LAB 3 REPORT

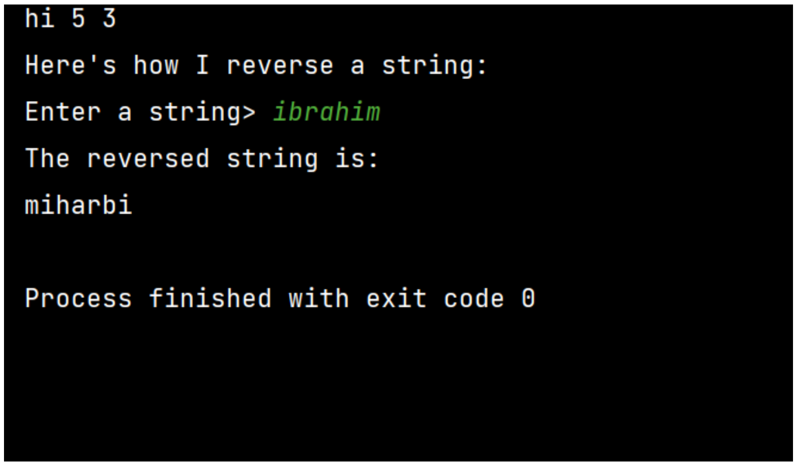
NAME: IBRAHIM ALSHAYEA

ID: 202176470

DATE: 2/17/2024

TASK 1: Nothing to do, just download and compile the program.

Task1 Output:

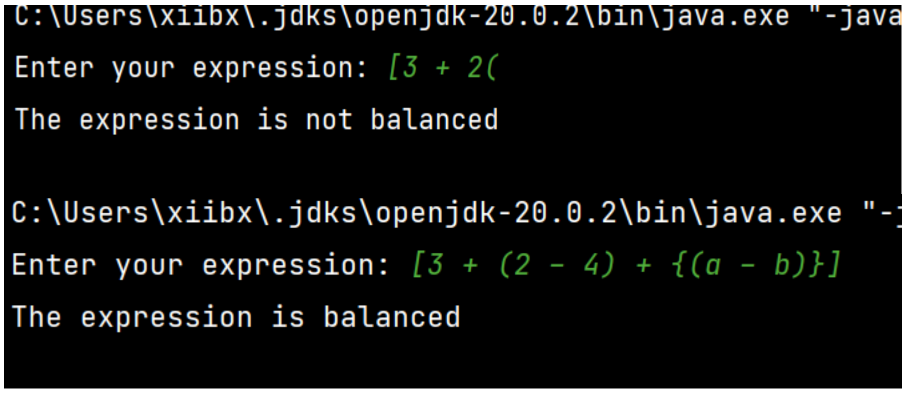


```
hi 5 3
Here's how I reverse a string:
Enter a string> ibrahim
The reversed string is:
miharbi

Process finished with exit code 0
```

Task2:

