



Apellidos, Nombre: Iglesias Nieto, Rodrigo



8. Proyecto pruebas de software

CA3.7 Documentouse a estratexia de probas e os resultados obtidos. (20%)

8.1 Analiza el código fuente “DecimalCalculator” que se ha facilitado para este proyecto y crea una batería pruebas de software unitarias automatizada y ejecútala. Muestra el código fuente de la clase de pruebas que se ha generado de forma automática, así como el resultado de la ejecución de las pruebas.

8.2 (Pruebas unitarias) Modifica el código fuente de las pruebas que se ha autogenerado para tener en cuenta, si es que procede, lo siguiente:

- Interfaz de módulo correspondiente.
- Impacto de los datos globales sobre el módulo.
- Estructuras de datos en el módulo.
- Las condiciones límite.
- Los distintos caminos de ejecución de las estructuras de control.

CA3.3 Realizáronse probas de regresión. (10%)

8.3 (Pruebas de regresión) Un desarrollador ha modificado el código del método divide de la siguiente forma:

```
public double divide(double operand2, double operand1) {  
    return operand1 / operand2;  
}
```

Implementa este cambio en el proyecto y realiza las correspondientes pruebas de regresión. En el caso de que haya errores de regresión, implementa una solución sabiendo que sólo se puede modificar el cuerpo del módulo e indica qué sentencias han sido modificadas.

CA3.4 Realizáronse probas de volume e estrés. (10%)

8.4 (Pruebas de volumen y estrés) Investiga si para el módulo dado sería necesario realizar pruebas de volumen y estrés. Investiga en qué casos sería necesario implementarlas.

CA3.5 Realizáronse pruebas de seguridad. (10%)

8.5 (Pruebas de seguridad) Investiga si para el módulo dado sería necesario realizar pruebas de seguridad. Investiga en qué casos sería necesario implementarlas.

CA3.6 Realizáronse pruebas de uso de recursos por parte da aplicación. (10%)

8.6 (Pruebas de uso de recursos) Investiga cuando tiempo tarda el programa en realizar 1.000.000.000 de divisiones consecutivas

Entrega el proyecto en formato pdf empleando la siguiente nomenclatura: UD8_Apellido1_Apellido2_Nombre.pdf

(Completar...)

8.1)

```
package calculator;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author iglnierod
 */
public class DecimalCalculatorTest {

    public DecimalCalculatorTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
    }

    @After
    public void tearDown() {
    }
}
```

```

/**
 * Test of add method, of class DecimalCalculator.
 */
@Test
public void testAdd() {
    System.out.println("add");
    double operand1 = 0.0;
    double operand2 = 0.0;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = 0.0;
    double result = instance.add(operand1, operand2);
    assertEquals(expectedResult, result, 0);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of subtract method, of class DecimalCalculator.
 */
@Test
public void testSubtract() {
    System.out.println("subtract");
    double operand1 = 0.0;
    double operand2 = 0.0;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = 0.0;
    double result = instance.subtract(operand1, operand2);
    assertEquals(expectedResult, result, 0);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of multiply method, of class DecimalCalculator.
 */
@Test
public void testMultiply() {
    System.out.println("multiply");
    double operand1 = 0.0;
    double operand2 = 0.0;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = 0.0;
    double result = instance.multiply(operand1, operand2);
    assertEquals(expectedResult, result, 0);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of divide method, of class DecimalCalculator.
 */
@Test
public void testDivide() {
    System.out.println("divide");
    double operand2 = 0.0;
    double operand1 = 0.0;
    DecimalCalculator instance = new DecimalCalculator();

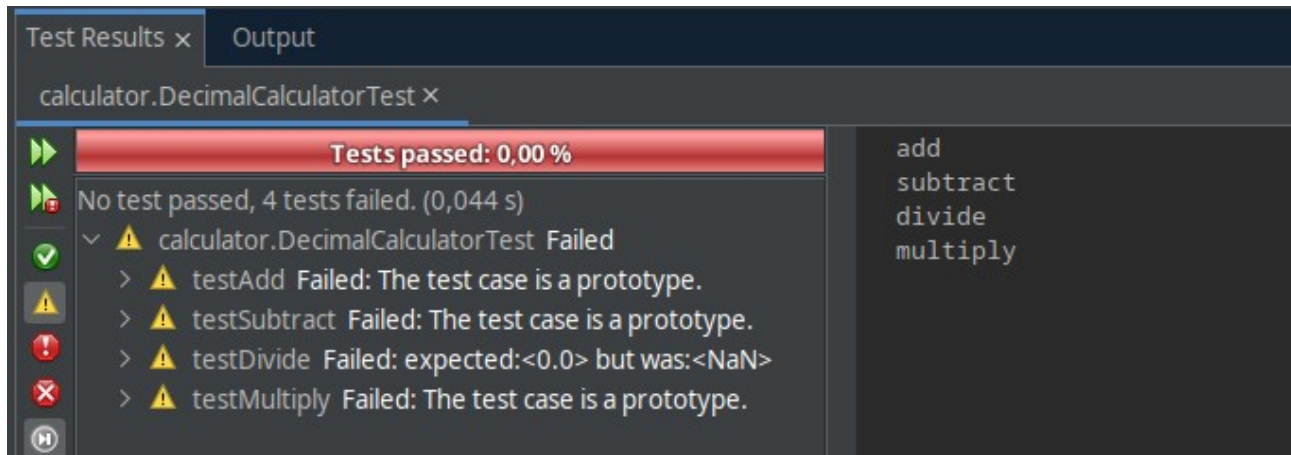
```

```

        double expResult = 0.0;
        double result = instance.divide(operand2, operand1);
        assertEquals(expResult, result, 0);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }
}

