);

                System.out.println();

                System.out.println(stack3.peek());
                System.out.println(stack4.peek());
                System.out.println("Are the contents of the stack 3 and
stack 4 equals? "+equal2);
        }

}
```



```java
package Lab_3;

import java.util.Stack;

public class IdenticalStacks {

    public static void main(String[] args) {
        Stack<Integer> stack1 = new Stack<>();
        Stack<Integer> stack2 = new Stack<>();
        Stack<Integer> stack3 = new Stack<>();
        Stack<Integer> stack4 = new Stack<>();

        stack1.push(1);
        stack1.push(5);
        stack1.push(3);

        stack2.push(1);
        stack2.push(5);
        stack2.push(4);

```

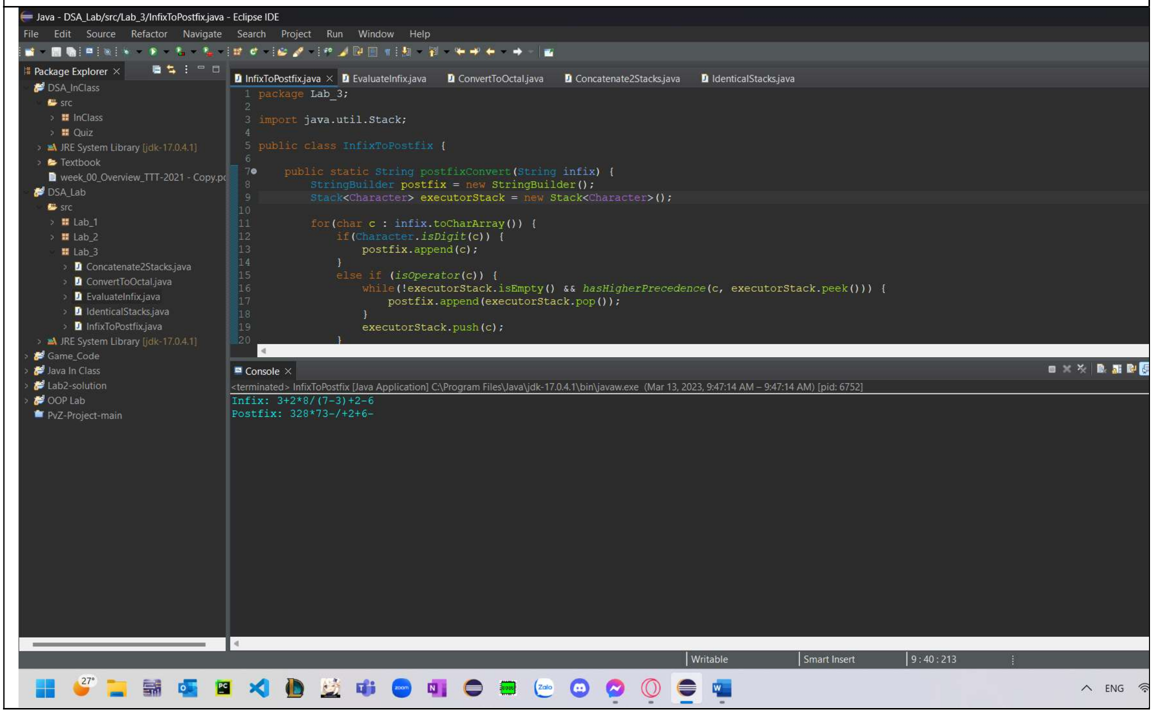
```
<terminated> IdenticalStacks [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe  (Mar 13, 2023, 9:44:24 AM – 9:44:24 AM) [pid: 12664]
3
4
Are the contents of the stack 1 and stack 2 equals? false

9
9
Are the contents of the stack 3 and stack 4 equals? true
```

Problem 2

Infix to Postfix

```java
package Lab_3;

import java.util.Stack;

public class InfixToPostfix {

    public static String postfixConvert(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> executorStack = new Stack<Character>();

        for(char c : infix.toCharArray()) {
            if(Character.isDigit(c)) {
                postfix.append(c);
            }
            else if (isOperator(c)) {
                while(!executorStack.isEmpty() &&
hasHigherPrecedence(c, executorStack.peek())) {
                    postfix.append(executorStack.pop());
                }
                executorStack.push(c);
            }
            else if (c == '(') {
                executorStack.push(c);
            }
            else if (c == ')') {
                while(!executorStack.isEmpty() &&
executorStack.peek() != '(') {
                    postfix.append(executorStack.pop());
                }
                executorStack.pop();
            }
        }

        while(!executorStack.isEmpty()) {
            postfix.append(executorStack.pop());
        }
```

```
            return postfix.toString();

        }

        private static boolean isOperator(char c) {
            return c == '+' || c == '-' || c == '*' || c == '/';
        }

        private static boolean hasHigherPrecedence(char op1, char op2)
 {
            if (op2 == '(' || op2 == ')') {
                return false;
            }
            else if ((op1 == '*' || op1 == '/') && (op2 == '+' || op2
 == '-')) {
                return false;
            }
            else {
                return true;
            }
        }

        public static void main(String[] args) {
            String infix = "3+2*8/(7-3)+2-6";
            String postfix = postfixConvert(infix);
            System.out.println("Infix: "+infix);
            System.out.println("Postfix: "+postfix);

