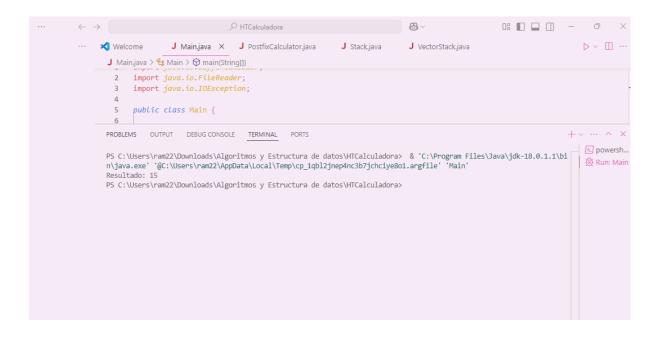
Capturas:

Funcionamiento clase Main:



En consola:

```
Windows PowerShell

Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\ram22\Downloads\Algoritmos y Estructura de datos\HTCalculadora> javac Main.java

PS C:\Users\ram22\Downloads\Algoritmos y Estructura de datos\HTCalculadora> java Main

Resultado: 15

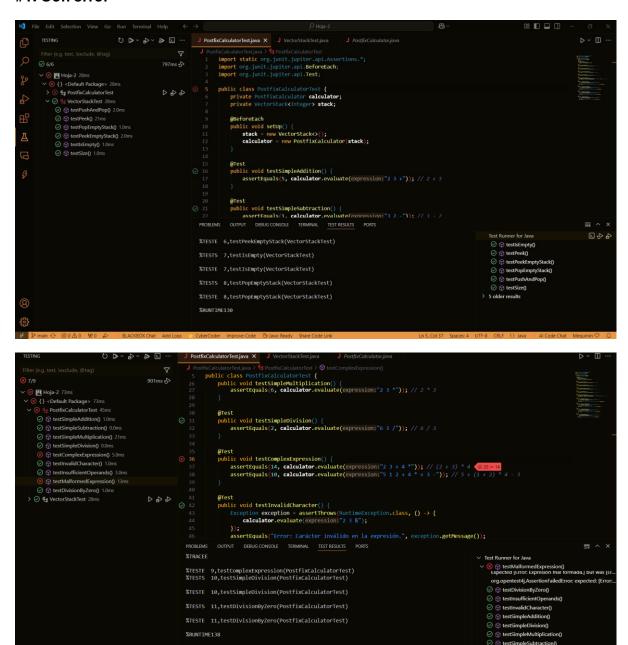
PS C:\Users\ram22\Downloads\Algoritmos y Estructura de datos\HTCalculadora>
```

• Para evaluar se utilizo la expresión en el archivo datos.txt: 1 2 + 4 * 3 +. Como funciona la calculadora es que hace el siguiente procedimiento:

```
push(1) // Envia el valor 1
push(2) // Envia el valor 2
Pop() // Función de procedimiento
2
pop() // Se aplica la función de suma
1
(1,2,+) = 3 // Se aplica la suma
Push(4) // Envia al siguiente valor (4)
Pop() // Función multiplicación
3
(3,4,*) =12 // Se realiza la función multiplicación
push(3)
Pop()
3
Pop()
12
(3, 12, +) =15
```

Pruebas Unitarias:

#1. Con error



#2. Correctas:

```
import static org.junit.jupiter.api.Assertions.*;
      1
      2
           import org.junit.jupiter.api.BeforeEach;
      3
           import org.junit.jupiter.api.Test;
     5
           public class PostfixCalculatorTest {
               private PostfixCalculator calculator;
      6
      7
               private VectorStack<Integer> stack;
      8
      9
               @BeforeEach
     10
                public void setUp() {
     11
                  stack = new VectorStack<>();
                    calculator = new PostfixCalculator(stack);
     12
     13
     14
     15
               @Test

    √ 16

                public void testSimpleAddition() {
                  assertEquals(5, calculator.evaluate(expression: "2 3 +")); // 2 + 3
     17
     18
     19
               @Test
     20
 21
                public void testSimpleSubtraction() {
    22
                  assertEquals(1, calculator.evaluate(expression:"3 2 -")); // 3 - 2
     23
     24
     25
               @Test
 public void testSimpleMultiplication() {
                  assertEquals(6, calculator.evaluate(expression:"2 3 *")); // 2 * 3
     27
     28
     29
  29
  30
           @Test
public void testSimpleDivision() {
  32
              assertEquals(2, calculator.evaluate(expression: "6 3 /")); // 6 / 3
  33
  34
  35
public void testComplexExpression() {
             assertEquals(20, calculator.evaluate(expression:"2 3 + 4 *")); // (2 + 3) * 4 assertEquals(14, calculator.evaluate(expression:"5 1 2 + 4 * + 3 -")); // 5 + ((1 + 2) * 4) - 3
  37
  38
  39
  40
  41
public void testInvalidCharacter() {
  43
              Exception exception = assertThrows(RuntimeException.class, () -> {
  44
                  calculator.evaluate(expression:"2 3 &");
  45
               assertEquals("Error: Carácter inválido en la expresión.", exception.getMessage());
  47
  49
public void testInsufficientOperands() {
  51
              Exception exception = assertThrows(RuntimeException.class, () -> {
  52
                  calculator.evaluate(expression:"2 +");
  53
  54
               assertEquals("Error: Operandos insuficientes.", exception.getMessage());
```

```
56
   57
            @Test
            public void testMalformedExpression() {
Exception exception = assertThrows(RuntimeException.class, () -> {
   59
                calculator.evaluate(expression:"2 3 + 4 + +");
   60
   61
               });
   62
                assertEquals("Error: Expresión mal formada.", exception.getMessage());
   63
   64
   65
         🖁 @Test
public void testDivisionByZero() {
   67
               Exception exception = assertThrows(ArithmeticException.class, () -> {
   68
                calculator.evaluate(expression:"4 0 /");
   69
   70
                assertEquals("Error: División por cero.", exception.getMessage());
   71
   72
```