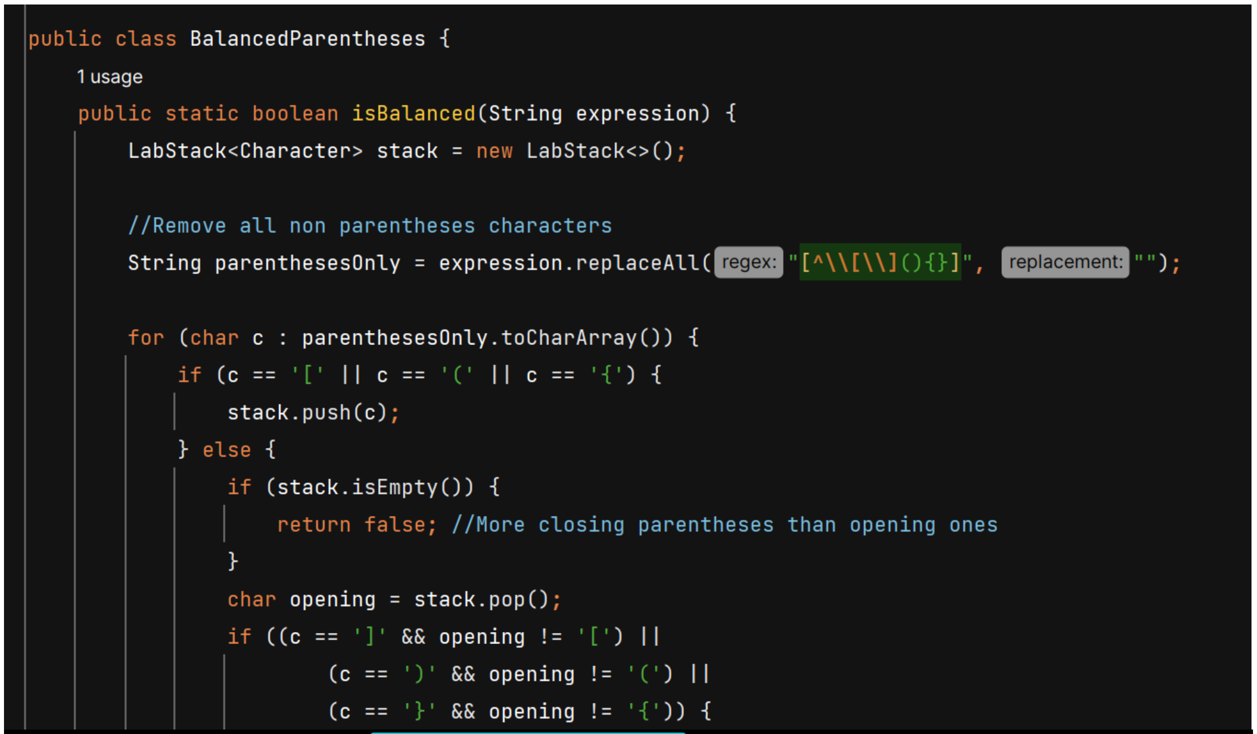
The BalancedParentheses class checks if a mathematical expression has balanced parentheses. It removes non parentheses characters and uses a stack to track opening parentheses. If the parentheses match it returns true; otherwise, it returns false. Finally, it prompts the user for input, evaluates the expression, and displays if the parentheses are balanced.

Task2 Output:

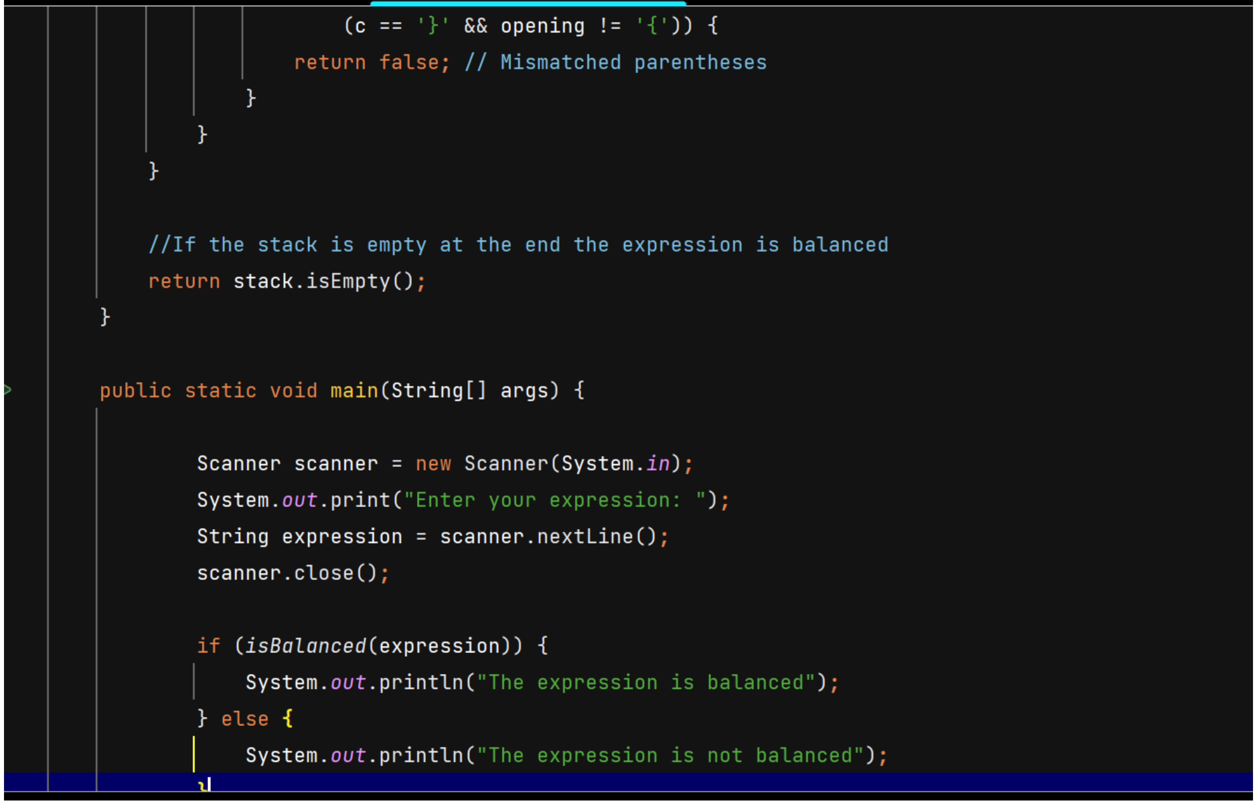


```
C:\Users\xiibx\.jdks\openjdk-20.0.2\bin\java.exe "-java
Enter your expression: [3 + 2(
The expression is not balanced

C:\Users\xiibx\.jdks\openjdk-20.0.2\bin\java.exe "-
Enter your expression: [3 + (2 - 4) + {(a - b)}]
The expression is balanced
```