        }

 }
```



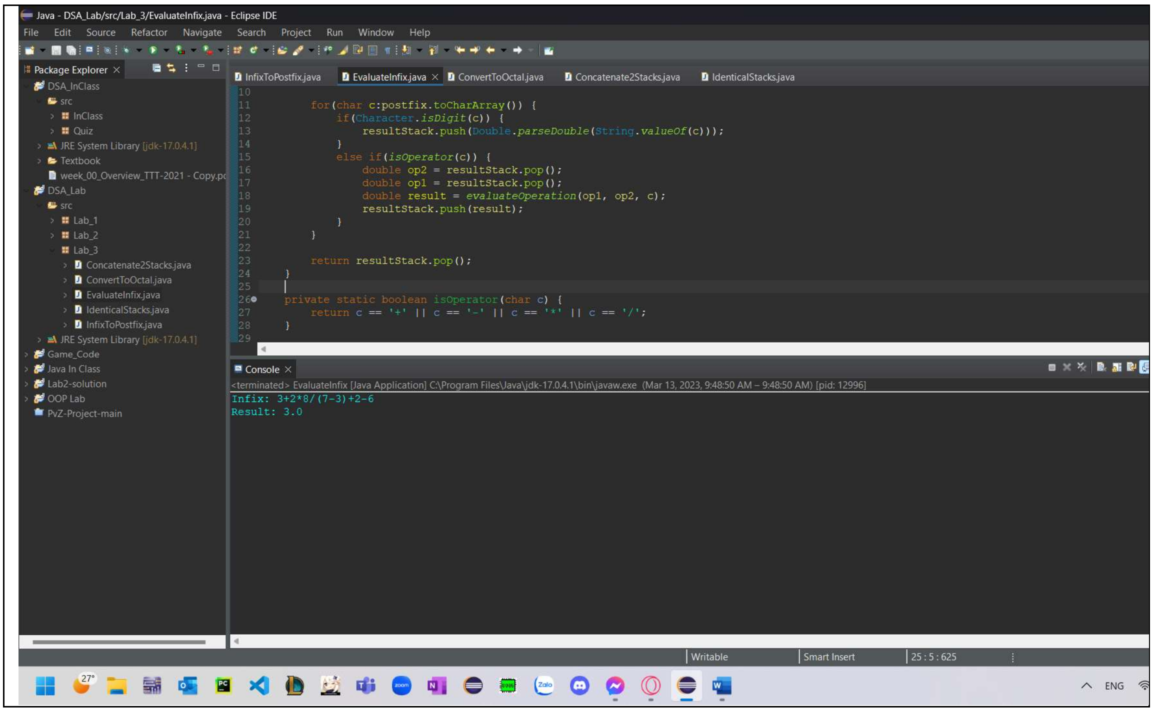
```
package Lab_3;

import java.util.Stack;

public class InfixToPostfix {

    public static String postfixConvert(String infix) {
        StringBuilder postfix = new StringBuilder();
        Stack<Character> executorStack = new Stack<Character>();

        for(char c : infix.toCharArray()) {
            if(Character.isDigit(c)) {
                postfix.append(c);
            }
            else if (isOperator(c)) {
                while(!executorStack.isEmpty() && hasHigherPrecedence(c, executorStack.peek())) {
                    postfix.append(executorStack.pop());
                }
                executorStack.push(c);
            }
```

```
<terminated> InfixToPostfix [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe  (Mar 13, 2023, 9:47:14 AM – 9:47:14 AM) [pid: 6752]
Infix: 3+2*8/(7-3)+2-6
Postfix: 328*73-/+2+6-
```

Evaluate Infix

```
package Lab_3;

import java.util.Stack;

public class EvaluateInfix extends InfixToPostfix{

      public static double evaluateInfix(String infix) {
            Stack<Double> resultStack = new Stack<Double>();
            String postfix = postfixConvert(infix);

            for(char c:postfix.toCharArray()) {
                  if(Character.isDigit(c)) {
      resultStack.push(Double.parseDouble(String.valueOf(c)));
                  }
                  else if(isOperator(c)) {
                        double op2 = resultStack.pop();
                        double op1 = resultStack.pop();
                        double result = evaluateOperation(op1, op2, 
c);
                        resultStack.push(result);
                  }
            }

            return resultStack.pop();
      }

      private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static double evaluateOperation(double op1, double op2, 
char operator) {
        switch (operator) {
            case '+':
                return op1 + op2;
            case '-':
                return op1 - op2;
            case '*':
                return op1 * op2;
            case '/':
                return op1 / op2;
            default:
                throw new IllegalArgumentException("Invalid operator: 
" + operator);
        }
    }

      public static void main(String[] args) {
            String infix = "3+2*8/(7-3)+2-6";
            double result = evaluateInfix(infix);
            System.out.println("Infix: "+infix);
            System.out.println("Result: "+result);
      }

