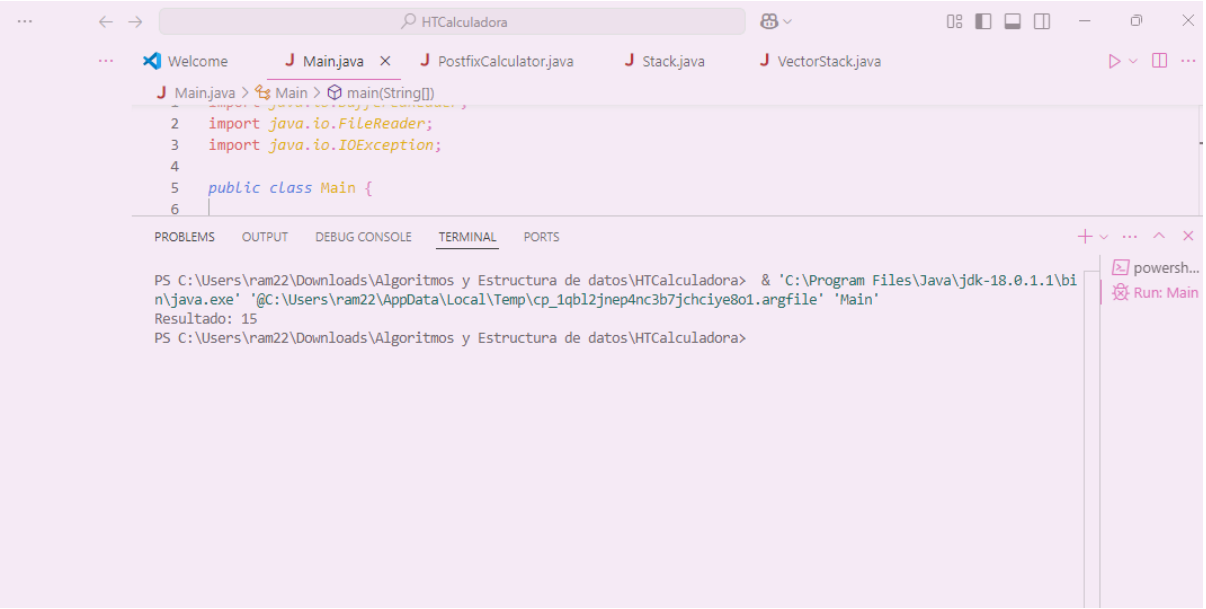
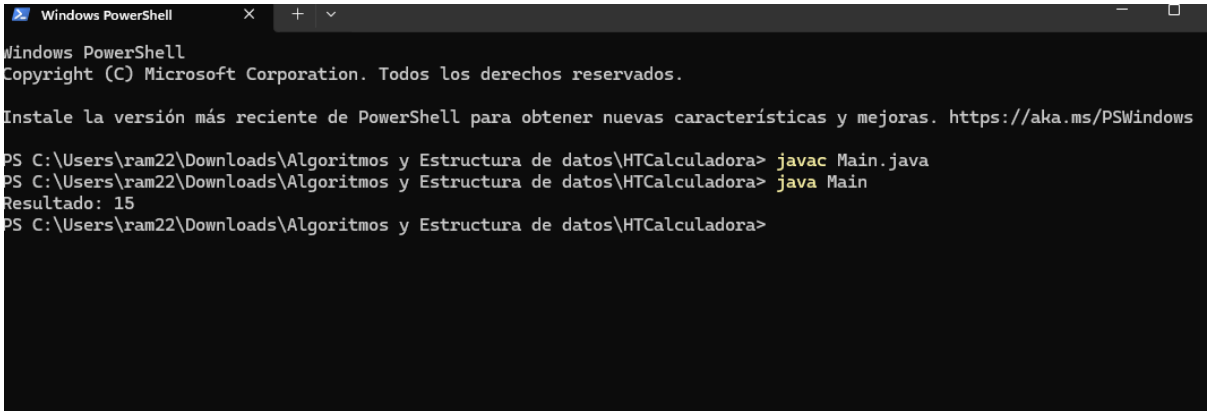
Capturas:

Funcionamiento clase Main:



En consola:



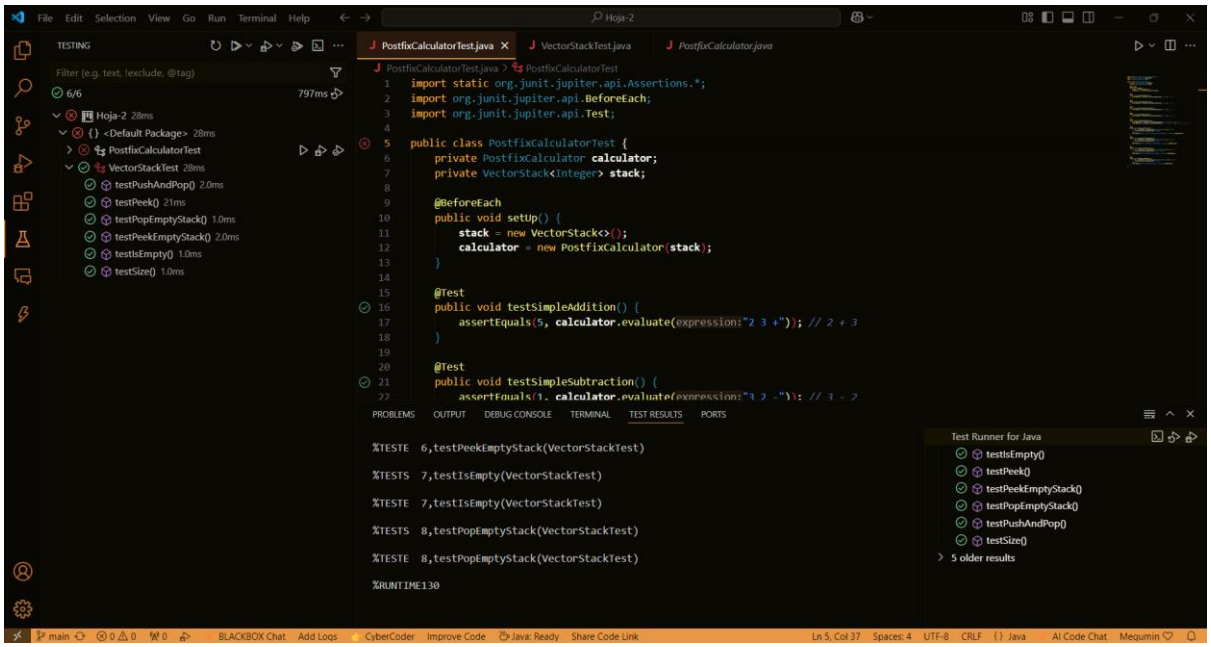
- Para evaluar se utilizo la expresión en el archivo datos.txt: 1 2 + 4 * 3 +. Como funciona la calculadora es que hace el siguiente procedimiento:

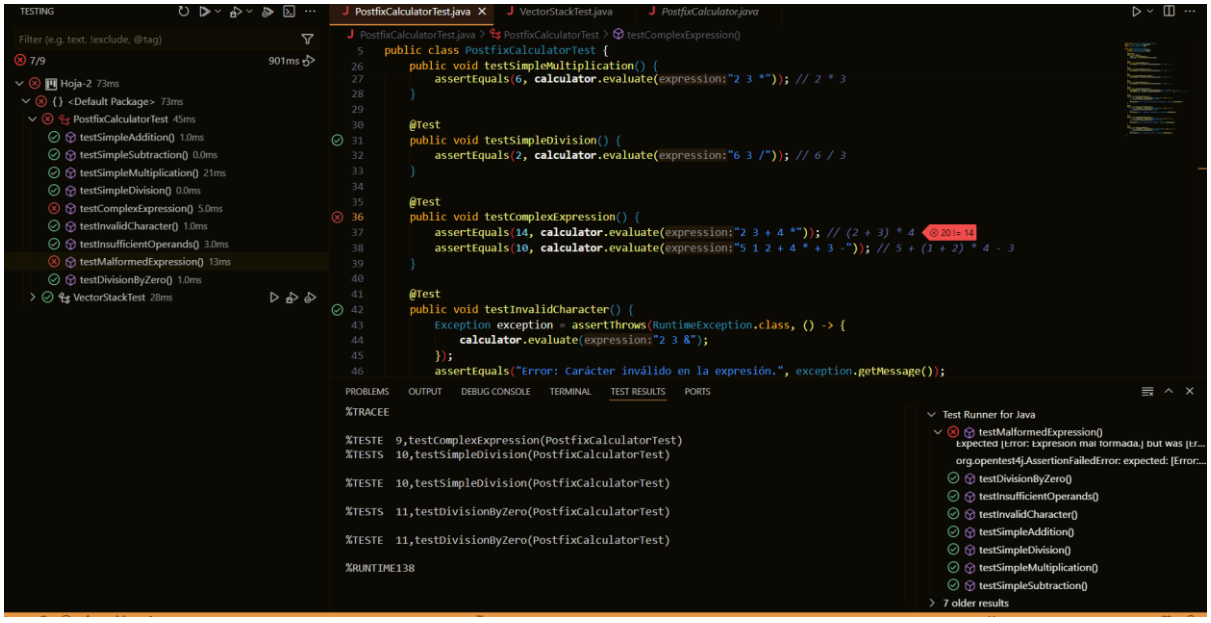
  push(1) // Envia el valor 1
  push(2) // Envia el valor 2
  Pop() // Función de procedimiento
  2
  pop() // Se aplica la función de suma
  1
  (1,2, +) = 3 // Se aplica la suma
  Push(4) // Envia al siguiente valor (4)
  Pop() // Función multiplicación
  3
  (3,4, *) =12 // Se realiza la función multiplicación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

(3, 12, +) =15

Pruebas Unitarias:

#1. Con error





#2. Correctas:

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class PostfixCalculatorTest {
    private PostfixCalculator calculator;
    private VectorStack<Integer> stack;

    @BeforeEach
    public void setUp() {
        stack = new VectorStack<>();
        calculator = new PostfixCalculator(stack);
    }

    @Test
    public void testSimpleAddition() {
        assertEquals(5, calculator.evaluate(expression:"2 3 +")); // 2 + 3
    }

    @Test
    public void testSimpleSubtraction() {
        assertEquals(1, calculator.evaluate(expression:"3 2 -")); // 3 - 2
    }

    @Test
    public void testSimpleMultiplication() {
        assertEquals(6, calculator.evaluate(expression:"2 3 *")); // 2 * 3
    }

    @Test
    public void testSimpleDivision() {
        assertEquals(2, calculator.evaluate(expression:"6 3 /")); // 6 / 3
    }

    @Test
    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
    }

    @Test
    public void testInvalidCharacter() {
        Exception exception = assertThrows(RuntimeException.class, () -> {
            calculator.evaluate(expression:"2 3 &");
        });
        assertEquals("Error: Carácter inválido en la expresión.", exception.getMessage());
    }

    @Test
    public void testInsufficientOperands() {
        Exception exception = assertThrows(RuntimeException.class, () -> {
            calculator.evaluate(expression:"2 +");
        });
        assertEquals("Error: Operandos insuficientes.", exception.getMessage());
    }
```

```
56
57      @Test
58      public void testMalformedExpression() {
59          Exception exception = assertThrows(RuntimeException.class, () -> {
60              calculator.evaluate(expression:"2 3 + 4 + +");
61          });
62          assertEquals("Error: Expresión mal formada.", exception.getMessage());
63      }
64
65      @Test
66      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  }
```