Task 2 Code:



```
public class BalancedParentheses {
    1 usage
    public static boolean isBalanced(String expression) {
        LabStack<Character> stack = new LabStack<>();

        //Remove all non parentheses characters
        String parenthesesOnly = expression.replaceAll(regex: "[^\\[\\](){}]",  replacement: "");

        for (char c : parenthesesOnly.toCharArray()) {
            if (c == '[' || c == '(' || c == '{') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) {
                    return false; //More closing parentheses than opening ones
                }
                char opening = stack.pop();
                if ((c == ']' && opening != '[') ||
                        (c == ')' && opening != '(') ||
                        (c == '}' && opening != '{')) {
```

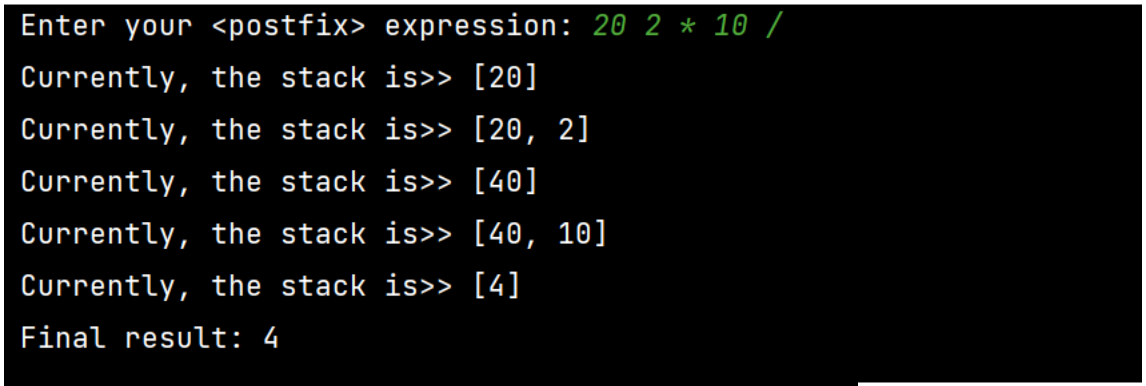


```
                        (c == '}' && opening != '{')) {
                    return false; // Mismatched parentheses
                }
            }
        }

        //If the stack is empty at the end the expression is balanced
        return stack.isEmpty();
    }

    public static void main(String[] args) {

            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter your expression: ");
            String expression = scanner.nextLine();
            scanner.close();

            if (isBalanced(expression)) {
                System.out.println("The expression is balanced");
            } else {
                System.out.println("The expression is not balanced");
            }
```

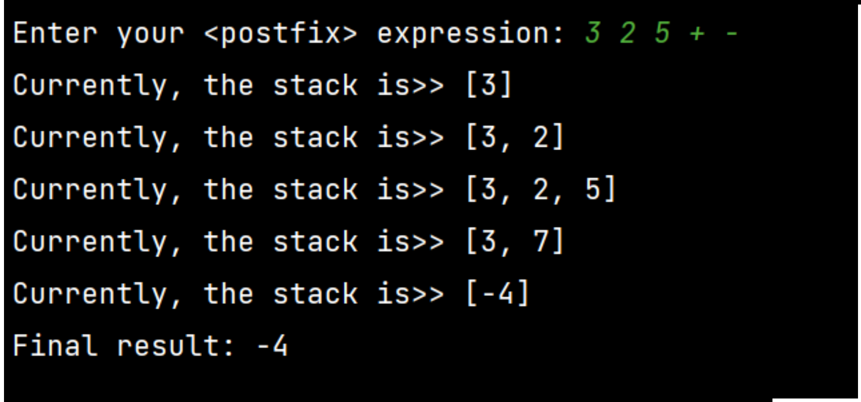
Task3:

The program checks each token in the expression: if it's an integer, it's pushed onto the stack. if it's an operator, it ensures there are enough operands to perform the operation. After processing all tokens, it verifies if there's only one element left in the stack, retrieving the result if so. Otherwise, it displays an error message.