}
```

NGUYỄN MINH ĐỨC – ITITIU21045

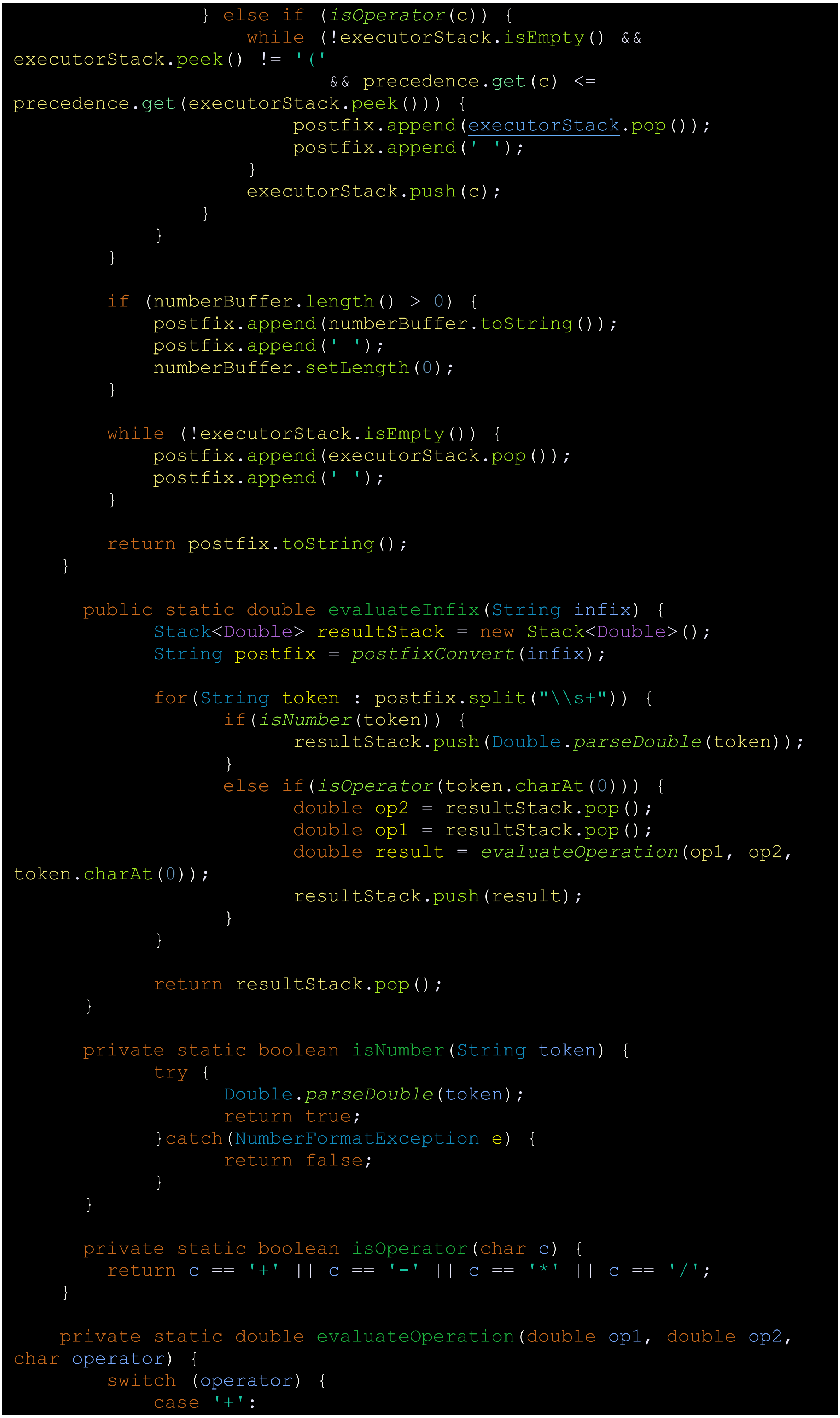


```java

        for(char c:postfix.toCharArray()) {
            if(Character.isDigit(c)) {
                resultStack.push(Double.parseDouble(String.valueOf(c)));
            }
            else if(isOperator(c)) {
                double op2 = resultStack.pop();
                double op1 = resultStack.pop();
                double result = evaluateOperation(op1, op2, c);
                resultStack.push(result);
            }
        }

        return resultStack.pop();
    }

    private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

```

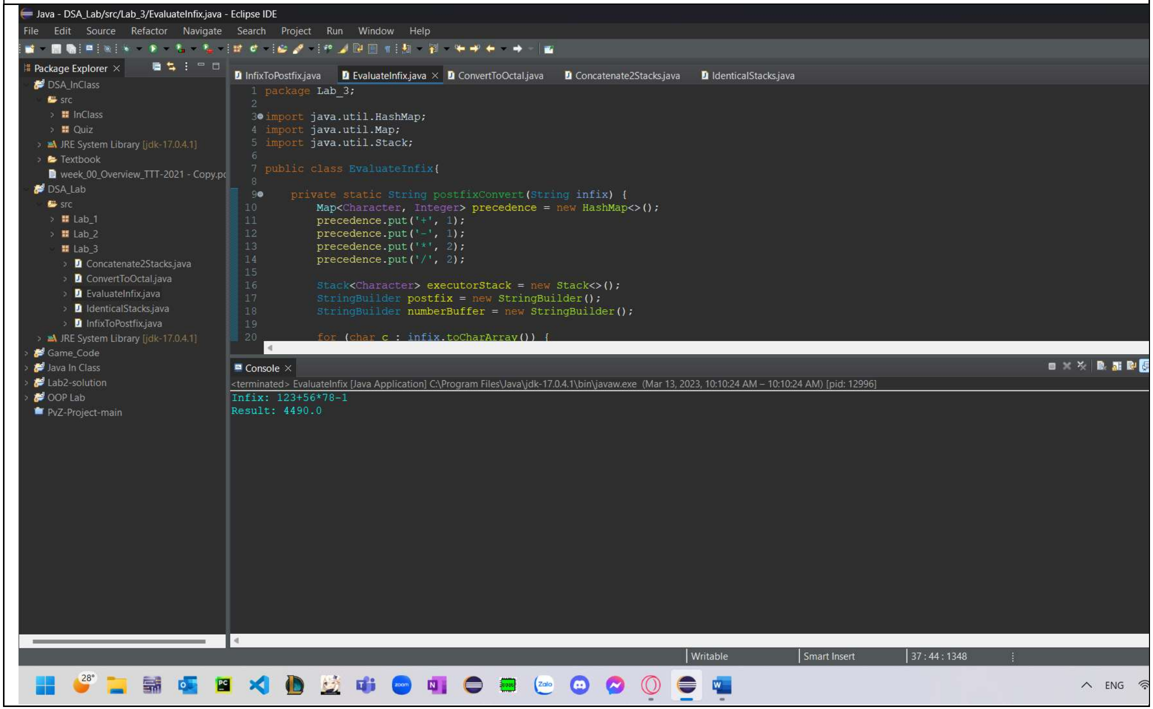
```
<terminated> EvaluateInfix [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe (Mar 13, 2023, 9:48:50 AM – 9:48:50 AM) [pid: 12996]
Infix: 3+2*8/(7-3)+2-6
Result: 3.0
```

Evaluate Multiple Digits Infix