```

Resultados:



8.2)

Código fuente de la clase DecimalCalculatorTest

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change
 * this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/UnitTests/JUnit4TestClass.java to edit
 * this template
 */
package calculator;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author iglnierod
 */
public class DecimalCalculatorTest {

    public DecimalCalculatorTest() {
    }

    @BeforeClass
    public static void setUpClass() {
        //System.out.println("Before Class");
    }
}

```

```

}

@AfterClass
public static void tearDownClass() {
    //System.out.println("After Class");
}

@Before
public void setUp() {
    //System.out.println("Before...");
}

@After
public void tearDown() {
    //System.out.println("After...");
}

/**
 * Test of add method, of class DecimalCalculator.
 */
@Test
public void testAdd1() {
    System.out.println("add");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = 0.0;
    double result = instance.add(operand1, operand2);
    assertEquals(expectedResult, result, 0.000001);
}

/**
 * Test of add method, of class DecimalCalculator.
 */
@Test
public void testAdd2() {
    System.out.println("add");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = Double.POSITIVE_INFINITY;
    double result = instance.add(operand1, operand2);
    assertEquals(expectedResult, result, 0);
}

/**
 * Test of subtract method, of class DecimalCalculator.
 */
@Test
public void testSubtract1() {
    System.out.println("subtract");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expectedResult = 0.0;
    double result = instance.subtract(operand1, operand2);

```

```

        assertEquals(expResult, result, 0);
    }

    /**
     * Test of subtract method, of class DecimalCalculator.
     */
    @Test
    public void testSubtract2() {
        System.out.println("subtract");
        double operand1 = Double.MAX_VALUE;
        double operand2 = Double.MAX_VALUE;
        DecimalCalculator instance = new DecimalCalculator();
        double expResult = 0.0;
        double result = instance.subtract(operand1, operand2);
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of multiply method, of class DecimalCalculator.
     */
    @Test
    public void testMultiply1() {
        System.out.println("multiply");
        double operand1 = Double.MIN_VALUE;
        double operand2 = Double.MIN_VALUE;
        DecimalCalculator instance = new DecimalCalculator();
        double expResult = Double.MIN_VALUE * Double.MIN_VALUE;
        double result = instance.multiply(operand1, operand2);
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of multiply method, of class DecimalCalculator.
     */
    @Test
    public void testMultiply2() {
        System.out.println("multiply");
        double operand1 = Double.MAX_VALUE;
        double operand2 = Double.MAX_VALUE;
        DecimalCalculator instance = new DecimalCalculator();
        double expResult = Double.POSITIVE_INFINITY;
        double result = instance.multiply(operand1, operand2);
        assertEquals(expResult, result, 0);
    }

    /**
     * Test of divide method, of class DecimalCalculator.
     */
    @Test
    public void testDivide1() {
        System.out.println("divide");
        double operand1 = Double.MIN_VALUE;
        double operand2 = Double.MIN_VALUE;
        DecimalCalculator instance = new DecimalCalculator();
        double expResult = 1;
        double result = instance.divide(operand1, operand2);
        assertEquals(expResult, result, 0);
    }
}