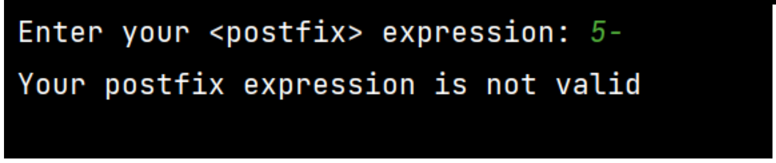
Task3 Output:



```
Enter your <postfix> expression: 20 2 * 10 /
Currently, the stack is>> [20]
Currently, the stack is>> [20, 2]
Currently, the stack is>> [40]
Currently, the stack is>> [40, 10]
Currently, the stack is>> [4]
Final result: 4
```

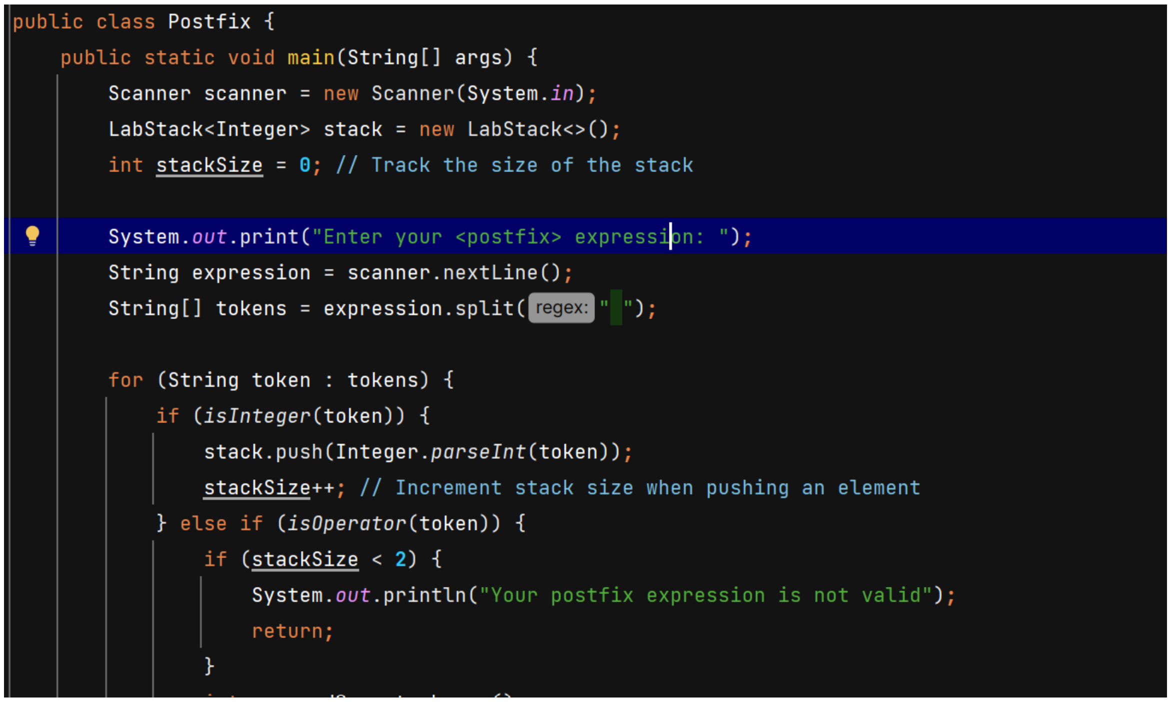


```
Enter your <postfix> expression: 3 2 5 + -
Currently, the stack is>> [3]
Currently, the stack is>> [3, 2]
Currently, the stack is>> [3, 2, 5]
Currently, the stack is>> [3, 7]
Currently, the stack is>> [-4]
Final result: -4
```