```java
package Lab_3;

import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

public class EvaluateInfix{

      private static String postfixConvert(String infix) {
            Map<Character, Integer> precedence = new HashMap<>();
        precedence.put('+', 1);
        precedence.put('-', 1);
        precedence.put('*', 2);
        precedence.put('/', 2);

        Stack<Character> executorStack = new Stack<>();
        StringBuilder postfix = new StringBuilder();
        StringBuilder numberBuffer = new StringBuilder();

        for (char c : infix.toCharArray()) {
            if (Character.isDigit(c)) {
                numberBuffer.append(c);
            } else {
                if (numberBuffer.length() > 0) {
                    postfix.append(numberBuffer.toString());
                    postfix.append(' ');
                    numberBuffer.setLength(0);
                }
                if (c == '(') {
                  executorStack.push(c);
                } else if (c == ')') {
                    while (!executorStack.isEmpty() &&
executorStack.peek() != '(') {
                        postfix.append(executorStack.pop());
                        postfix.append(' ');
                    }
                    executorStack.pop();
```



```
                } else if (isOperator(c)) {
                    while (!executorStack.isEmpty() &&
executorStack.peek() != '('
                           && precedence.get(c) <=
precedence.get(executorStack.peek())) {
                        postfix.append(executorStack.pop());
                        postfix.append(' ');
                    }
                    executorStack.push(c);
                }
            }
        }

        if (numberBuffer.length() > 0) {
            postfix.append(numberBuffer.toString());
            postfix.append(' ');
            numberBuffer.setLength(0);
        }

        while (!executorStack.isEmpty()) {
            postfix.append(executorStack.pop());
            postfix.append(' ');
        }

        return postfix.toString();
    }

      public static double evaluateInfix(String infix) {
            Stack<Double> resultStack = new Stack<Double>();
            String postfix = postfixConvert(infix);

            for(String token : postfix.split("\\s+")) {
                  if(isNumber(token)) {
                        resultStack.push(Double.parseDouble(token));
                  }
                  else if(isOperator(token.charAt(0))) {
                        double op2 = resultStack.pop();
                        double op1 = resultStack.pop();
                        double result = evaluateOperation(op1, op2,
token.charAt(0));
                        resultStack.push(result);
                  }
            }

            return resultStack.pop();
      }

      private static boolean isNumber(String token) {
            try {
                  Double.parseDouble(token);
                  return true;
            }catch(NumberFormatException e) {
                  return false;
            }
      }

      private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static double evaluateOperation(double op1, double op2,
char operator) {
        switch (operator) {
            case '+':
                return op1 + op2;
            case '-':
                return op1 - op2;
            case '*':
                return op1 * op2;
            case '/':
                return op1 / op2;
            default:
                throw new IllegalArgumentException("Invalid operator: 
" + operator);
        }
    }

    public static void main(String[] args) {
        String infix = "123+56*78-1";
        double result = evaluateInfix(infix);
        System.out.println("Infix: "+infix);
        System.out.println("Result: "+result);
    }

