

**Enunciado.**

En BK han recibido algunas quejas de clientes sobre defectos en su software.

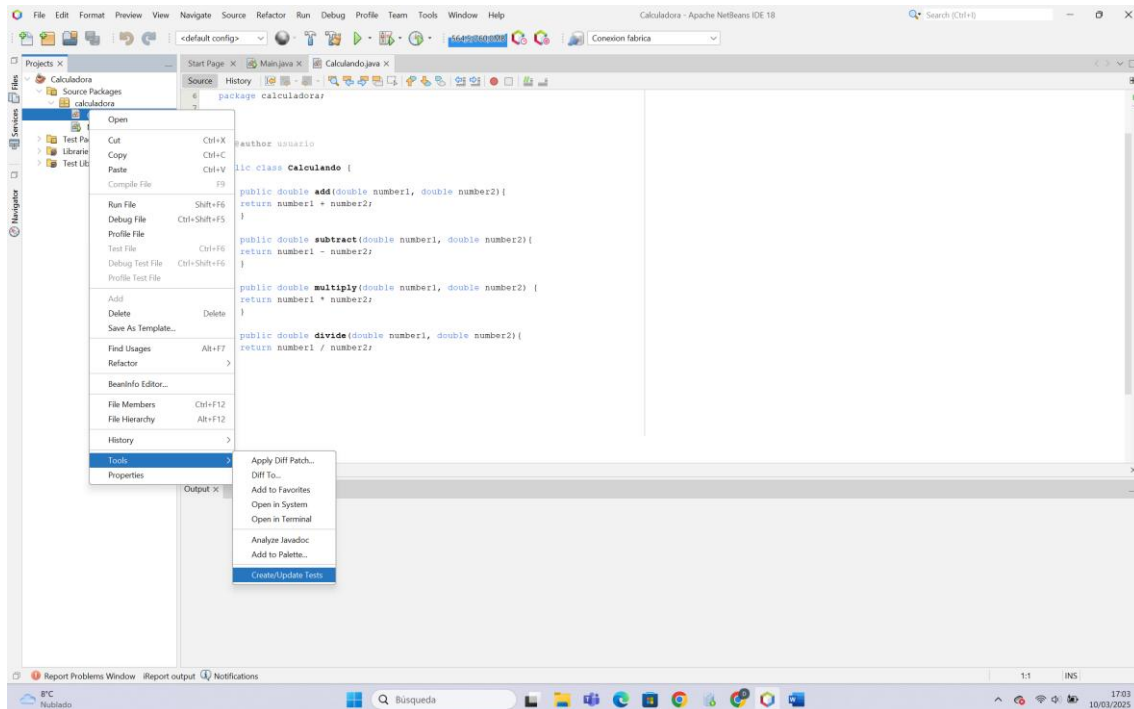
Ada está muy enfadada porque no se han seguido los protocolos de pruebas que la empresa tiene estandarizados. Por eso, en el nuevo proyecto que se va a desarrollar, tendrás que plantear la estrategia que asegure que los errores van a ser los mínimos posibles. Sabiendo que:

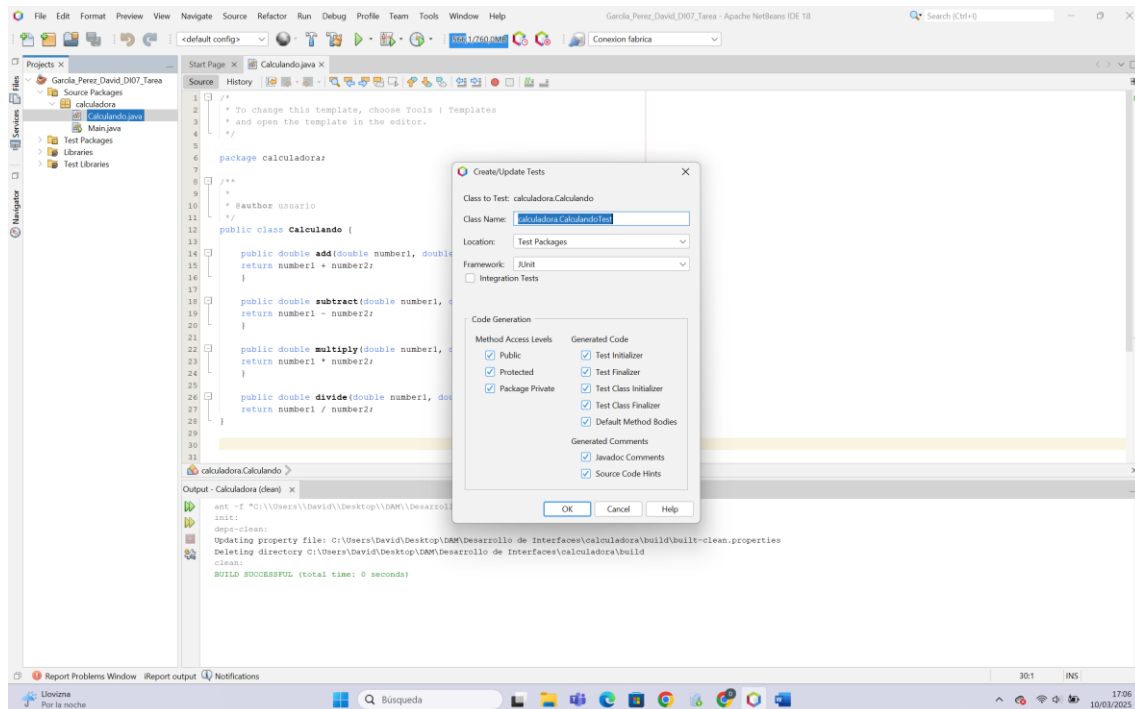
- Se trata de una aplicación desarrollada en Java
- Se van a realizar todas las pruebas vistas en la unidad.
- En principio, sólo se hará una versión por cada prueba.

Para desarrollar esta actividad necesitarás tener instalado NetBeans y JUnit. Durante el desarrollo del módulo, hemos estado trabajando con la versión 8.2 de NetBeans, el cuál ya trae incorporado Junit. En el caso de utilizar otra versión, asegúrate de tener instalado Junit en NetBeans.

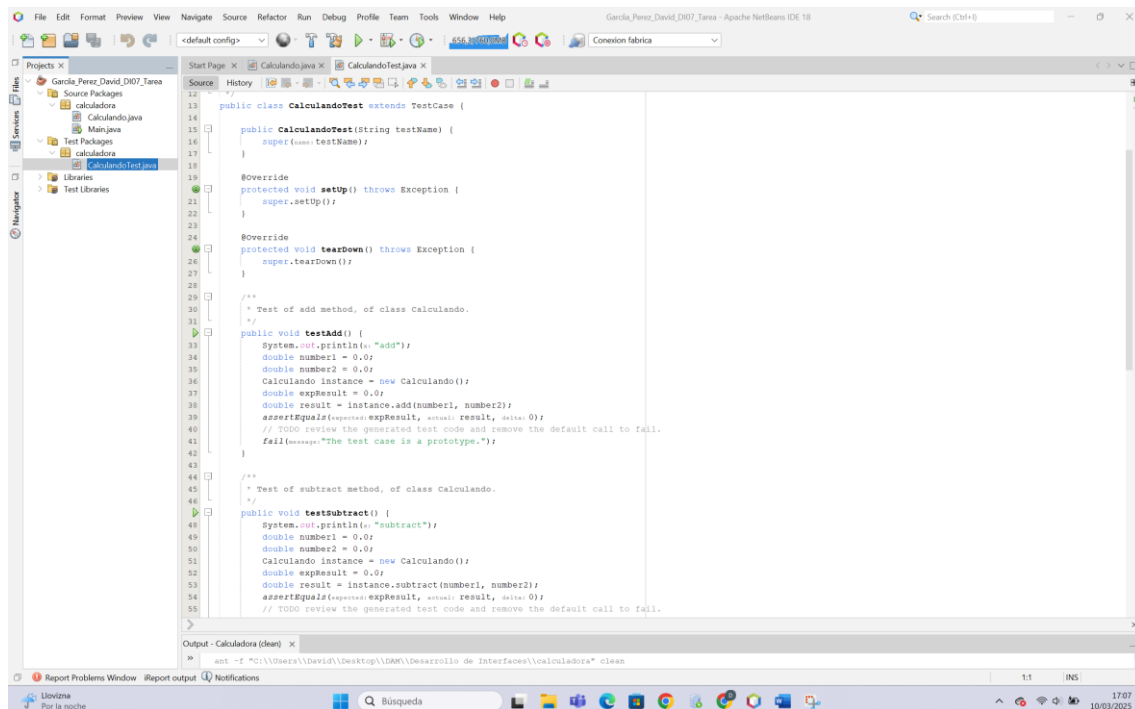
1. Más abajo puedes descargar el Proyecto Java, ábrelo con NetBeans. Observa los métodos definidos en la clase Calculando.java. Vamos a probar cada método de la clase con JUnit. Para ello, deberás de seleccionar la clase y en el menú Herramientas deberás de seleccionar la opción Create /update Tests. Nos aparecerá una ventana donde consta la clase a la que se le van a realizar las pruebas y la ubicación de las mismas. Seleccionaremos como Framework Junit y veremos que el código de la aplicación importa automáticamente el framework Junit. Como solución a este apartado deberás de aportar el código de la clase generado. (Calificación 1 punto)