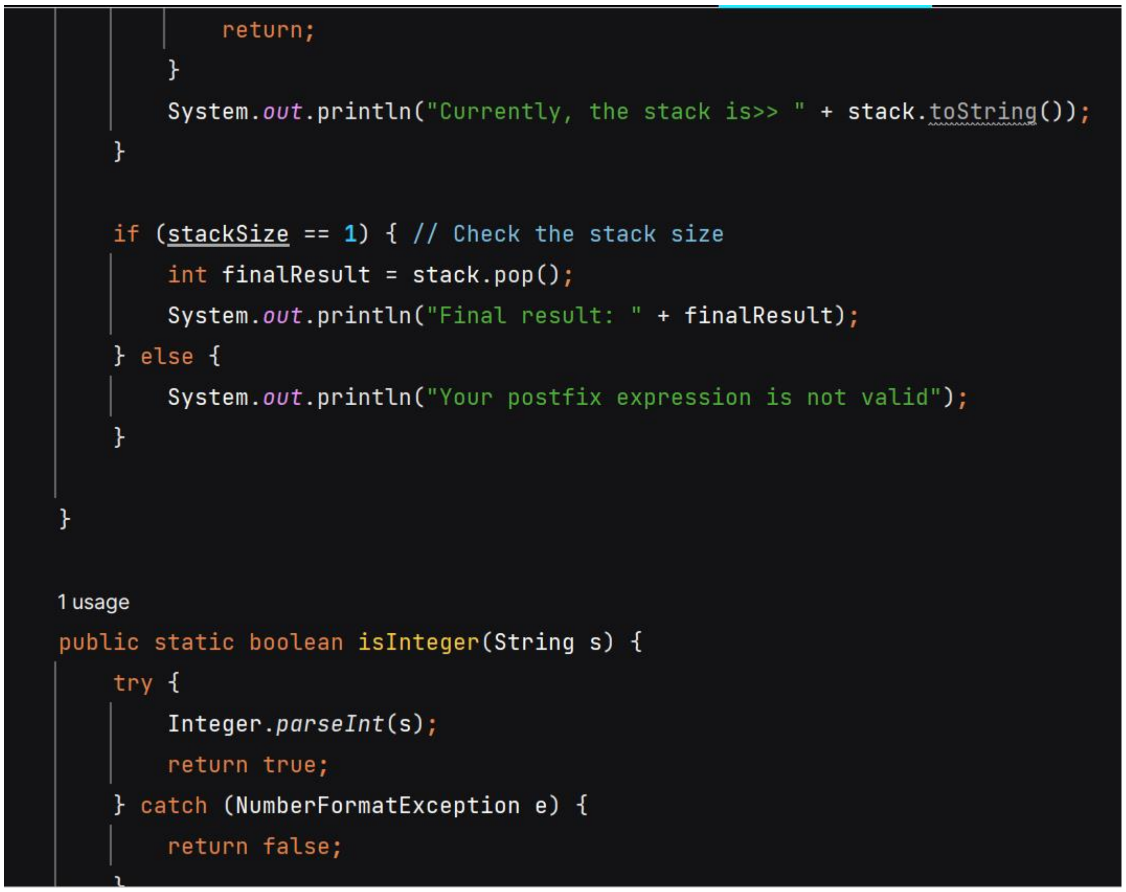


```
Enter your <postfix> expression: 5-
Your postfix expression is not valid
```