}
```

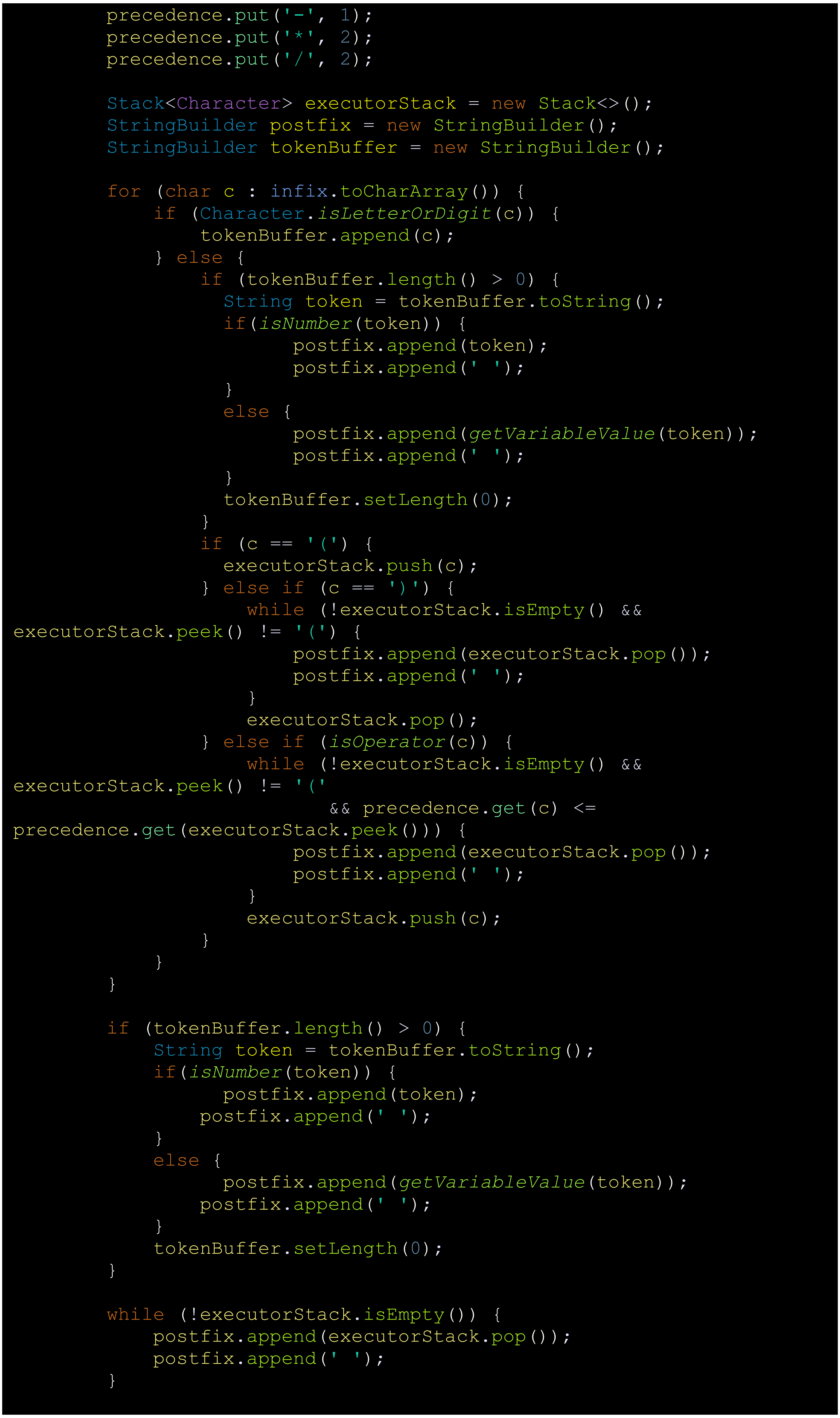


```
package Lab_3;

import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

public class EvaluateInfix{

    private static String postfixConvert(String infix) {
        Map<Character, Integer> precedence = new HashMap<>();
        precedence.put('+', 1);
        precedence.put('-', 1);
        precedence.put('*', 2);
        precedence.put('/', 2);

        Stack<Character> executorStack = new Stack<>();
        StringBuilder postfix = new StringBuilder();
        StringBuilder numberBuffer = new StringBuilder();

        for (char c : infix.toCharArray()) {
```

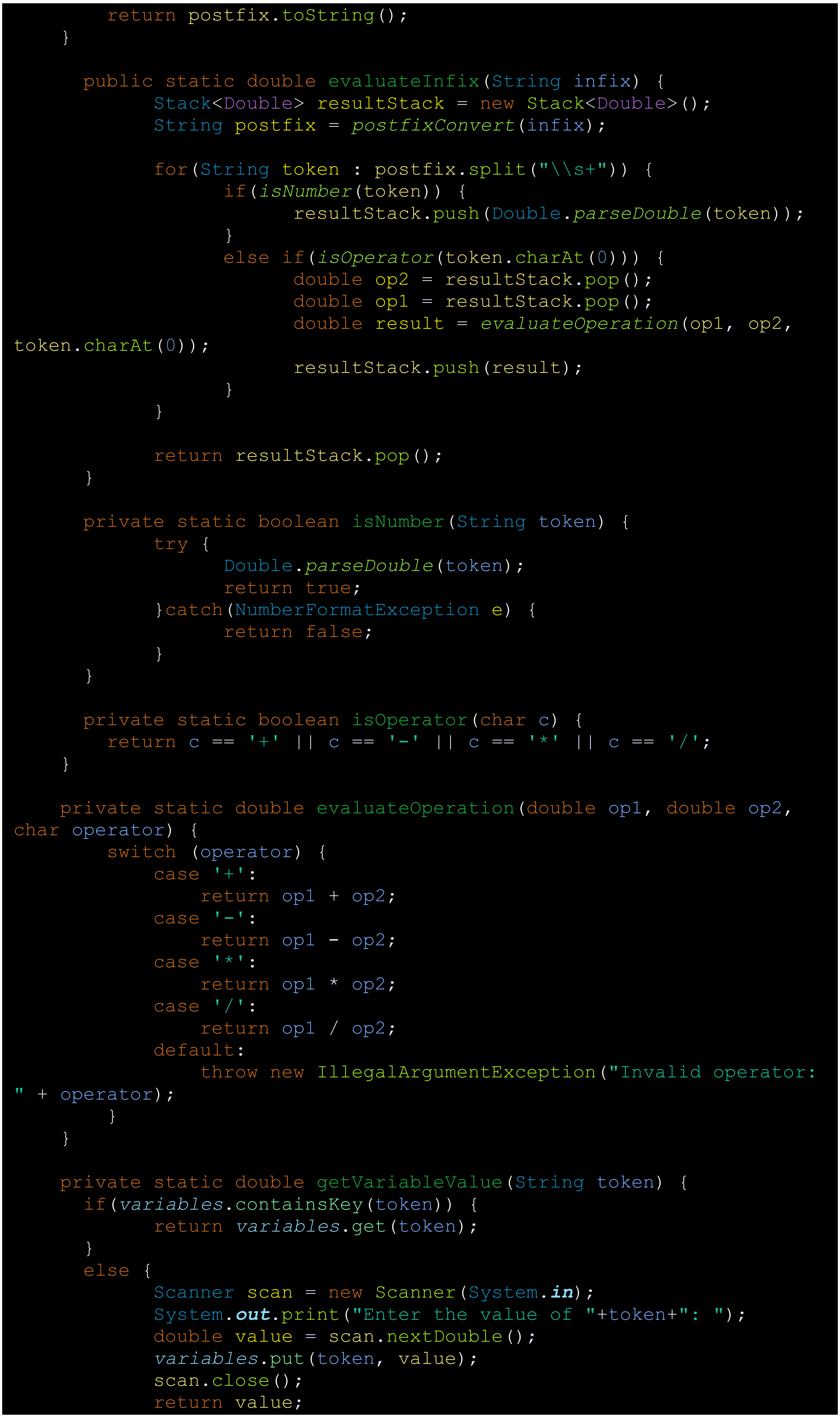
```
<terminated> EvaluateInfix [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe (Mar 13, 2023, 10:10:24 AM – 10:10:24 AM) [pid: 12996]
Infix: 123+56*78-1
Result: 4490.0
```

Evaluate Infix With Variables

