

5/5/2025

TP1

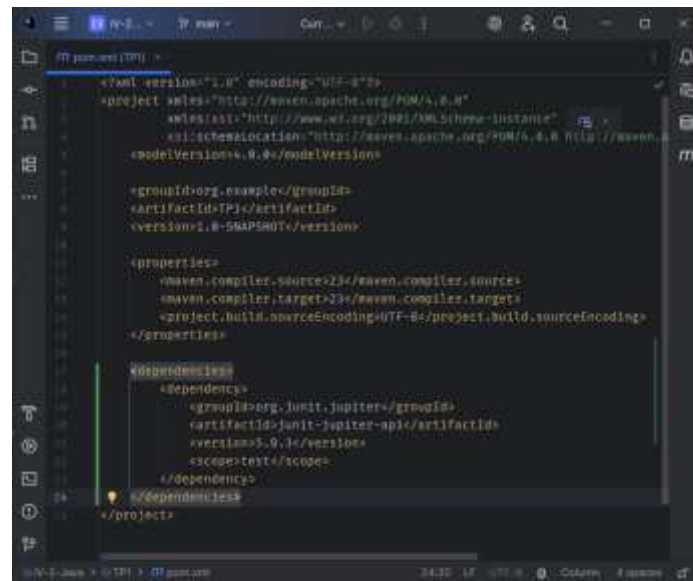
**Desenvolvimento de Serviços Web e Testes
com Java**

Professor(a): Bernardo Petry Prates

LINK GITHUB

<https://github.com/faculdade-infnet/IV-2-Java/tree/main/TP1>

1. EXERCÍCIO



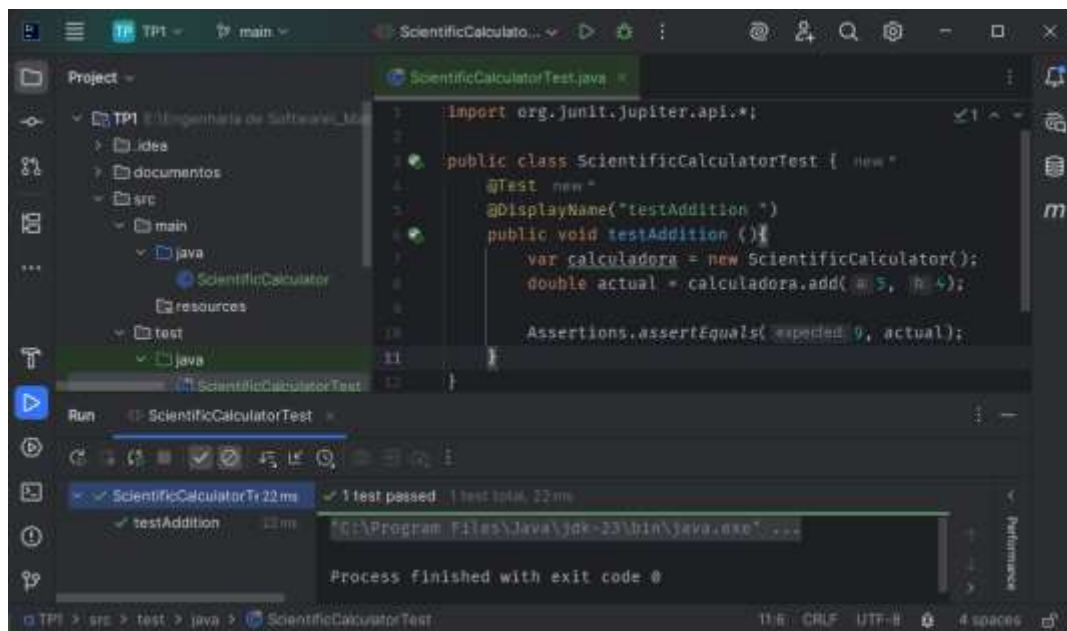
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns: xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>TP1</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>23</maven.compiler.source>
    <maven.compiler.target>23</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.9.3</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

2. EXERCÍCIO



```
import org.junit.jupiter.api.*;

public class ScientificCalculatorTest {
    @Test
    @DisplayName("testAddition")
    public void testAddition () {
        var calculadora = new ScientificCalculator();
        double actual = calculadora.add(5, 4);

        Assertions.assertEquals(9, actual);
    }
}
```