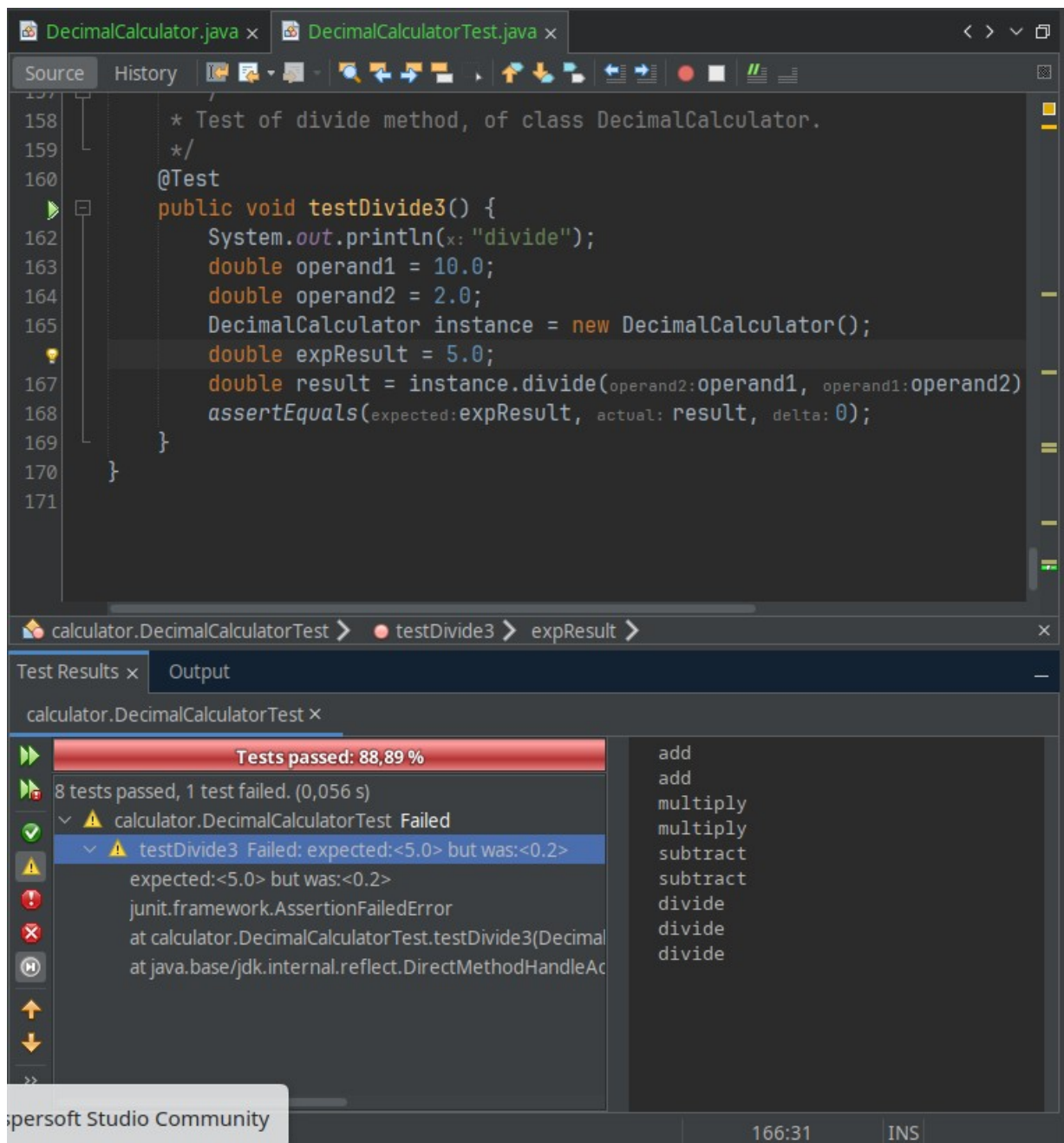
```

```

/**
 * Test of divide method, of class DecimalCalculator.
 */
@Test
public void testDivide2() {
    System.out.println("divide");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1;
    double result = instance.divide(operand1, operand2);
    assertEquals(expResult, result, 0);
}
}

```

8.3)



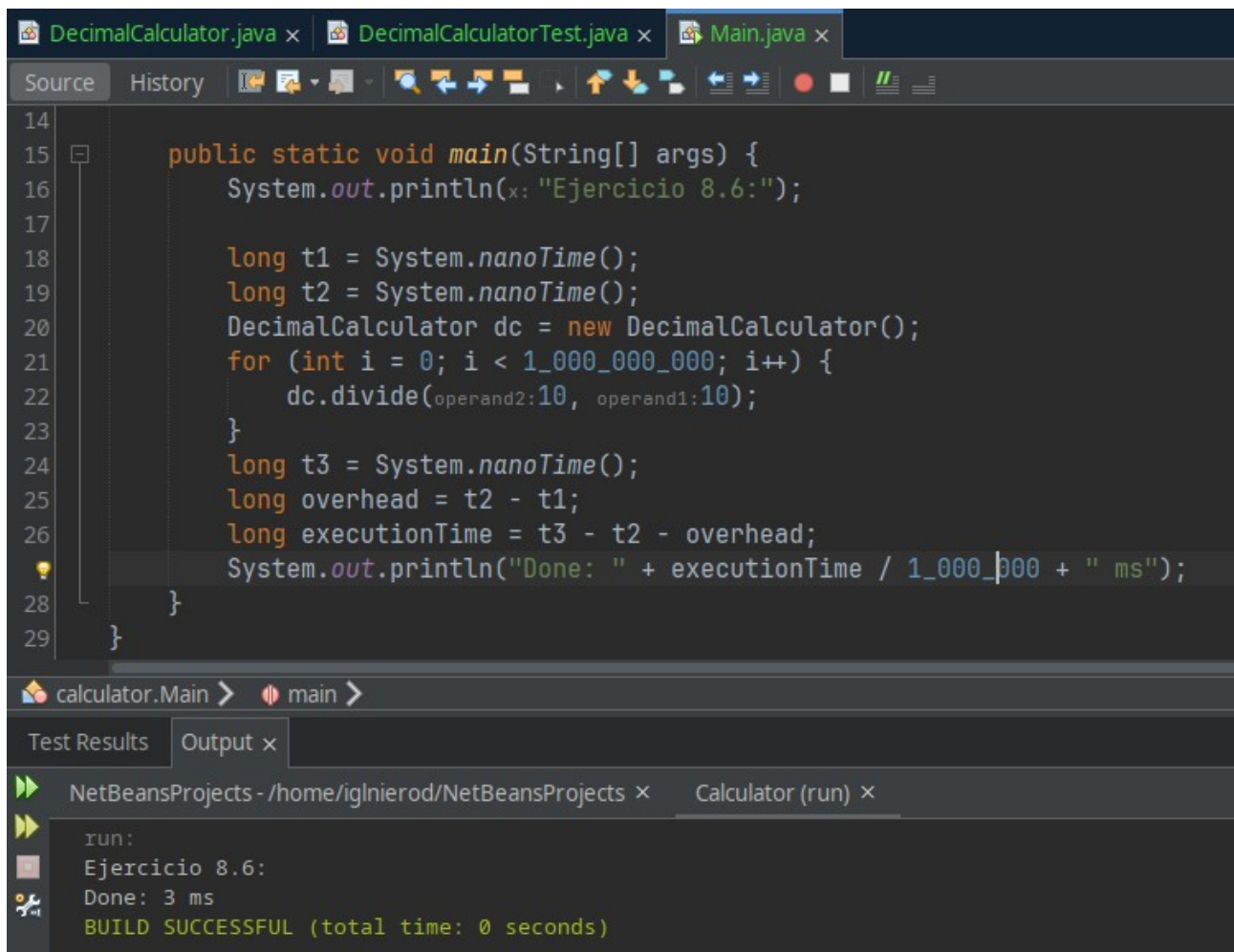
8.4)

No es necesario al no tener contexto, sería un malgasto de tiempo en este caso. Por lo contrario, es necesario en los casos en los que la seguridad del sistema ejecute varias operaciones por segundo de esta clase.

8.5)

No, en este caso no es necesario. Sería necesario si hubiese manera de dejar la aplicación en un estado vulnerable en el que el usuario pudiese acceder a cosas que no debería.

8.6)



The screenshot shows an IDE with three open files: `DecimalCalculator.java`, `DecimalCalculatorTest.java`, and `Main.java`. The `Main.java` file is active, displaying the following Java code:

```
14
15 public static void main(String[] args) {
16     System.out.println("Ejercicio 8.6:");
17
18     long t1 = System.nanoTime();
19     long t2 = System.nanoTime();
20     DecimalCalculator dc = new DecimalCalculator();
21     for (int i = 0; i < 1_000_000_000; i++) {
22         dc.divide(operand2:10, operand1:10);
23     }
24     long t3 = System.nanoTime();
25     long overhead = t2 - t1;
26     long executionTime = t3 - t2 - overhead;
27     System.out.println("Done: " + executionTime / 1_000_000 + " ms");
28 }
29 }
```

Below the code editor, the IDE shows the execution path: `calculator.Main > main >`. The `Test Results` and `Output` tabs are visible. The `Output` tab shows the following text:

```
run:
Ejercicio 8.6:
Done: 3 ms
BUILD SUCCESSFUL (total time: 0 seconds)
```