```
package Lab_3;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Stack;

public class EvaluateInfix{

    private static Map<String, Double> variables = new HashMap<>();

    private static String postfixConvert(String infix) {
        Map<Character, Integer> precedence = new HashMap<>();
     precedence.put('+', 1);
```

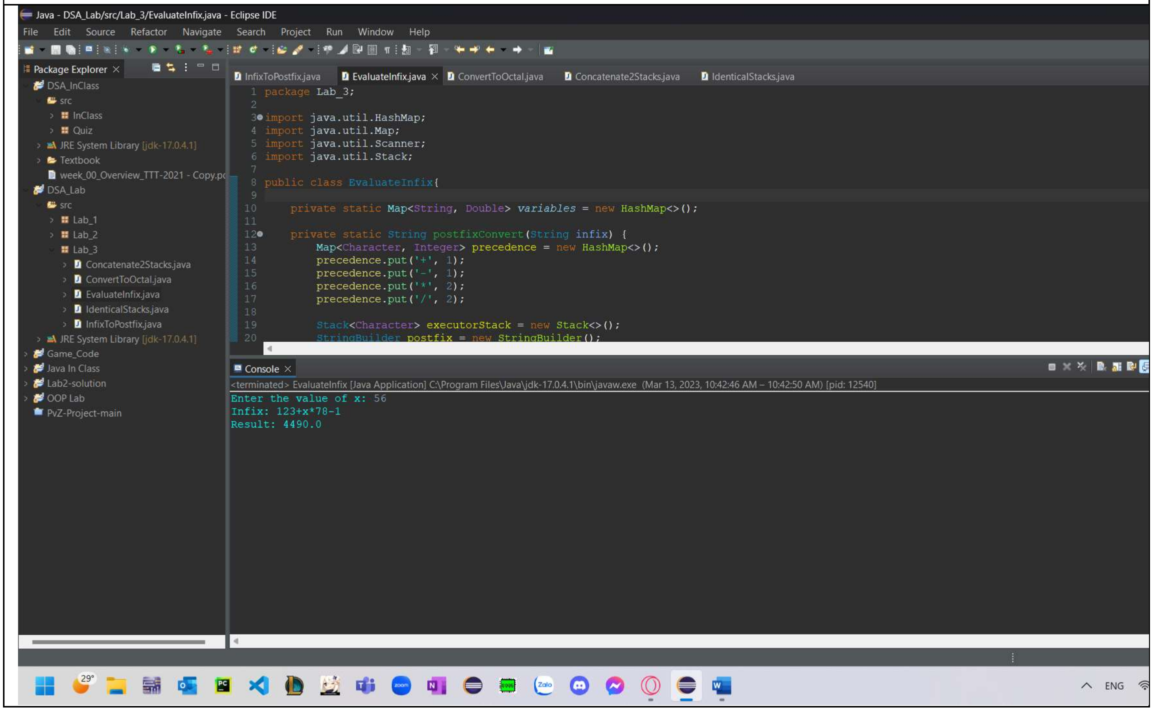


```java
        precedence.put('-', 1);
        precedence.put('*', 2);
        precedence.put('/', 2);

        Stack<Character> executorStack = new Stack<>();
        StringBuilder postfix = new StringBuilder();
        StringBuilder tokenBuffer = new StringBuilder();

        for (char c : infix.toCharArray()) {
            if (Character.isLetterOrDigit(c)) {
                tokenBuffer.append(c);
            } else {
                if (tokenBuffer.length() > 0) {
                  String token = tokenBuffer.toString();
                  if(isNumber(token)) {
                        postfix.append(token);
                        postfix.append(' ');
                  }
                  else {
                        postfix.append(getVariableValue(token));
                        postfix.append(' ');
                  }
                  tokenBuffer.setLength(0);
                }
                if (c == '(') {
                  executorStack.push(c);
                } else if (c == ')') {
                    while (!executorStack.isEmpty() &&
executorStack.peek() != '(') {
                        postfix.append(executorStack.pop());
                        postfix.append(' ');
                    }
                    executorStack.pop();
                } else if (isOperator(c)) {
                    while (!executorStack.isEmpty() &&
executorStack.peek() != '('
                           && precedence.get(c) <=
precedence.get(executorStack.peek())) {
                        postfix.append(executorStack.pop());
                        postfix.append(' ');
                    }
                    executorStack.push(c);
                }
            }
        }

        if (tokenBuffer.length() > 0) {
            String token = tokenBuffer.toString();
            if(isNumber(token)) {
                  postfix.append(token);
                postfix.append(' ');
            }
            else {
                  postfix.append(getVariableValue(token));
                postfix.append(' ');
            }
            tokenBuffer.setLength(0);
        }

        while (!executorStack.isEmpty()) {
            postfix.append(executorStack.pop());
            postfix.append(' ');
        }
```