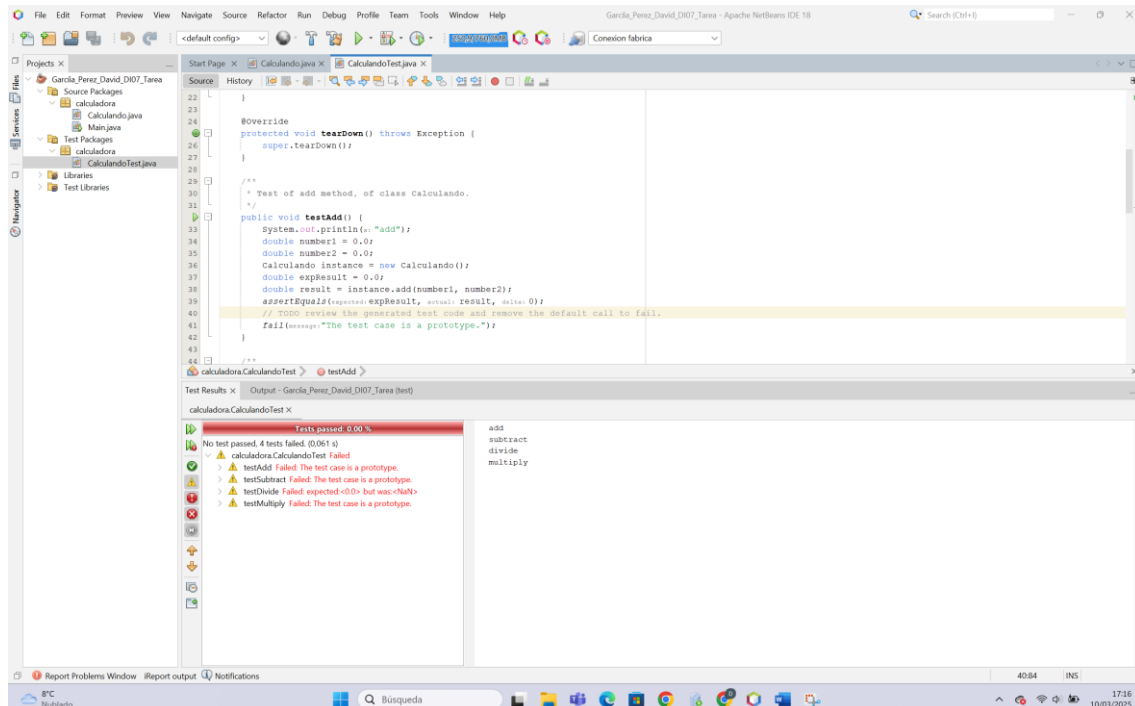
Seleccionamos la clase Calculando y pulsamos en el menú de Herramientas la opción Create/Update Tests:





Como podemos ver a continuación, se ha generado la clase CalculandoTest:



**2. Selecciona la nueva clase de pruebas que has generado. Ejecútala. Realiza una captura de la ventana Test results como solución a este apartado. (Calificación 1 punto)**

Al ejecutar esta clase podemos ver como los 4 test que se han creado (uno por cada método de la clase Calculando) fallan con las clases de prueba.

**3. Accede al código de la clase de pruebas y elimina las líneas:**

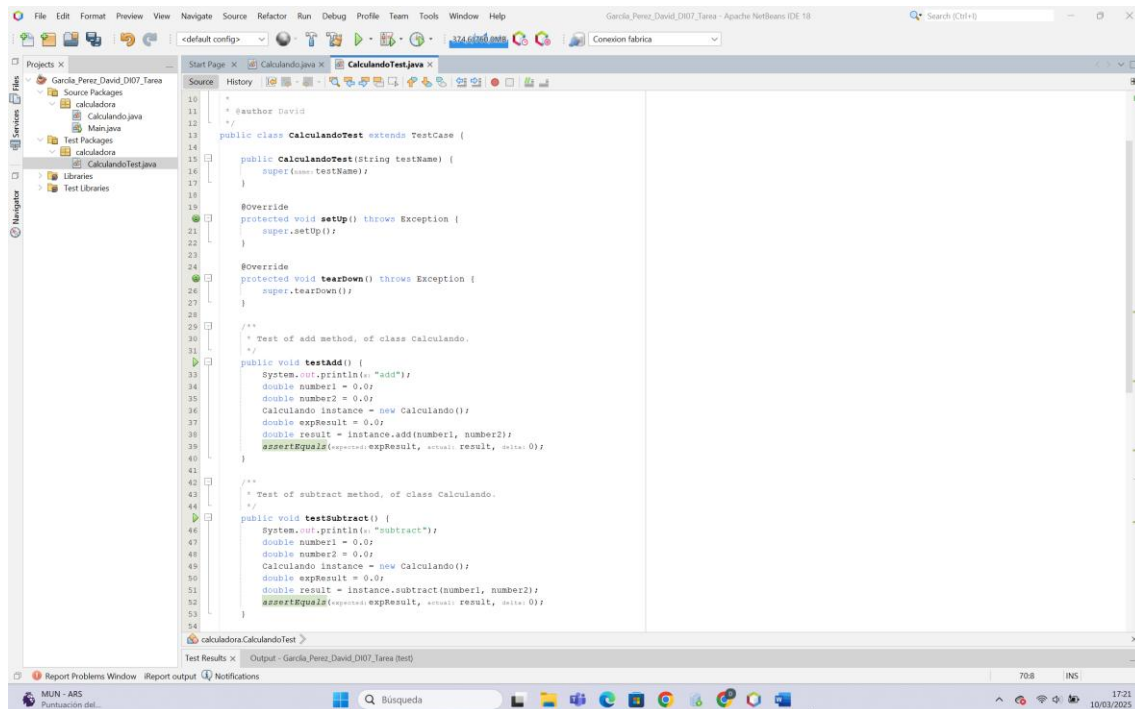
**// TODO review the generated test code and remove the default call to fail.**

**fail("The test case is a prototype.");**

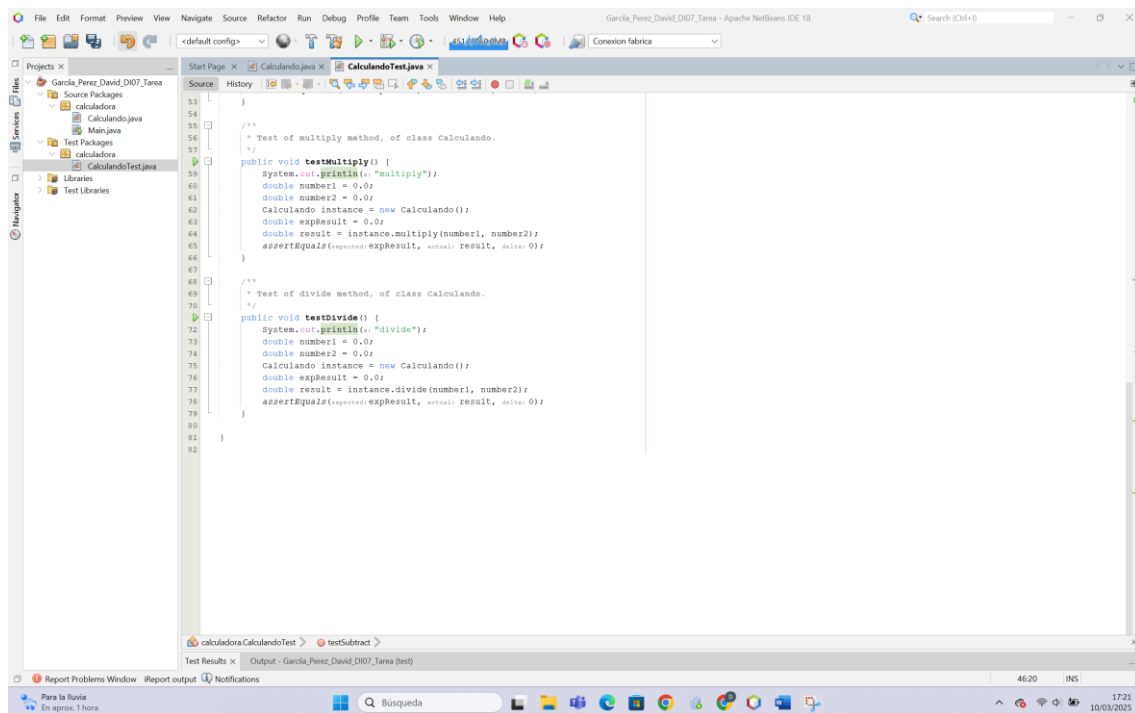
**que aparece al final de cada método. Como solución a este apartado deberás de entregar el código de la clase generado. (Calificación 1 punto)**

Como hemos visto anteriormente, Netbeans genera automáticamente el código de test de la clase Calculando que incluye los métodos add, subtract, multiply y divide.

