Apellidos, Nombre: Maceira Barca Xian

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 probas de seguridade. (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.

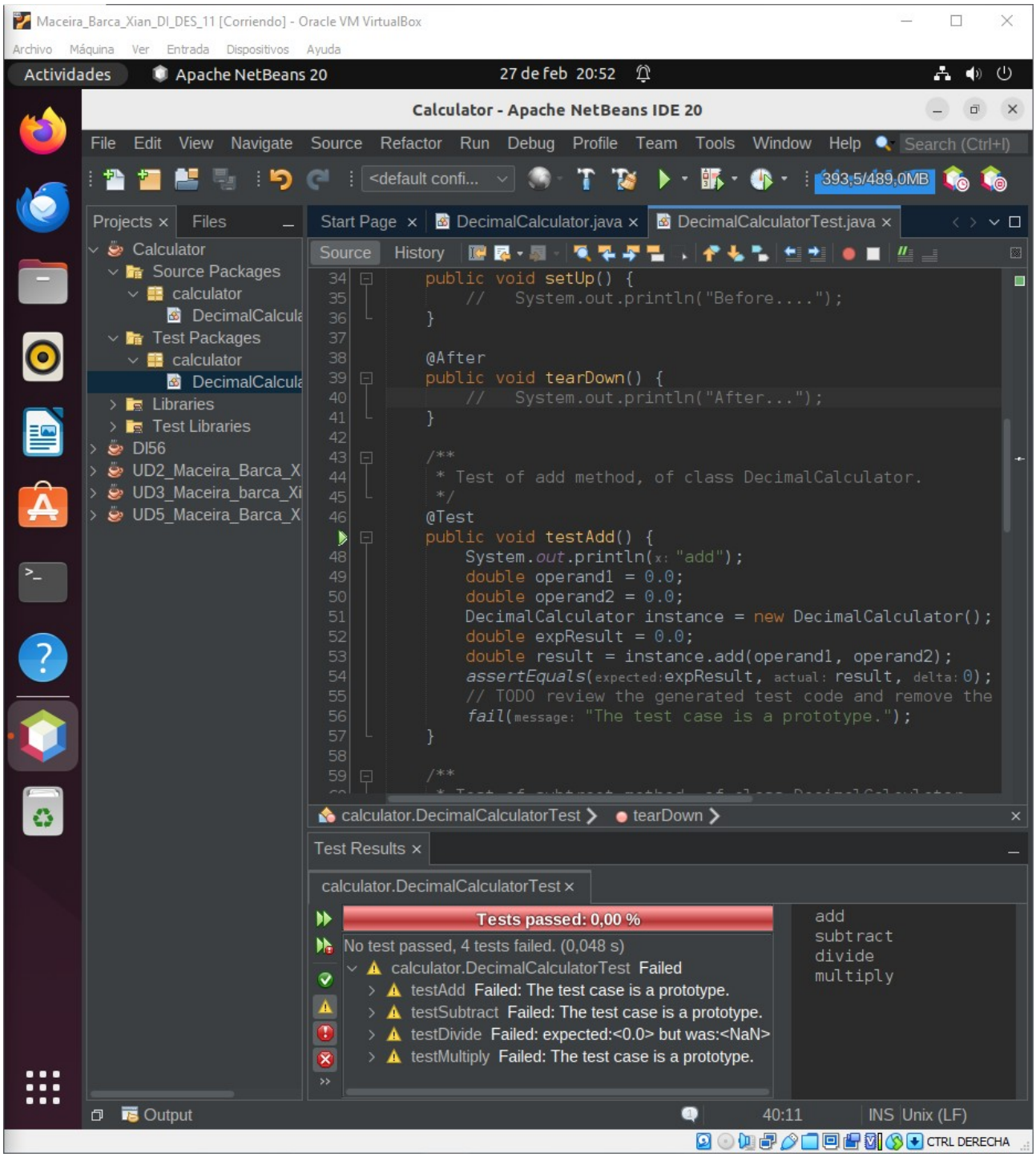
CA3.6 Realizáronse probas de uso de recursos por parte da aplicación. (10%)

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 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.**

```
/*
 * 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 maceira_barca_xian
 */
public class DecimalCalculatorTest {

    public DecimalCalculatorTest() {
    }
```

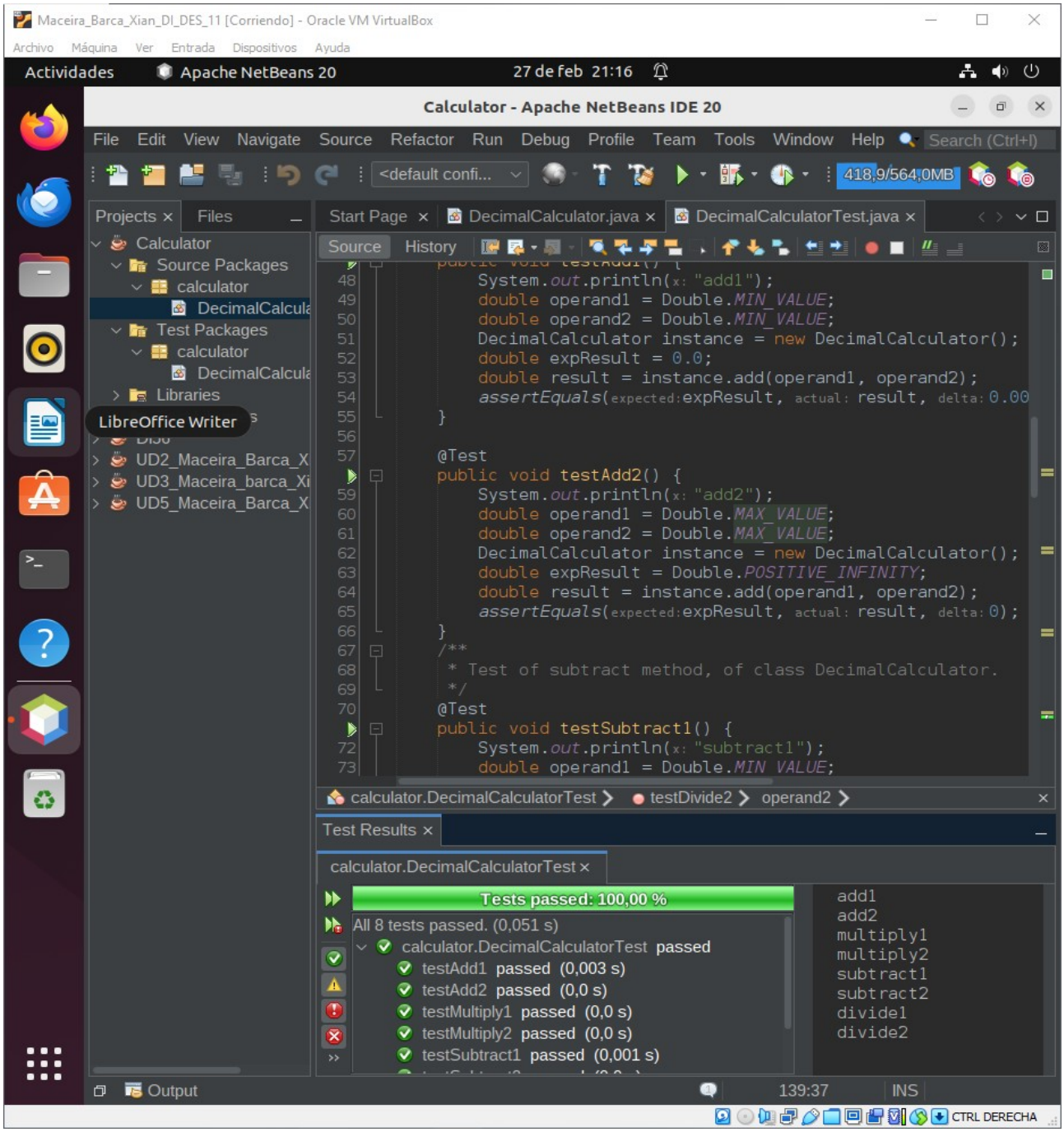
```
@BeforeClass
public static void setUpClass() {
    //    System.out.println("Before Class");
}

@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("add1");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 0.0;
```

```
    double result = instance.add(operand1, operand2);
    assertEquals(expResult, result, 0.000001);
}

@Test
public void testAdd2() {
    System.out.println("add2");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = Double.POSITIVE_INFINITY;
    double result = instance.add(operand1, operand2);
    assertEquals(expResult, result, 0);
}
/**
 * Test of subtract method, of class DecimalCalculator.