```
        return postfix.toString();
    }

      public static double evaluateInfix(String infix) {
            Stack<Double> resultStack = new Stack<Double>();
            String postfix = postfixConvert(infix);

            for(String token : postfix.split("\\s+")) {
                  if(isNumber(token)) {
                        resultStack.push(Double.parseDouble(token));
                  }
                  else if(isOperator(token.charAt(0))) {
                        double op2 = resultStack.pop();
                        double op1 = resultStack.pop();
                        double result = evaluateOperation(op1, op2,
token.charAt(0));
                        resultStack.push(result);
                  }
            }

            return resultStack.pop();
      }

      private static boolean isNumber(String token) {
            try {
                  Double.parseDouble(token);
                  return true;
            }catch(NumberFormatException e) {
                  return false;
            }
      }

      private static boolean isOperator(char c) {
        return c == '+' || c == '-' || c == '*' || c == '/';
    }

    private static double evaluateOperation(double op1, double op2,
char operator) {
        switch (operator) {
            case '+':
                return op1 + op2;
            case '-':
                return op1 - op2;
            case '*':
                return op1 * op2;
            case '/':
                return op1 / op2;
            default:
                throw new IllegalArgumentException("Invalid operator:
" + operator);
        }
    }

    private static double getVariableValue(String token) {
      if(variables.containsKey(token)) {
            return variables.get(token);
      }
      else {
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter the value of "+token+": ");
            double value = scan.nextDouble();
            variables.put(token, value);
            scan.close();
            return value;
```

```
		}
	}

	public static void main(String[] args) {
		String infix = "123+x*78-1";
		double result = evaluateInfix(infix);
		System.out.println("Infix: "+infix);
		System.out.println("Result: "+result);
	}

}
```

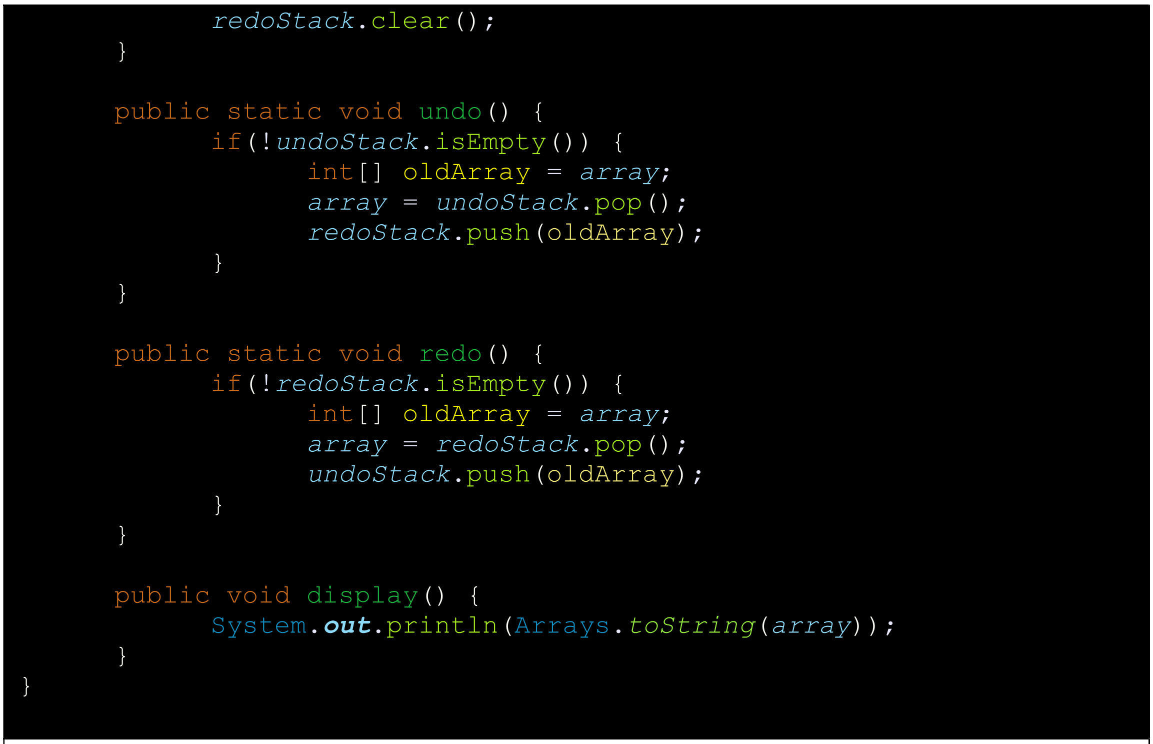


```
package Lab_3;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Stack;

public class EvaluateInfix{

    private static Map<String, Double> variables = new HashMap<>();

    private static String postfixConvert(String infix) {
        Map<Character, Integer> precedence = new HashMap<>();
        precedence.put('+', 1);
        precedence.put('-', 1);
        precedence.put('*', 2);
        precedence.put('/', 2);