Procedemos a eliminar las líneas indicadas:

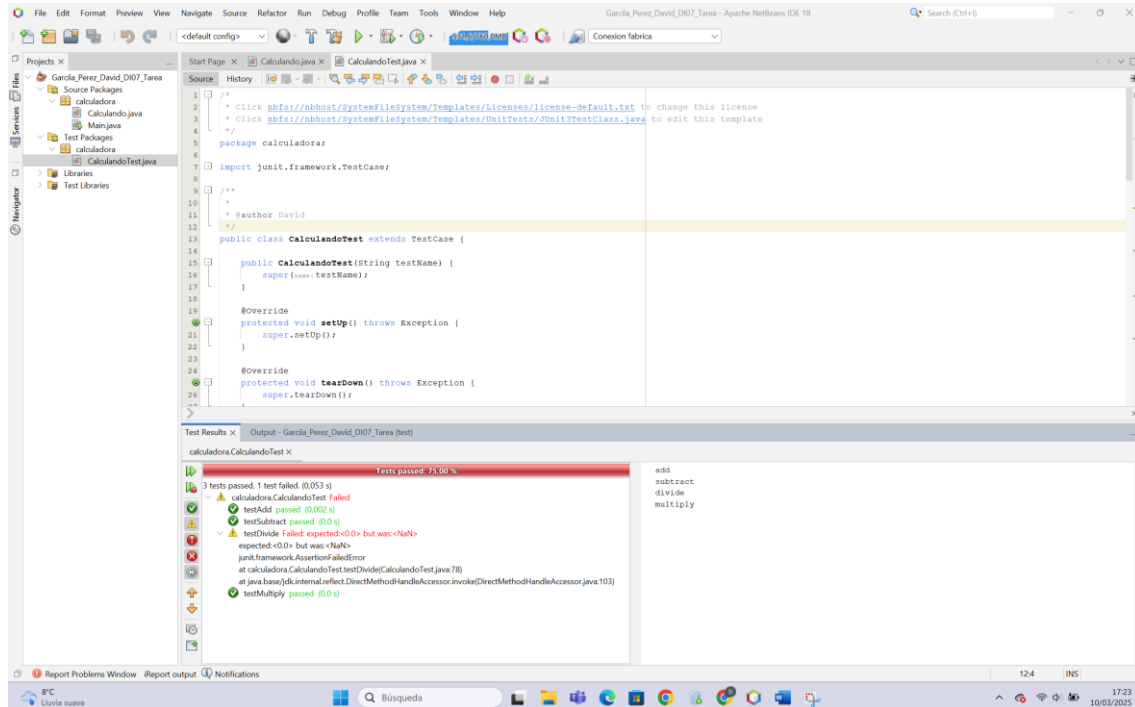


```
10  *
11  * @author David
12  */
13  public class CalculandoTest extends TestCase {
14
15      public CalculandoTest(String testName) {
16          super(testName);
17      }
18
19      @Override
20      protected void setUp() throws Exception {
21          super.setUp();
22      }
23
24      @Override
25      protected void tearDown() throws Exception {
26          super.tearDown();
27      }
28
29      /**
30       * Test of add method, of class Calculando.
31       */
32      public void testAdd() {
33          System.out.println("add");
34          double number1 = 0.0;
35          double number2 = 0.0;
36          Calculando instance = new Calculando();
37          double expectedResult = 0.0;
38          double result = instance.add(number1, number2);
39          assertEquals(expectedResult, result, 0.0);
40      }
41
42      /**
43       * Test of subtract method, of class Calculando.
44       */
45      public void testSubtract() {
46          System.out.println("subtract");
47          double number1 = 0.0;
48          double number2 = 0.0;
49          Calculando instance = new Calculando();
50          double expectedResult = 0.0;
51          double result = instance.subtract(number1, number2);
52          assertEquals(expectedResult, result, 0.0);
53      }
54  }
```



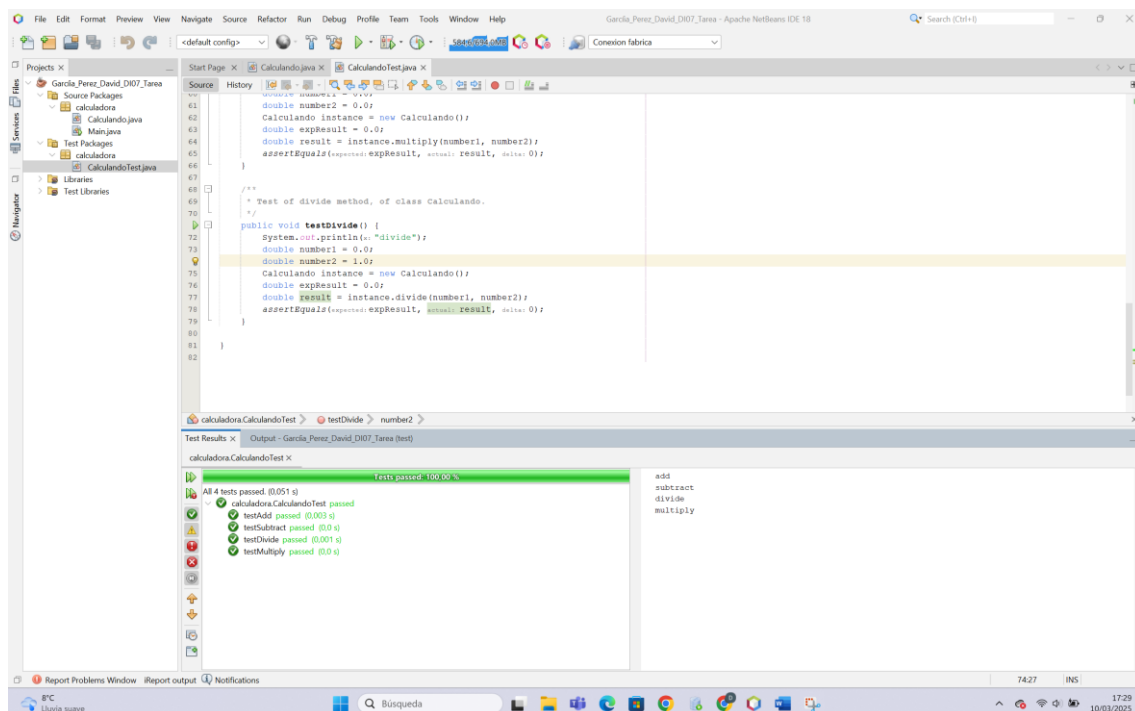
```
53  }
54
55      /**
56       * Test of multiply method, of class Calculando.
57       */
58      public void testMultiply() {
59          System.out.println("multiply");
60          double number1 = 0.0;
61          double number2 = 0.0;
62          Calculando instance = new Calculando();
63          double expectedResult = 0.0;
64          double result = instance.multiply(number1, number2);
65          assertEquals(expectedResult, result, 0.0);
66      }
67
68      /**
69       * Test of divide method, of class Calculando.
70       */
71      public void testDivide() {
72          System.out.println("divide");
73          double number1 = 0.0;
74          double number2 = 0.0;
75          Calculando instance = new Calculando();
76          double expectedResult = 0.0;
77          double result = instance.divide(number1, number2);
78          assertEquals(expectedResult, result, 0.0);
79      }
80
81
82  }
```

4-. Selecciona la clase de prueba y ejecútala de nuevo. Debes de corregir todos los errores asignándole valores a las variables. Al final, debes de conseguir que la ejecución de la prueba sea satisfactoria. Como solución a este apartado deberás de aportar el código de la clase de prueba una vez que ha sido modificado para conseguir que las pruebas fueran satisfactorias. (Calificación 1 punto)



Como podemos ver al ejecutar la clase de pruebas, nos da un fallo en la clase testDivide, ya que al dividir 0 entre 0, el valor que da es NaN.

Para que la prueba salga satisfactoria modificamos el código de esta clase para hacer que la división sea posible, en este caso modificamos los valores para que divida 0/1:



En esta ocasión todos los tests pasan sin problemas.