Run ScientificCalculatorTest

ScientificCalculatorTest 22 ms ✓ 1 test passed | test total: 22 ms

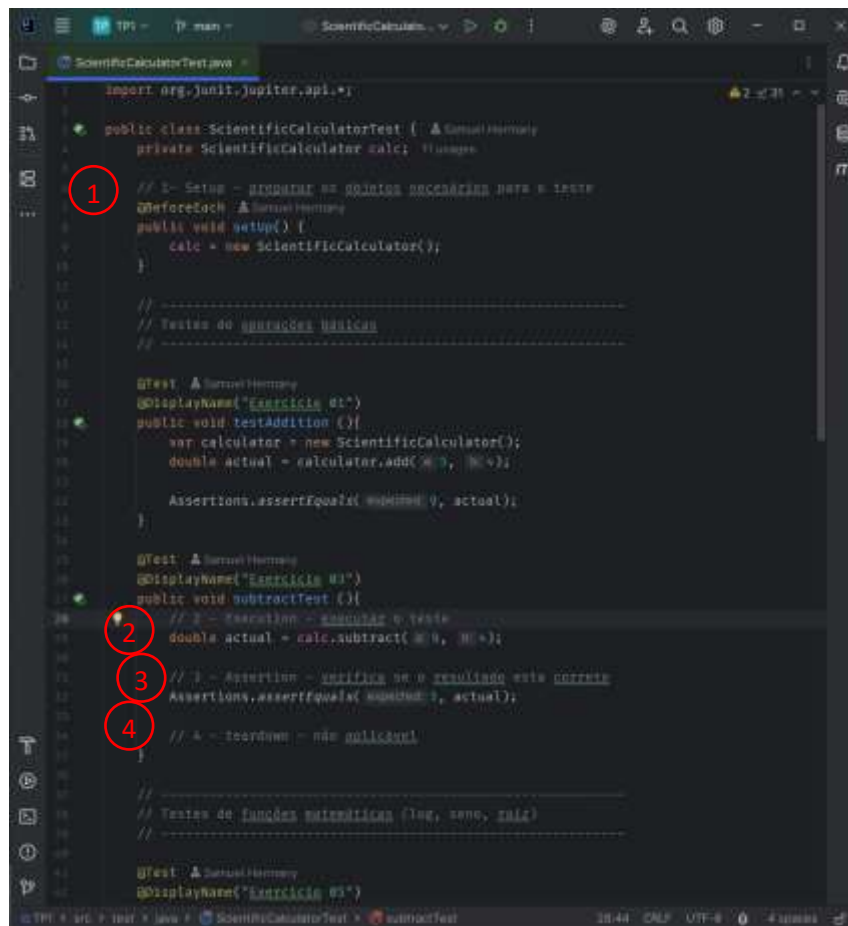
testAddition 22 ms

Process finished with exit code 0

Acredito que quando pede o significado dos itens sejam:

- Verde - O teste foi executado com sucesso;
- Vermelho - O teste falhou;
- Amarelo – Ocorreu uma exceção durante o teste.

3. EXERCÍCIO



```
import org.junit.jupiter.api.*;

public class ScientificCalculatorTest {
    private ScientificCalculator calc;

    // 1- Setup - preparar os objetos necessários para o teste
    @BeforeEach
    public void setUp() {
        calc = new ScientificCalculator();
    }

    // -----
    // Testes de operações básicas
    // -----

    @Test
    @DisplayName("Exercício 01")
    public void testAddition() {
        var calculator = new ScientificCalculator();
        double actual = calculator.add(5, 10);

        Assertions.assertEquals(expected: 15, actual);
    }

    @Test
    @DisplayName("Exercício 02")
    public void subtractTest() {
        // 2 - Execution - executar o teste
        double actual = calc.subtract(10, 10);

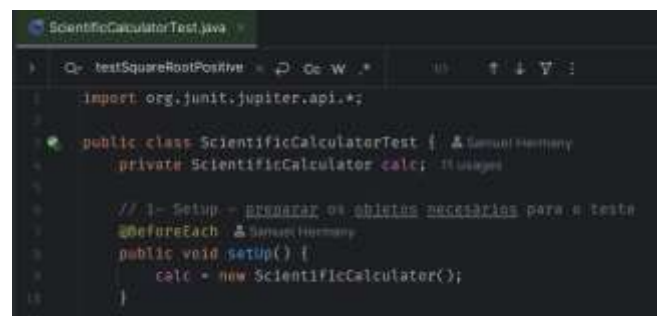
        // 3 - Assertion - verificar se o resultado está correto
        Assertions.assertEquals(expected: 0, actual);

        // 4 - Teardown - não aplicar
    }

    // -----
    // Testes de funções matemáticas (log, seno, etc)
    // -----

    @Test
    @DisplayName("Exercício 03")
}
```