        Stack<Character> executorStack = new Stack<>();
        StringBuilder postfix = new StringBuilder();
```

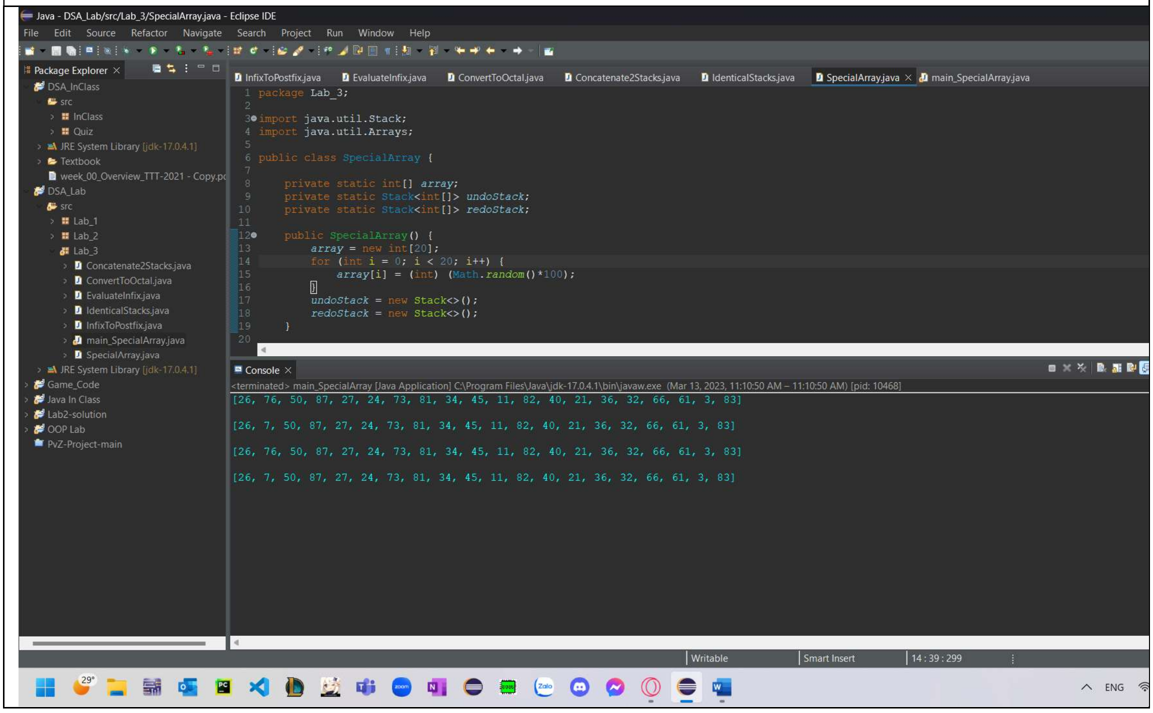
```
<terminated> EvaluateInfix [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe  (Mar 13, 2023, 10:42:46 AM – 10:42:50 AM) [pid: 12540]
Enter the value of x: 56
Infix: 123+x*78-1
Result: 4490.0
```

Problem 4

```
package Lab_3;

import java.util.Stack;
import java.util.Arrays;

public class SpecialArray {

	private static int[] array;
	private static Stack<int[]> undoStack;
	private static Stack<int[]> redoStack;

	public SpecialArray() {
		array = new int[20];
		for (int i = 0; i < 20; i++) {
			array[i] = (int) (Math.random()*100);
		}
		undoStack = new Stack<>();
		redoStack = new Stack<>();
	}

	public void ArrayUpdate(int index, int x) {
		int[] oldArray = array.clone();
		array[index] = x;
		undoStack.push(oldArray);
```



```
            redoStack.clear();
        }

        public static void undo() {
            if(!undoStack.isEmpty()) {
                int[] oldArray = array;
                array = undoStack.pop();
                redoStack.push(oldArray);
            }
        }

        public static void redo() {
            if(!redoStack.isEmpty()) {
                int[] oldArray = array;
                array = redoStack.pop();
                undoStack.push(oldArray);
            }
        }

        public void display() {
            System.out.println(Arrays.toString(array));
        }
}
```



```
package Lab_3;

import java.util.Stack;
import java.util.Arrays;

public class SpecialArray {

    private static int[] array;
    private static Stack<int[]> undoStack;
    private static Stack<int[]> redoStack;

    public SpecialArray() {
        array = new int[20];
        for (int i = 0; i < 20; i++) {
            array[i] = (int) (Math.random()*100);
        }
        undoStack = new Stack<>();
        redoStack = new Stack<>();
    }

```

```
<terminated> main_SpecialArray [Java Application] C:\Program Files\Java\jdk-17.0.4.1\bin\javaw.exe (Mar 13, 2023, 11:10:50 AM – 11:10:50 AM) [pid: 10468]
[26, 76, 50, 87, 27, 24, 73, 81, 34, 45, 11, 82, 40, 21, 36, 32, 66, 61, 3, 83]

[26, 7, 50, 87, 27, 24, 73, 81, 34, 45, 11, 82, 40, 21, 36, 32, 66, 61, 3, 83]

[26, 76, 50, 87, 27, 24, 73, 81, 34, 45, 11, 82, 40, 21, 36, 32, 66, 61, 3, 83]

[26, 7, 50, 87, 27, 24, 73, 81, 34, 45, 11, 82, 40, 21, 36, 32, 66, 61, 3, 83]
```