 */
@Test
public void testSubtract1() {
    System.out.println("subtract1");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 0.0;
    double result = instance.subtract(operand1, operand2);
    assertEquals(expResult, result, 0);

}

@Test
```

```
public void testSubtract2() {
    System.out.println("subtract2");
    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("multiply1");
    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
public void testMultiply2() {
    System.out.println("multiply2");
    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("divide1");
    double operand1 = Double.MIN_VALUE;
    double operand2 = Double.MIN_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1.0;
    double result = instance.divide(operand1, operand2);
    assertEquals(expResult, result, 0);

}

@Test
public void testDivide2() {
    System.out.println("divide2");
    double operand1 = Double.MAX_VALUE;
    double operand2 = Double.MAX_VALUE;
    DecimalCalculator instance = new DecimalCalculator();
    double expResult = 1.0;
    double result = instance.divide(operand1, operand2);
```

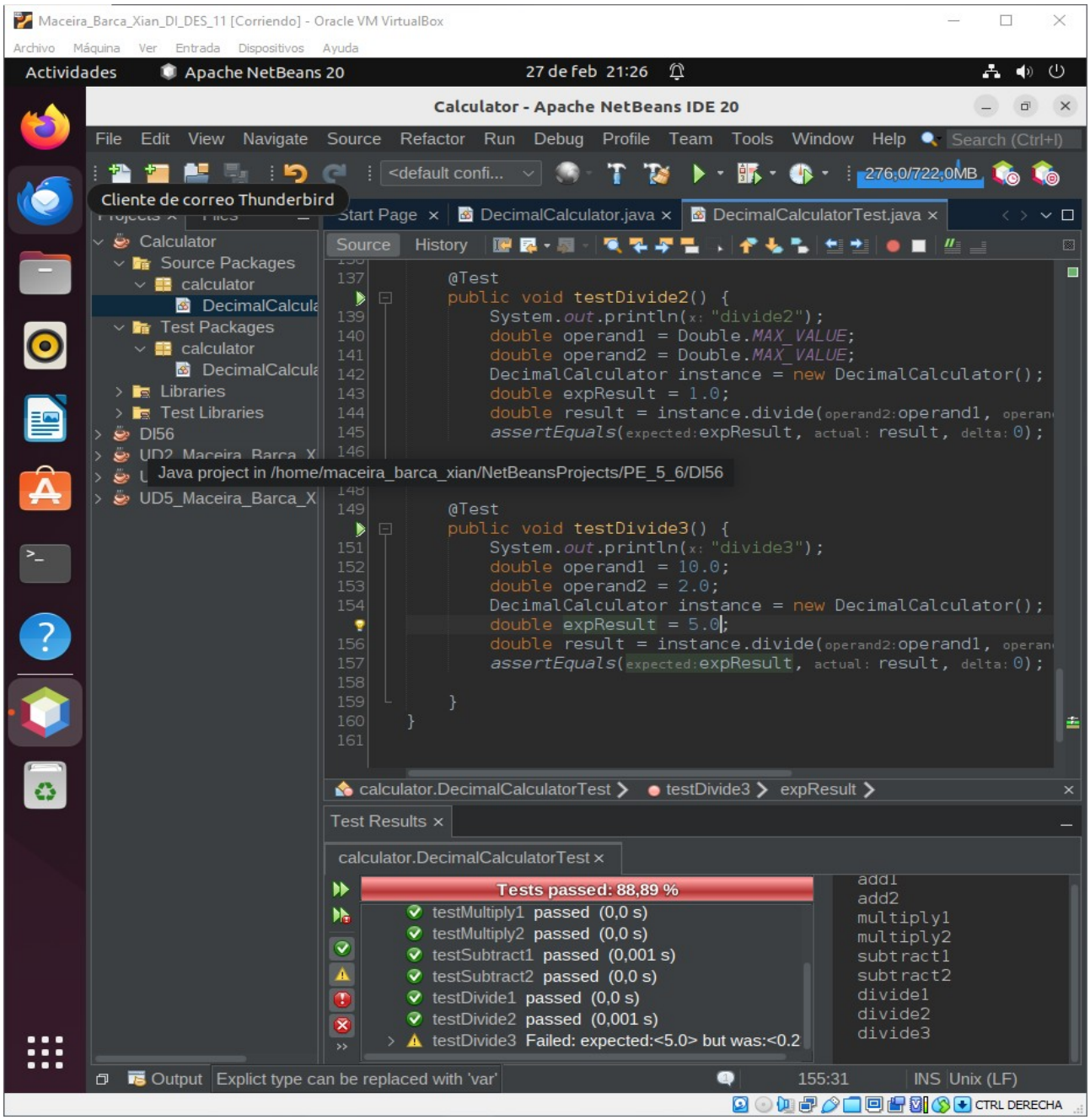
```
        assertEquals(expResult, result, 0);

    }

}
```

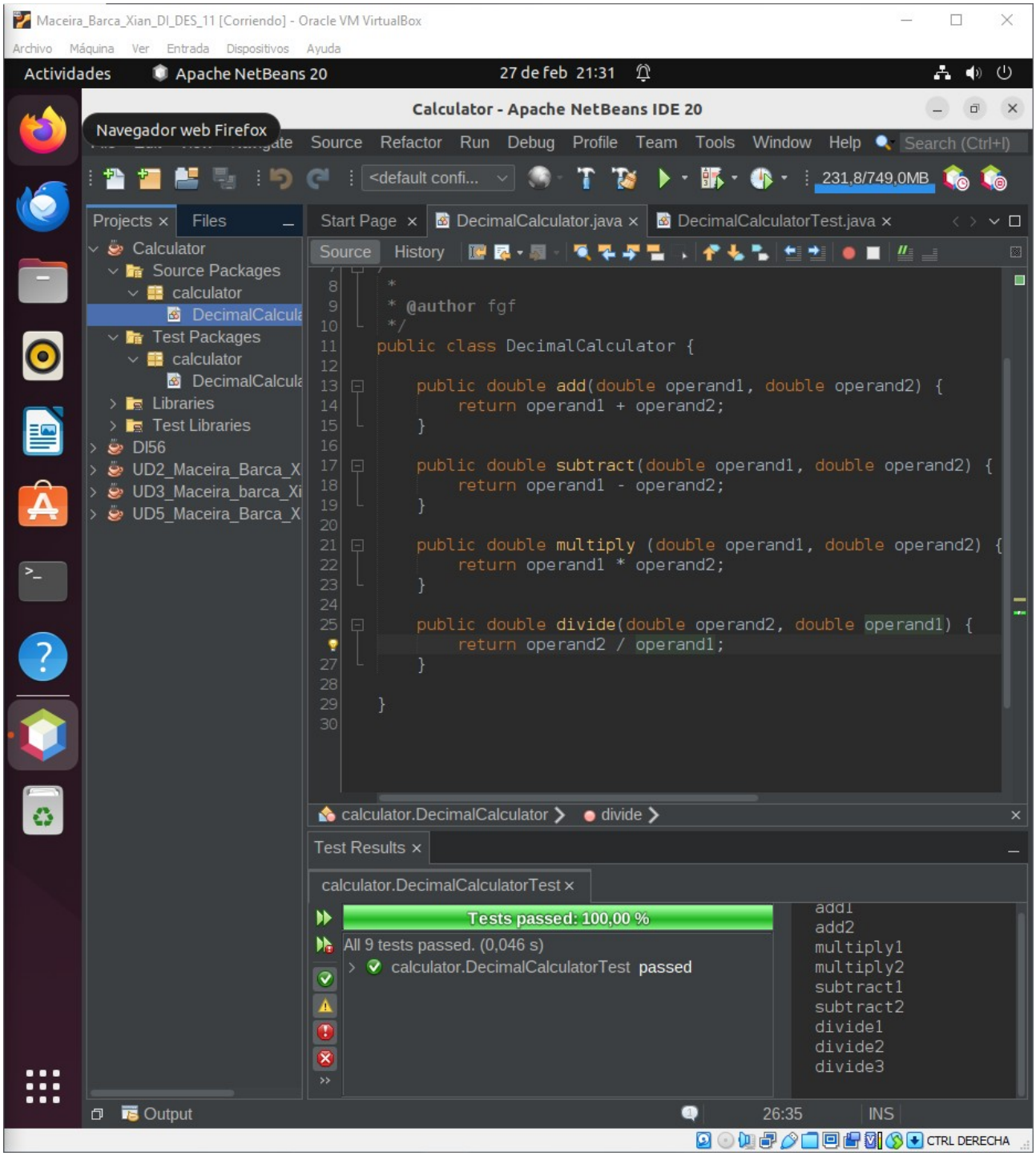


8.3 (Pruebas de regresión) Un desarrollador ha modificado el código del método divide de la siguiente forma:

```
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.