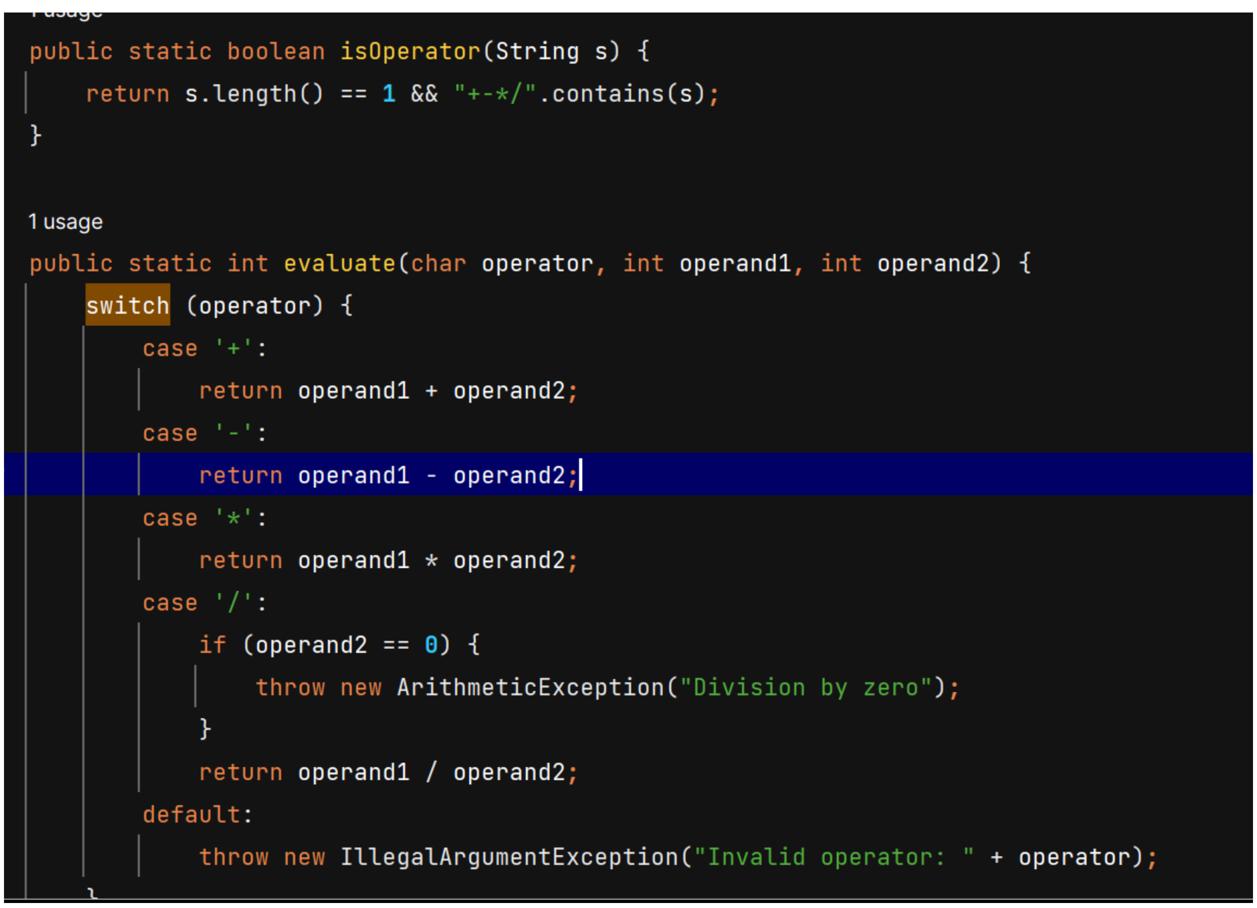
Task3 Code:



```java
public class Postfix {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LabStack<Integer> stack = new LabStack<>();
        int stackSize = 0; // Track the size of the stack

        System.out.print("Enter your <postfix> expression: ");
        String expression = scanner.nextLine();
        String[] tokens = expression.split(" ");

        for (String token : tokens) {
            if (isInteger(token)) {
                stack.push(Integer.parseInt(token));
                stackSize++; // Increment stack size when pushing an element
            } else if (isOperator(token)) {
                if (stackSize < 2) {
                    System.out.println("Your postfix expression is not valid");
                    return;
                }
```



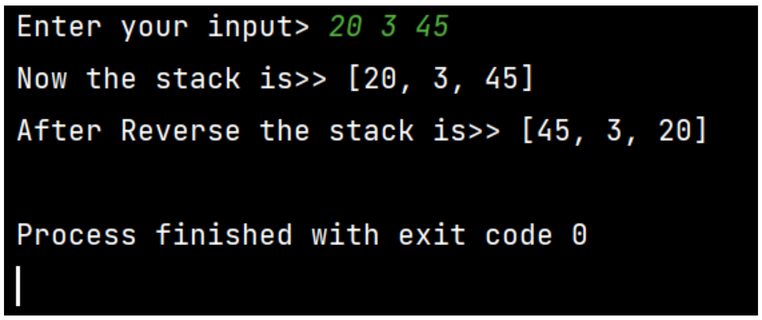
```
            return;
        }
        System.out.println("Currently, the stack is>> " + stack.toString());
    }

    if (stackSize == 1) { // Check the stack size
        int finalResult = stack.pop();
        System.out.println("Final result: " + finalResult);
    } else {
        System.out.println("Your postfix expression is not valid");
    }

}

1 usage
public static boolean isInteger(String s) {
    try {
        Integer.parseInt(s);
        return true;
    } catch (NumberFormatException e) {
        return false;
```



```
public static boolean isOperator(String s) {
    return s.length() == 1 && "+-*/".contains(s);
}

1 usage
public static int evaluate(char operator, int operand1, int operand2) {
    switch (operator) {
        case '+':
            return operand1 + operand2;
        case '-':
            return operand1 - operand2;
        case '*':
            return operand1 * operand2;
        case '/':
            if (operand2 == 0) {
                throw new ArithmeticException("Division by zero");
            }
            return operand1 / operand2;
        default:
            throw new IllegalArgumentException("Invalid operator: " + operator);
```