A continuación modificamos los test y añadimos más pruebas en cada uno de los métodos, con valores extremos que podrían hacer que fallen:

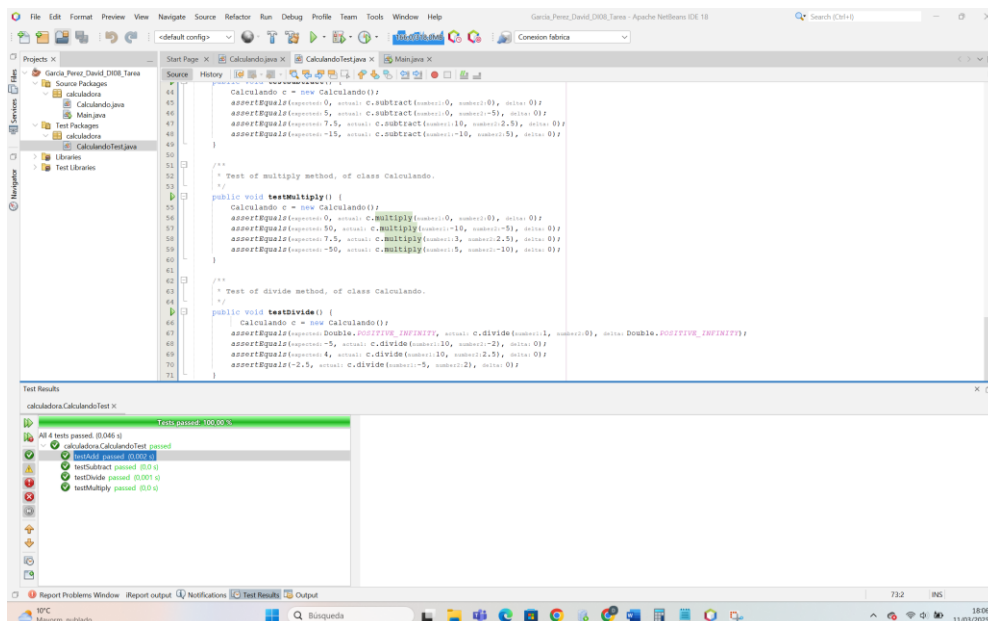
```
/**
 * Test of add method, of class Calculando.
 */
public void testAdd() {
    Calculando c = new Calculando();
    assertEquals(expected: 0, actual: c.add(number1:0, number2:0), delta: 0);
    assertEquals(expected: 5, actual: c.add(number1:-5, number2:10), delta: 0);
    assertEquals(expected: 2.5, actual: c.add(number1:5, -2.5), delta: 0);
    assertEquals(expected: -5, actual: c.add(-2.5, -2.5), delta: 0);
}

/**
 * Test of subtract method, of class Calculando.
 */
public void testSubtract() {
    Calculando c = new Calculando();
    assertEquals(expected: 0, actual: c.subtract(number1:0, number2:0), delta: 0);
    assertEquals(expected: 5, actual: c.subtract(number1:0, number2:-5), delta: 0);
    assertEquals(expected: 7.5, actual: c.subtract(number1:10, number2:2.5), delta: 0);
    assertEquals(expected: -15, actual: c.subtract(number1:-10, number2:5), delta: 0);
}

/**
 * Test of multiply method, of class Calculando.
 */
public void testMultiply() {
    Calculando c = new Calculando();
    assertEquals(expected: 0, actual: c.multiply(number1:0, number2:0), delta: 0);
    assertEquals(expected: 50, actual: c.multiply(number1:-10, number2:-5), delta: 0);
    assertEquals(expected: 7.5, actual: c.multiply(number1:3, number2:2.5), delta: 0);
    assertEquals(expected: -50, actual: c.multiply(number1:5, number2:-10), delta: 0);
}

/**
 * Test of divide method, of class Calculando.
 */
public void testDivide() {
    Calculando c = new Calculando();
    assertEquals(expected: Double.POSITIVE_INFINITY, actual: c.divide(number1:1, number2:0), delta: Double.POSITIVE_INFINITY);
    assertEquals(expected: -5, actual: c.divide(number1:10, number2:-2), delta: 0);
    assertEquals(expected: 4, actual: c.divide(number1:10, number2:2.5), delta: 0);
    assertEquals(-2.5, actual: c.divide(number1:-5, number2:2), delta: 0);
}
```

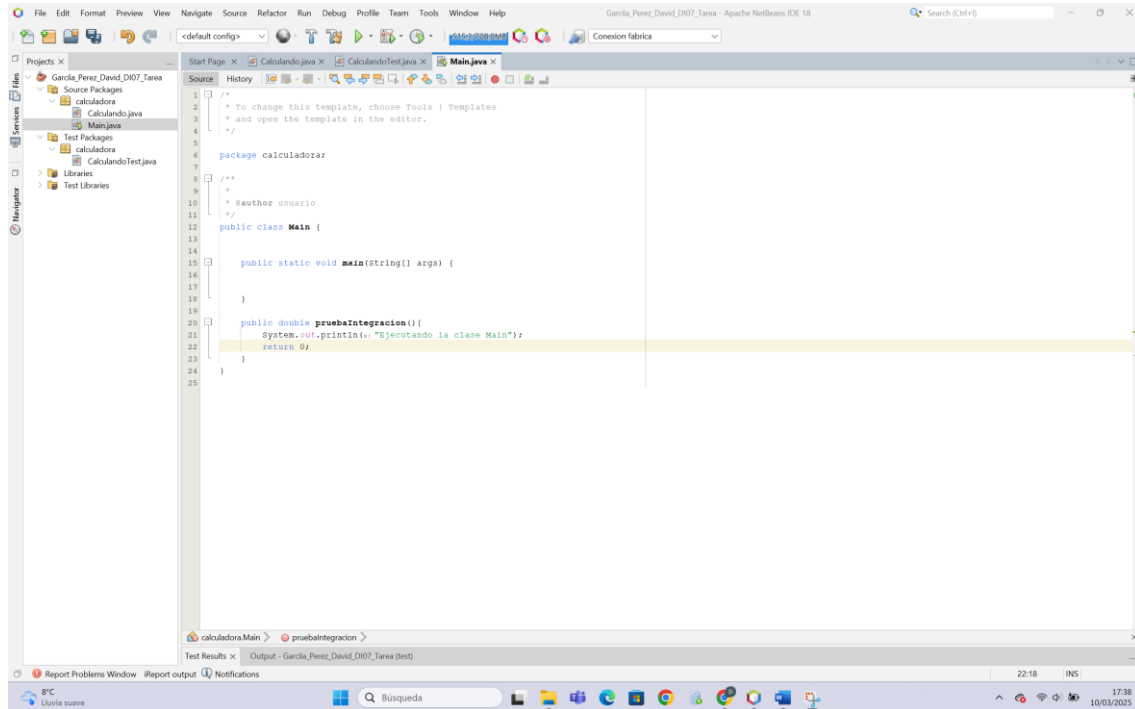
Al ejecutar los test, comprobamos que todos funcionan con los valores extremos introducidos.



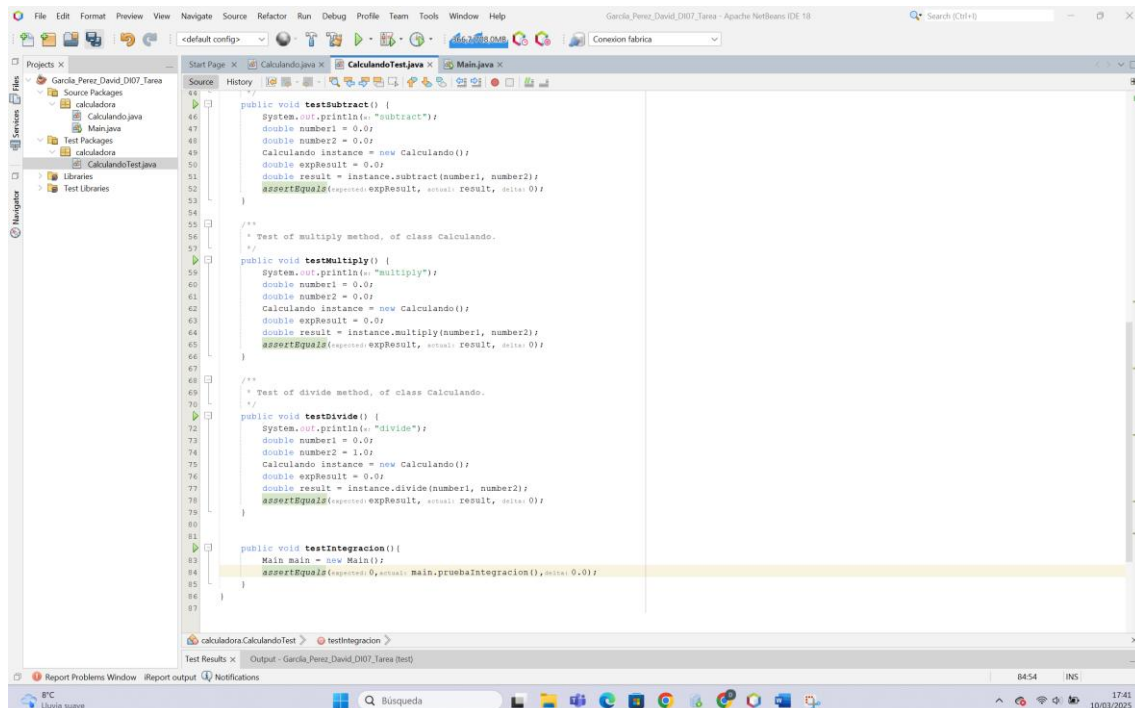
## 5-. Implementa la planificación de las pruebas de integración, sistema y regresión.(Calificación 2 puntos)