Maceira_Barca_Xian_DI_DES_11 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Actividades
Apache NetBeans 20
27 de feb 21:31
Navegador web Firefox
Calculator - Apache NetBeans IDE 20
Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
231,8/749,0MB
Projects
Files
Calculator
Source Packages
calculator
DecimalCalcula
Test Packages
calculator
DecimalCalcula
Libraries
Test Libraries
DI56
UD2_Maceira_Barca_X
UD3_Maceira_barca_Xi
UD5_Maceira_Barca_X
Start Page
DecimalCalculator.java
DecimalCalculatorTest.java
Source
History
* @author fgf
public class DecimalCalculator {
public double add(double operand1, double operand2) {
return operand1 + operand2;
public double subtract(double operand1, double operand2) {
return operand1 - operand2;
public double multiply (double operand1, double operand2) {
return operand1 * operand2;
public double divide(double operand2, double operand1) {
return operand2 / operand1;
calculator.DecimalCalculator
divide
Test Results
calculator.DecimalCalculatorTest
Tests passed: 100,00 %
All 9 tests passed. (0,046 s)
calculator.DecimalCalculatorTest passed
add1
add2
multiply1
multiply2
subtract1
subtract2
divide1
divide2
divide3
Output
26:35
INS
CTRL DERECHA

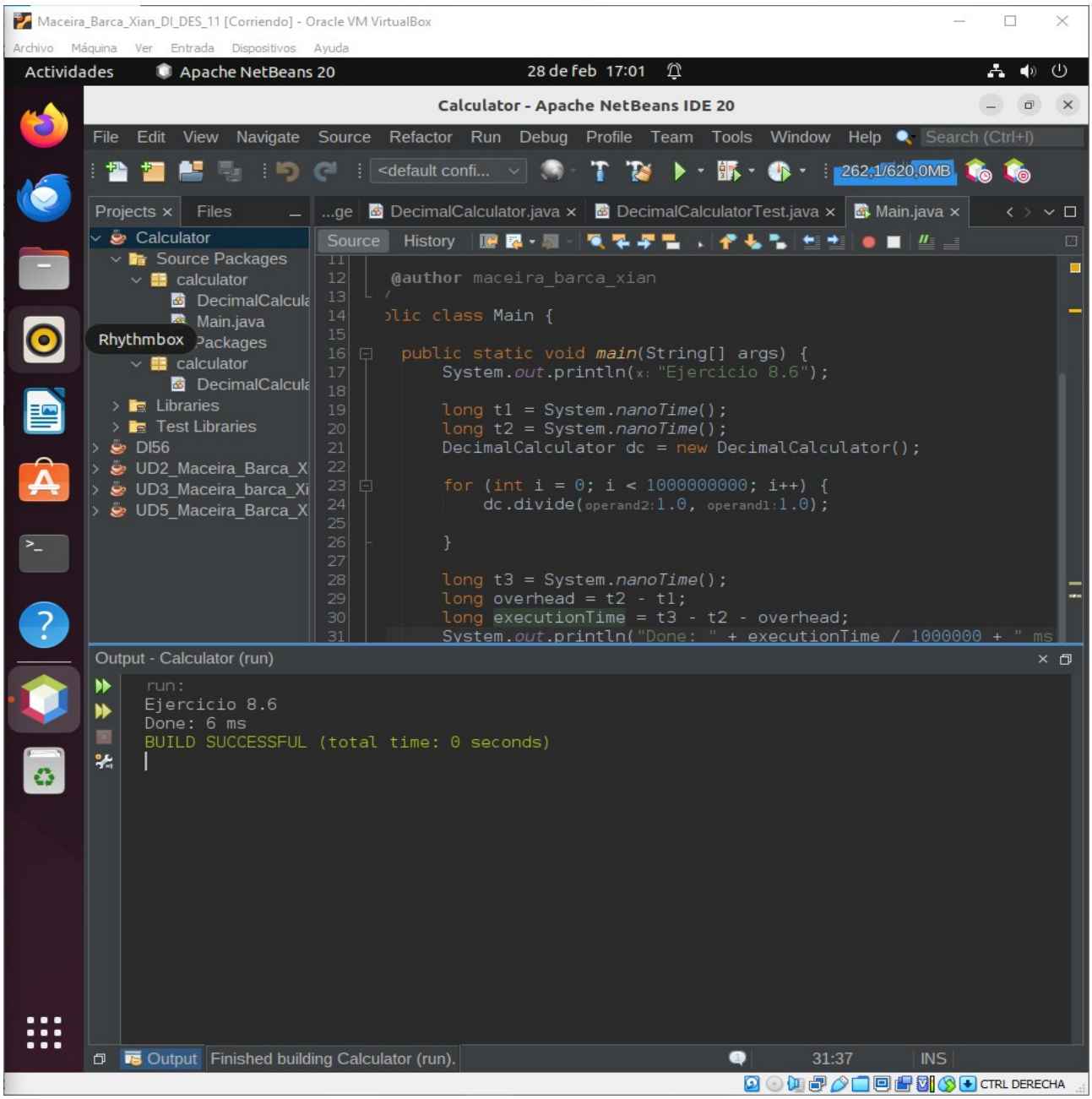
8.4 (Pruebas de volumen y estrés) Investiga si para el módulo dado sería necesario realizar pruebas de volumen y estrés. Investiga en qué casos sería necesario implementarlas.

En este caso no sería necesario hacer pruebas de volumen y estrés, esto solo sería necesario en casos donde sabemos que la aplicación va a ser usada por una gran cantidad de usuarios.