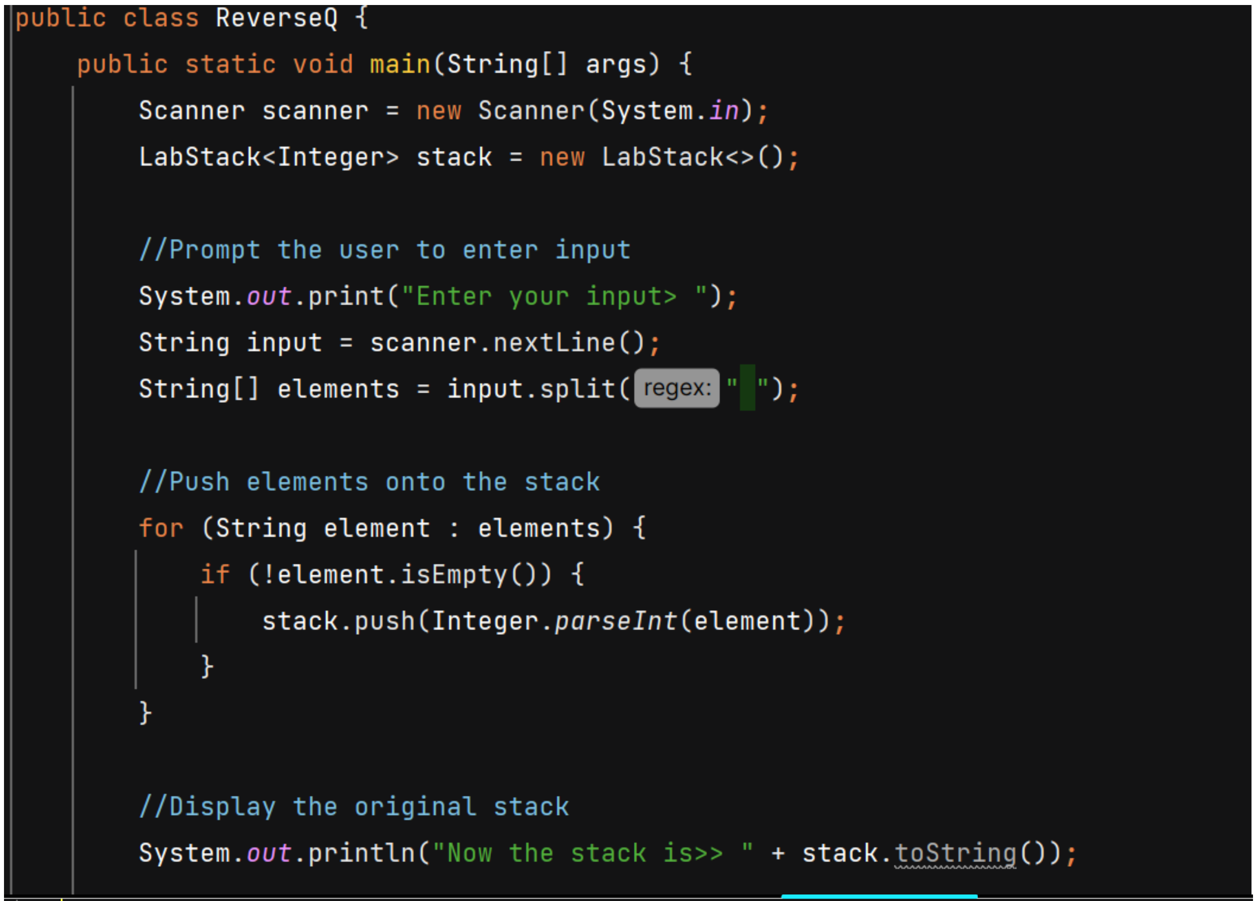
Task4: The program reverses the order of elements in a stack using a LabQueue. It takes input from the user, splits it into individual elements, and pushes them onto the stack. Then, it reverses the order of elements by dequeuing them from the stack, enqueuing them into a LabQueue, and then dequeuing them from the queue and pushing them back onto the stack. Finally, it displays the reversed stack content.