### - Pruebas de integración.

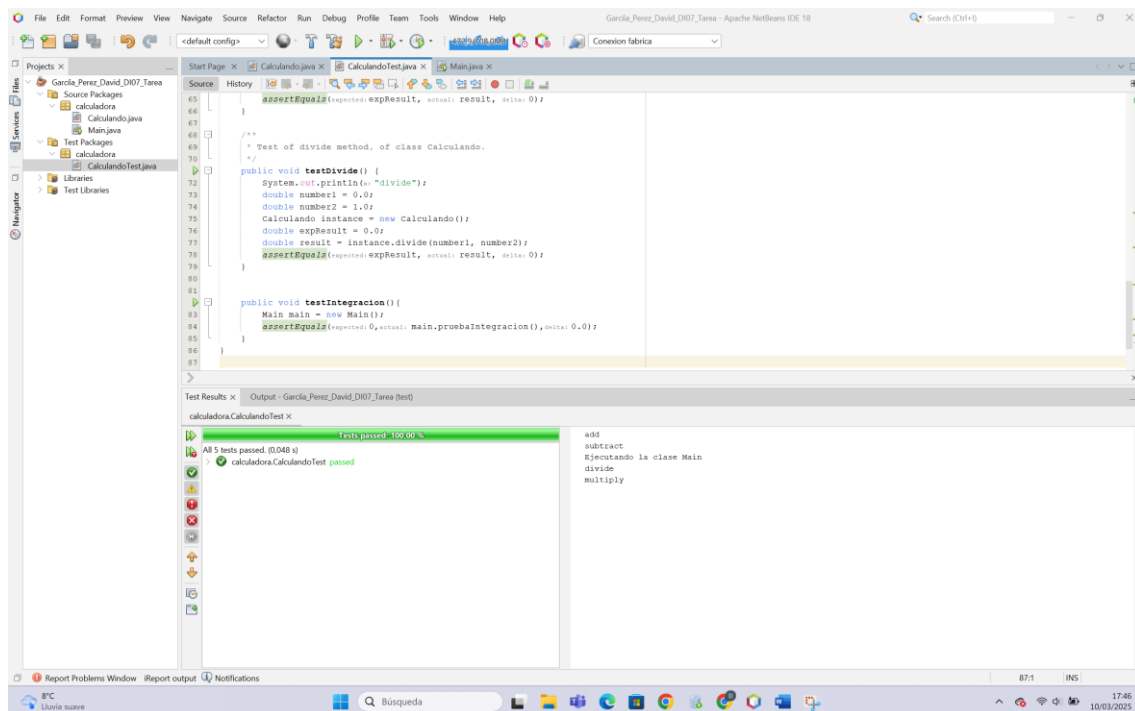
Para realizar las pruebas de integración crearemos un método dentro de la clase Main que instancie a la clase Calculando y use alguno de sus métodos, para hacer un test que compruebe que el resultado es el esperado.



Para poder hacer esta prueba de integración creamos el método testIntegracion().

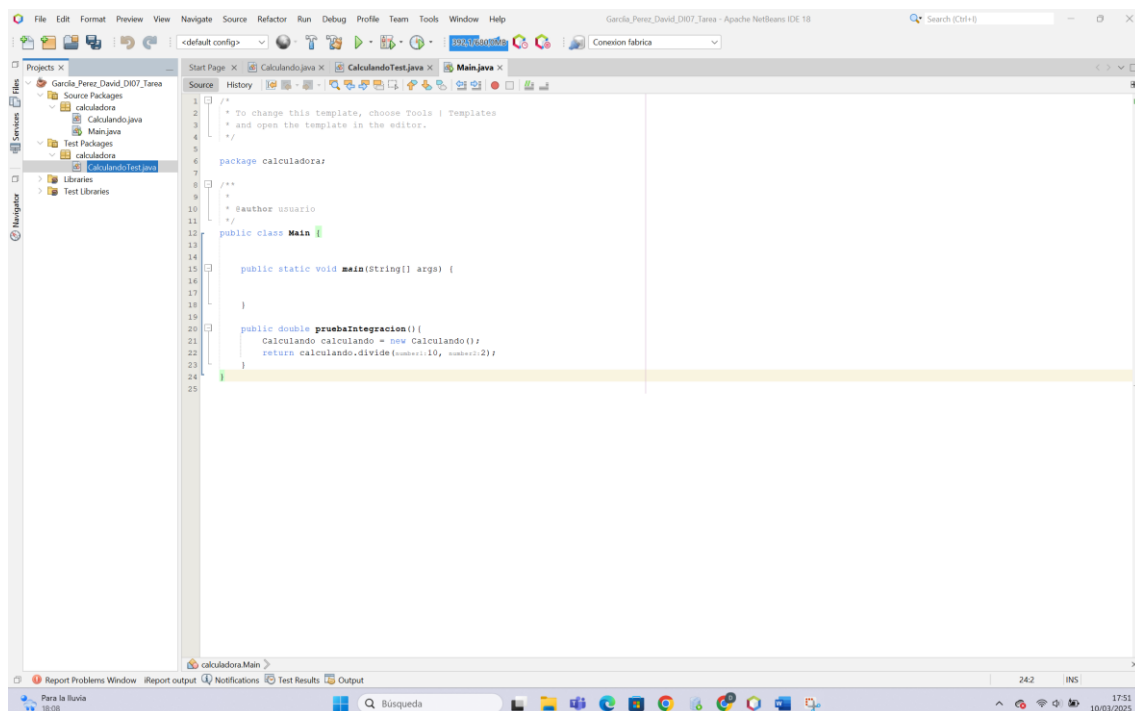


Para probarlo, hacemos que el método pruebaIntegracion() devuelva 0, y ejecutamos el main:

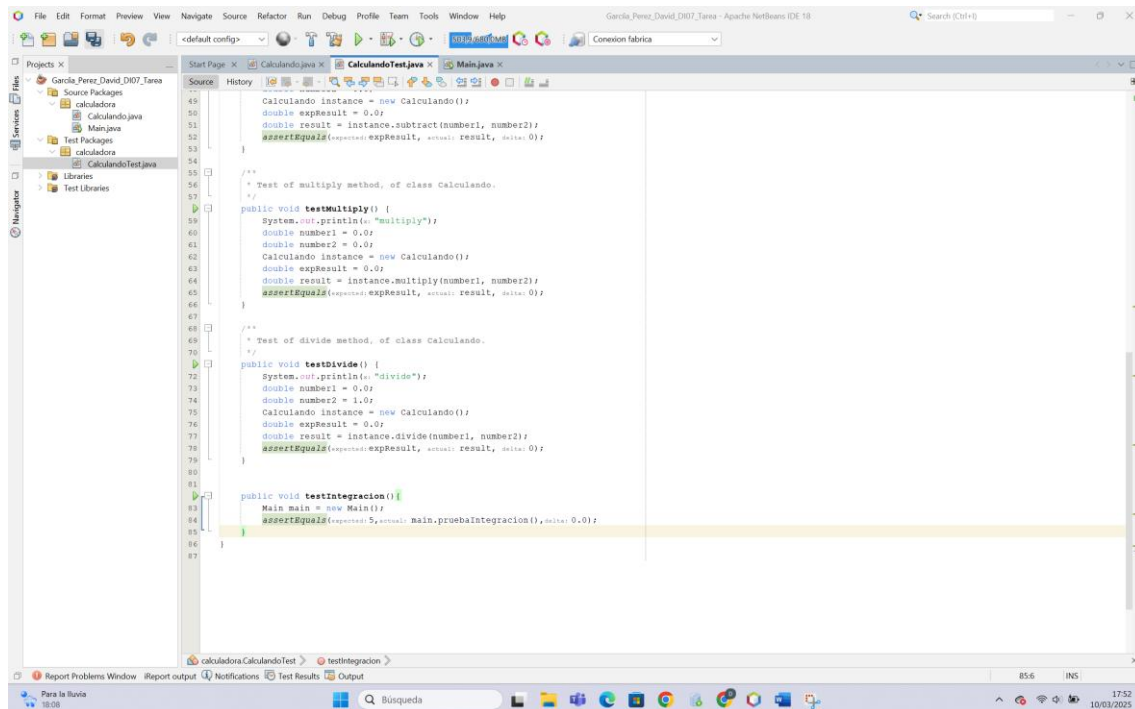


Como se puede observar, el método Main se ha ejecutado ya que se ha impreso por pantalla y el test ha dado el resultado esperado.