4. EXERCÍCIO



```
import org.junit.jupiter.api.*;

public class ScientificCalculatorTest {
    private ScientificCalculator calc;

    // 1- Setup - preparar os objetos necessários para o teste
    @BeforeEach
    public void setUp() {
        calc = new ScientificCalculator();
    }

    @Test
    @DisplayName("Exercício 04")
    public void testSquareRootPositive() {
        double actual = calc.squareRoot(25);

        Assertions.assertEquals(expected: 5, actual);
    }

    @Test
    @DisplayName("Exercício 05")
    public void logMultiplasEntradasTest() {
        double actual = calc.log(1);
        Assertions.assertEquals(expected: 0, actual);

        actual = calc.log(10);
        Assertions.assertEquals(expected: 2.302585092994046, actual);

        actual = calc.log(100);
        Assertions.assertEquals(expected: 4.605170185988092, actual);
    }
}
```



```
@Test
@DisplayName("Exercício 05")
public void logMultiplasEntradasTest() {
    double actual = calc.log(1);
    Assertions.assertEquals(expected: 0, actual);

    actual = calc.log(10);
    Assertions.assertEquals(expected: 2.302585092994046, actual);

    actual = calc.log(100);
    Assertions.assertEquals(expected: 4.605170185988092, actual);
}
```

5. EXERCÍCIO

The screenshot shows an IDE with a Java test class `ScientificCalculatorTest`. The test method `testSquareRootPositive` is highlighted. The code is as follows:

```
@Test
@DisplayName("Exercício 05")
public void testSquareRootPositive () {
    double actual = calc.squareRoot(25);

    Assertions.assertEquals(5, actual);
}
```

The Run button is clicked, and the test is executed. The output window shows the following:

```
ScientificCalculatorTest: 24 ms
Exercício 05: 24 ms
1 test passed, 1 test total, 24 ms
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Process finished with exit code 0
```

6. EXERCÍCIO

The screenshot shows an IDE with a Java test class `ScientificCalculatorTest`. The test method `testSquareRootNegative` is highlighted. The code is as follows:

```
@Test
@DisplayName("Exercício 06")
public void testSquareRootNegative () {
    // Verifica se IllegalArgumentException é lançada quando o número é negativo
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        calc.squareRoot(-25);
    });
}
```

The Run button is clicked, and the test is executed. The output window shows the following:

```
ScientificCalculatorTest: 25 ms
Exercício 06: 25 ms
1 test passed, 1 test total, 25 ms
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Process finished with exit code 0
```

7. EXERCÍCIO

The screenshot shows an IDE with a Java test class `ScientificCalculatorTest`. The test method `testDivideByZero` is highlighted. The code is as follows:

```
@Test
@DisplayName("Exercício 07")
public void testDivideByZero () {
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        calc.divide(5, 0);
    });
}
```

The Run button is clicked, and the test is executed. The output window shows the following:

```
ScientificCalculatorTest: 26 ms
Exercício 07: 26 ms
1 test passed, 1 test total, 26 ms
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
Process finished with exit code 0
```

8. EXERCÍCIO

```
49 @Test
50 @DisplayName("Exercício 08")
51 public void logMultiplasEntradasTest () {
52     double actual = calc.log( 1);
53     Assertions.assertEquals( expected: 0, actual);
54
55     actual = calc.log( 10);
56     Assertions.assertEquals( expected: 2.302585092994046, actual);
57
58     actual = calc.log( 100);
59     Assertions.assertEquals( expected: 4.605170185988092, actual);
60 }
61
62 @Test
63 @DisplayName("Exercício 08")
64 public void sinMultiplasEntradasTest () {
65     double actual = calc.sin( degrees: 0);
66     Assertions.assertEquals( expected: 0, actual);
67
68     // Adicionando tolerância no teste
69     actual = calc.sin( degrees: 30);
70     Assertions.assertEquals( expected: 0.5, actual, delta: 0.0001);
71
72     actual = calc.sin( degrees: 90);
73     Assertions.assertEquals( expected: 1.0, actual);
74 }
75
76 Run ScientificCalculatorTest.sinMultiplasEntradasTest
77
78 ScientificCalculatorTest: 23 ms
79 Exercício 08: 23 ms
80
81 1 test passed 1 test total: 23 ms
82
83 C:\Program Files\Java\jdk-23\bin\java.exe
84
85 Process finished with exit code 0
86
87 src > test > java > ScientificCalculatorTest > sinMultiplasEntradasTest 73:30 CRLF UTF-8 4 spaces
```

9. EXERCÍCIO

- Quais métodos da calculadora merecem mais atenção nos testes?

Na minha visão são aqueles que possuem uma complexidade maior, como divisão que deve ser testada divisão por zero, e métodos como log e square root pois possuem a possibilidade de erro ao utilizar valores negativos

- Como a cobertura de código pode ajudar a identificar lacunas?

Se uma linha de código ou ramo não está sendo executado durante os testes, você verá que essa parte precisa de mais atenção, evidenciando uma possível área de testes.

10. EXERCÍCIO

Exercicio 06 –	nome_antigo	- nome_correto
	testSquareRootNegative	- squareRootNegativeTest
Exercicio 07 –	nome_antigo	- nome_correto
	testDivideByZero	- divideByZeroTest

11. EXERCÍCIO

Está no código os grupos por comentário