Task4 Output:

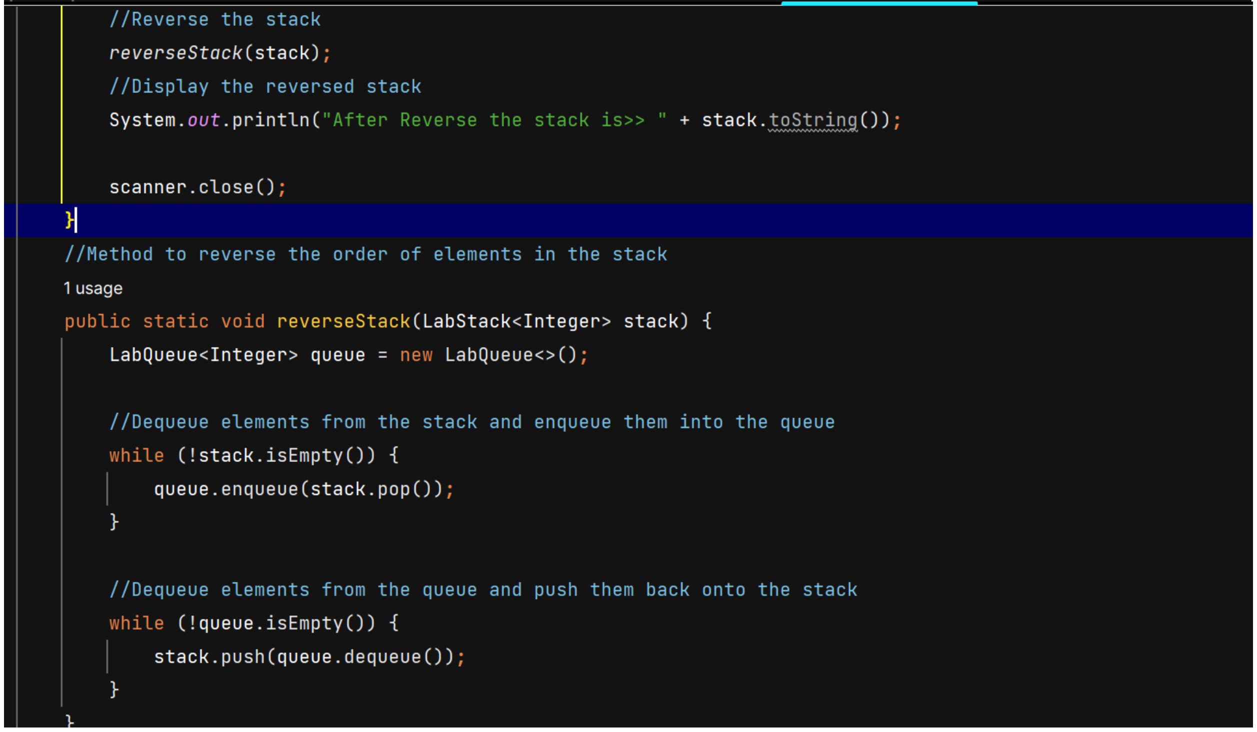


```
Enter your input> 20 3 45
Now the stack is>> [20, 3, 45]
After Reverse the stack is>> [45, 3, 20]

Process finished with exit code 0
```

Task4 Code:



```java
public class ReverseQ {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        LabStack<Integer> stack = new LabStack<>();

        //Prompt the user to enter input
        System.out.print("Enter your input> ");
        String input = scanner.nextLine();
        String[] elements = input.split( regex: " ");

        //Push elements onto the stack
        for (String element : elements) {
            if (!element.isEmpty()) {
                stack.push(Integer.parseInt(element));
            }
        }

        //Display the original stack
        System.out.println("Now the stack is>> " + stack.toString());
```



```java
        //Reverse the stack
        reverseStack(stack);
        //Display the reversed stack
        System.out.println("After Reverse the stack is>> " + stack.toString());

        scanner.close();
    }
    //Method to reverse the order of elements in the stack
    1 usage
    public static void reverseStack(LabStack<Integer> stack) {
        LabQueue<Integer> queue = new LabQueue<>();

        //Dequeue elements from the stack and enqueue them into the queue
        while (!stack.isEmpty()) {
            queue.enqueue(stack.pop());
        }

        //Dequeue elements from the queue and push them back onto the stack
        while (!queue.isEmpty()) {
            stack.push(queue.dequeue());
        }
    }
```