Ahora probamos llamando al método divide en el método pruebaIntegración() de la clase Main. Probaremos a hacer la división 10/2 y esperamos el resultado 5:



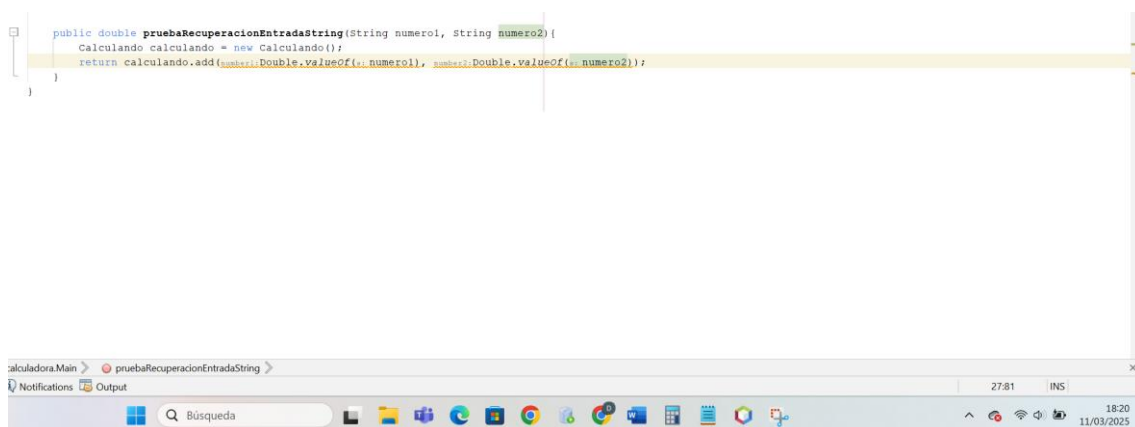




#### - Prueba de sistema.

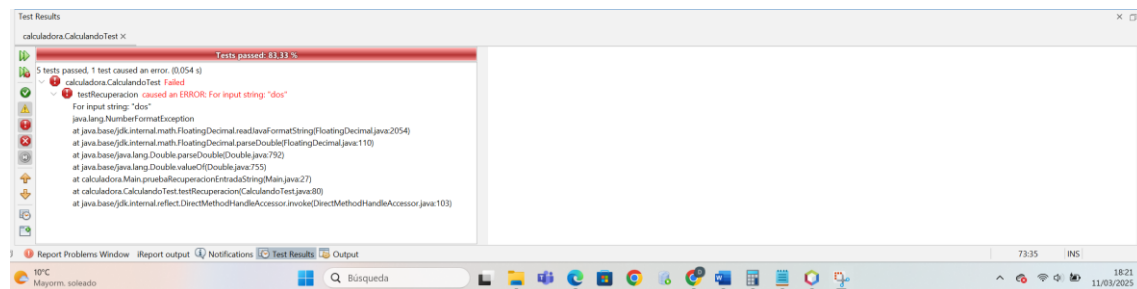
Esta prueba comprueba que el funcionamiento del programa en su conjunto cumple los requisitos de seguridad, fiabilidad, exactitud y velocidad. Con las pruebas unitarias hemos comprobado que la fiabilidad y exactitud de los cálculos de nuestro programa es correcta, además también hemos podido ver que los tiempos para ejecutar los tests han sido rápidos por lo que podemos considerar que cumple los requisitos de velocidad.

También se realiza a continuación la prueba de Recuperación forzando el fallo de software para comprobar si es capaz de recuperarse. Esto lo hacemos introduciendo entradas no esperadas por la aplicación, por ejemplo, valores de tipo String:

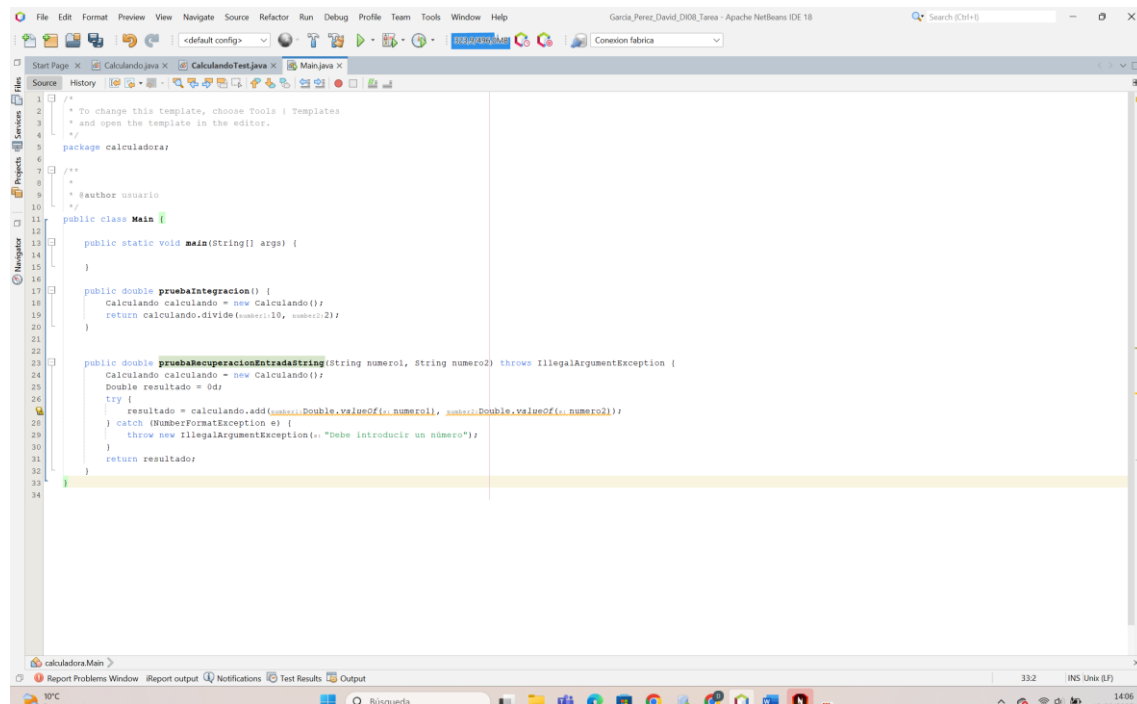


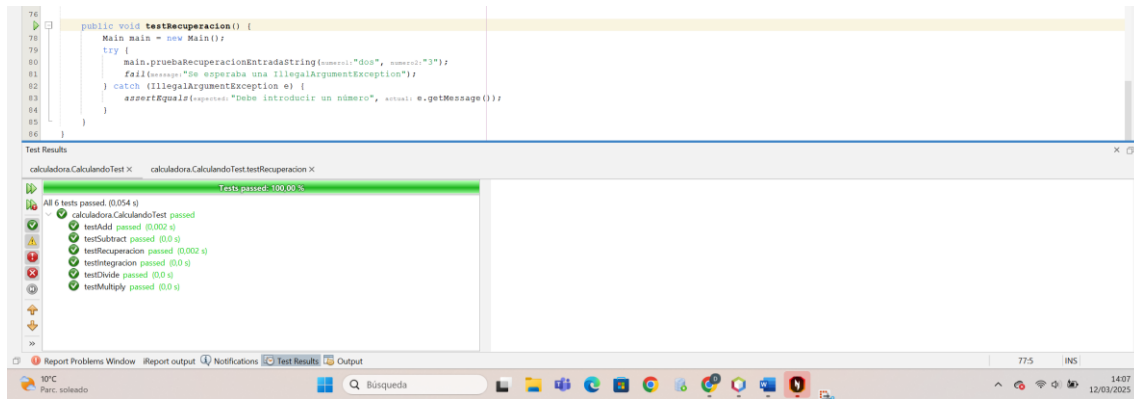


Como podemos comprobar, este test da un fallo al no reconocer la entrada String como válida:

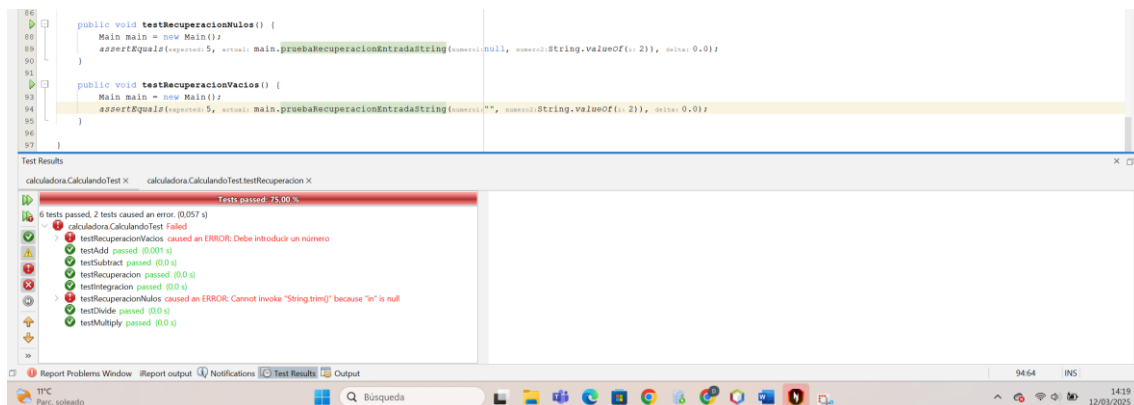


Para controlar las entradas, modificaremos el método main para que lance la excepción `NumberFormatException` con el mensaje “Se debe introducir un número”, para que en caso de fallo lance este mensaje por consola. También modificaremos el test `pruebaRecuperaciónEntradaString` con el propósito de verificar que el método `pruebaRecuperacionEntradaString()` de la clase `Main` maneje correctamente los casos en los que se ingresan valores no numéricos, lanzando la excepción esperada.

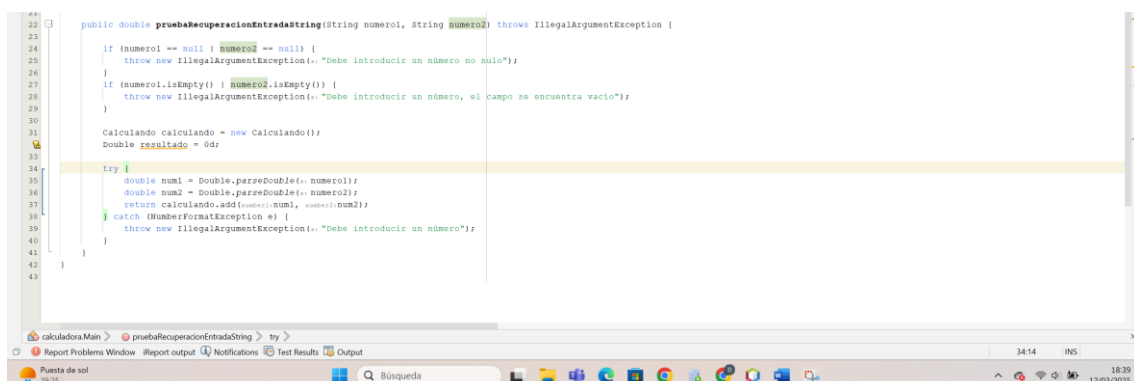


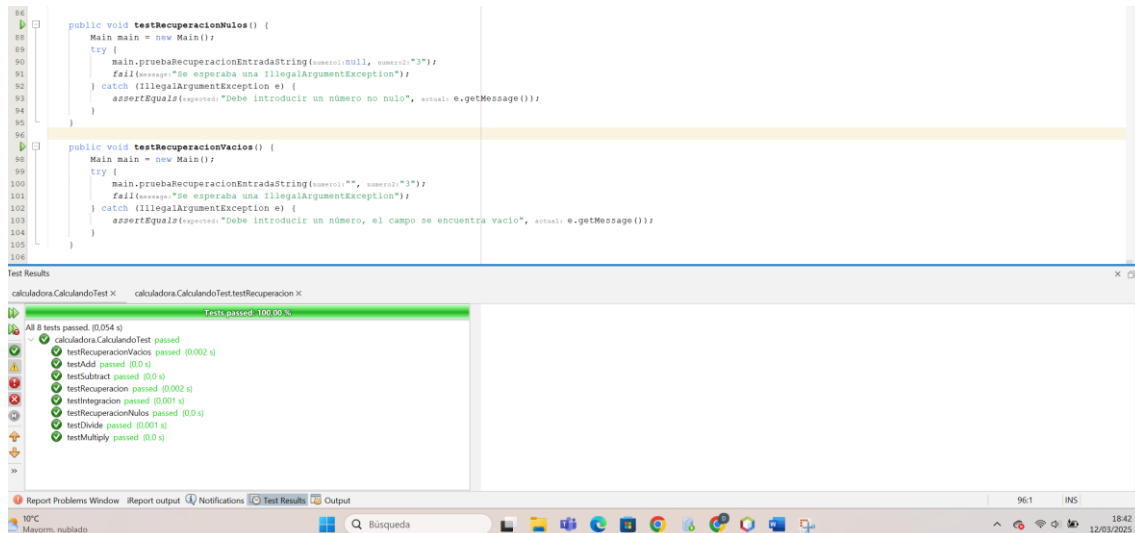


También se comprueba los casos en los que se introduce un número nulo o vacío, viendo que este caso lanza la excepción `NullPointerException`, lanzando el mensaje de que solo se admiten número estando en este caso el campo vacío o nulo:



Para controlar las entradas, modificaremos el método `main` incluyendo las condiciones de introducir un valor nulo o vacío para que amenje en tal caso las excepciones. También modificaremos `testRecuperacionVacios()` y `testRecuperacionNulos()` con el propósito de verificar que el método `pruebaRecuperacionEntradaString()` de la clase `Main` maneje correctamente los casos en los que se ingresan valores no correctos, lanzando la excepción esperada.





- **Pruebas de Regresión.**

Estas pruebas se han ido realizando indirectamente cuando hemos ejecutado los test Junit probando las diferentes partes del programa.

Por lo que podemos concluir estas pruebas se han pasado satisfactoriamente.

## 6-. Planifica las restantes pruebas, estableciendo qué parámetros se van a analizar. (Calificación 2 puntos).

- **Prueba de aceptación.**

Esta prueba valida si el comportamiento del programa cumple con todas sus características. Como en este caso su función es devolver el resultado esperado por el tipo de operación ejecutada por la aplicación, podemos concluir que estas pruebas han pasado satisfactoriamente, ya que hemos usado distintos valores para cada operación y los test han devuelto el resultado esperado.

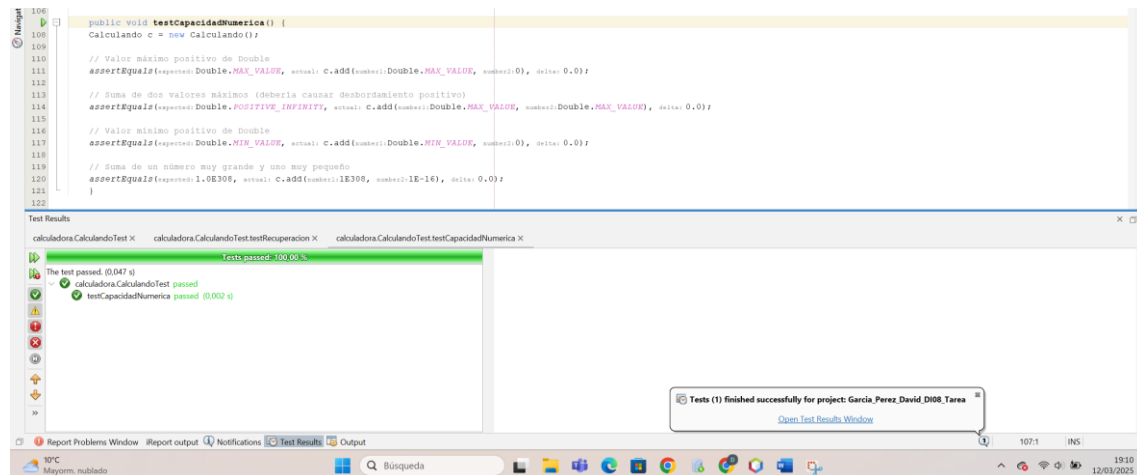
- **Prueba de capacidad.**

Las pruebas de capacidad (también llamadas pruebas de límites o pruebas de estrés) evalúan cómo se comporta un sistema bajo condiciones extremas, como valores numéricos muy grandes, operaciones repetitivas o cargas inusuales.