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

No, mientras no se nos proporcione un entorno en el cual sea necesario proteger la aplicación no se realizaran este tipo de pruebas, de esta manera se abaratan los costes.

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


Maceira_Barca_Xian_DI_DES_11 [Corriendo] - Oracle VM VirtualBox
Archivo Máquina Ver Entrada Dispositivos Ayuda
Actividades
Apache NetBeans 20
28 de feb 17:01
Calculator - Apache NetBeans IDE 20
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default confi...
262,1/620,0MB
Projects
Files
Calculator
Source Packages
calculator
DecimalCalcula
Main.java
Packages
calculator
DecimalCalcula
Libraries
Test Libraries
DI56
UD2_Maceira_Barca_X
UD3_Maceira_barca_Xi
UD5_Maceira_Barca_X
Rhythmbox
...ge
DecimalCalculator.java
DecimalCalculatorTest.java
Main.java
Source
History
@author maceira_barca_xian
olic class Main {
public static void main(String[] args) {
System.out.println(x: "Ejercicio 8.6");
long t1 = System.nanoTime();
long t2 = System.nanoTime();
DecimalCalculator dc = new DecimalCalculator();
for (int i = 0; i < 1000000000; i++) {
dc.divide(operand2:1.0, operand1:1.0);
}
long t3 = System.nanoTime();
long overhead = t2 - t1;
long executionTime = t3 - t2 - overhead;
System.out.println("Done: " + executionTime / 1000000 + " ms
Output - Calculator (run)
run:
Ejercicio 8.6
Done: 6 ms
BUILD SUCCESSFUL (total time: 0 seconds)
Output
Finished building Calculator (run).
31:37
INS
CTRL DERECHA