Esta prueba permite ver como se comporta el programa en situaciones extremas, por ejemplo, probando los límites de los tipos de datos, el manejo de desbordamientos y la estabilidad del sistema.

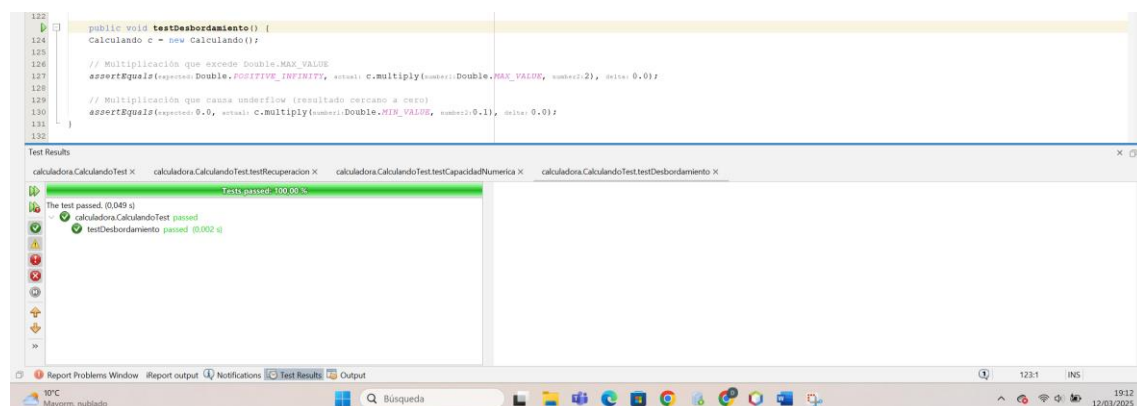
- **Prueba de capacidad numérica (límites de datos)**

Verifican el manejo de valores extremos (máximos, mínimos o valores que podrían causar desbordamientos)



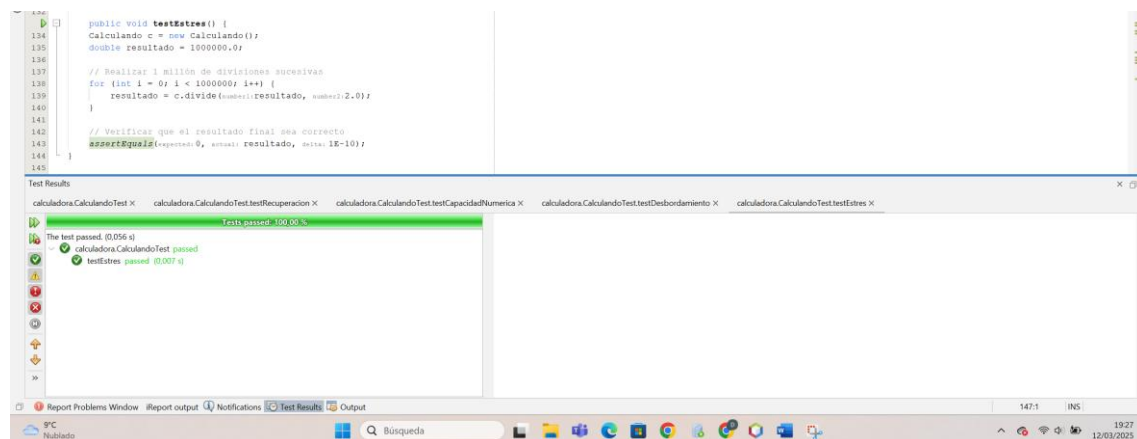
- **Prueba de desbordamiento**

Validan cómo maneja el código operaciones que superan los límites de los tipos de datos.



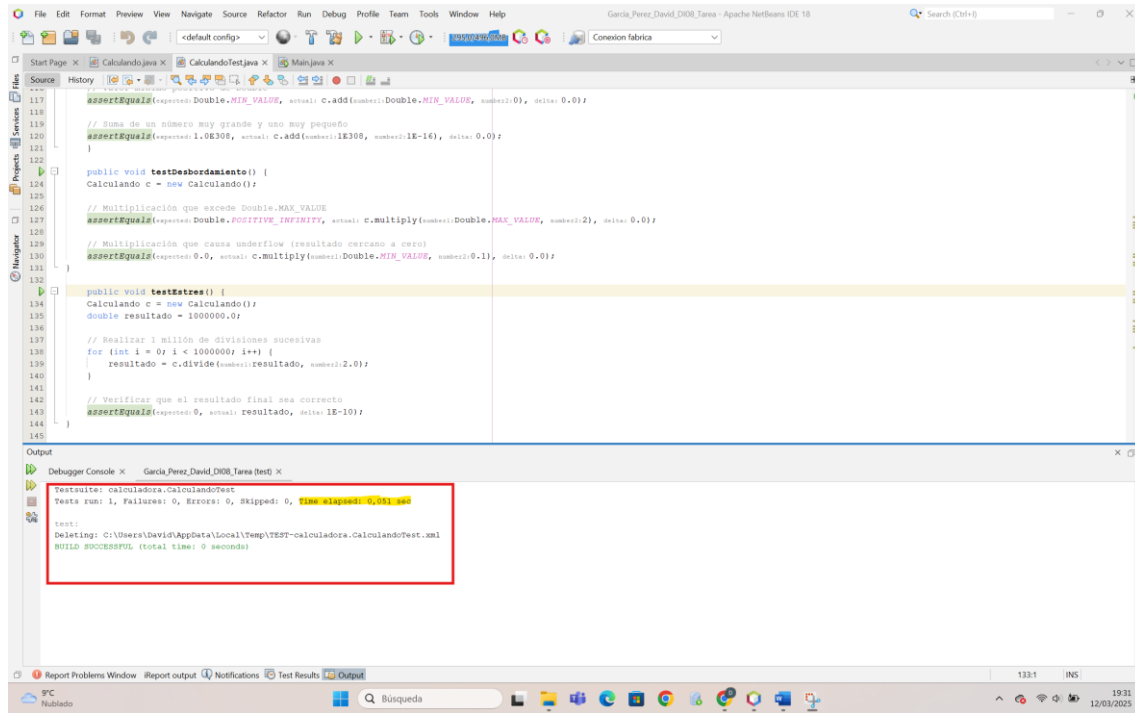
- **Prueba de estrés (operaciones repetitivas)**

Evalúan la estabilidad del sistema tras ejecutar múltiples operaciones consecutivas.



### - Prueba de rendimiento

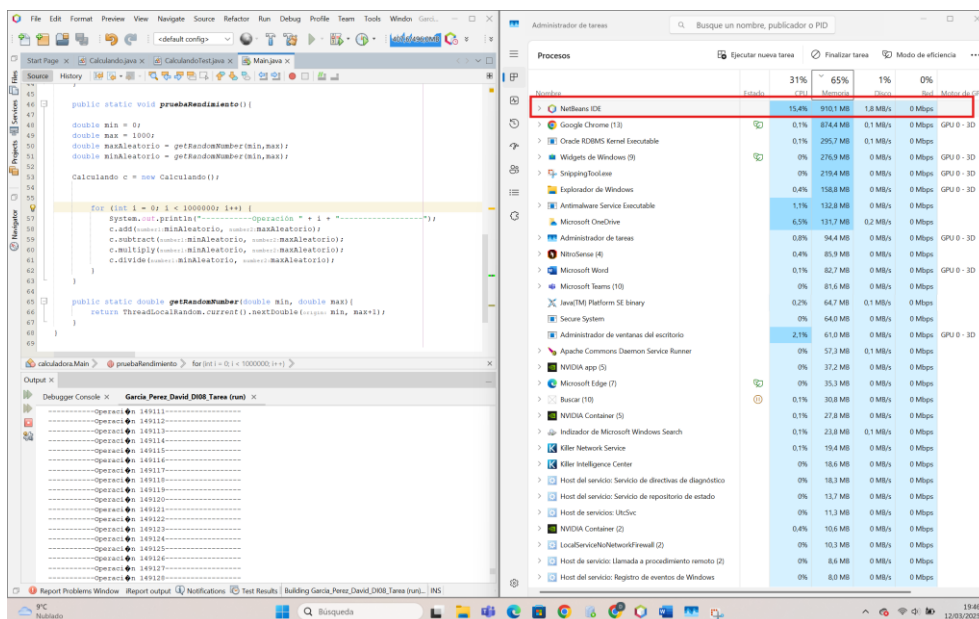
Esta prueba consiste en determinar el tiempo de respuesta de la aplicación, esta prueba puede derivarse la prueba de capacidad anterior. Viendo que esta se ha ejecutado en 0,051 segundos. Se puede considerar óptima ya que el programa ha conseguido realizar un millón de operaciones en tan solo este tiempo.



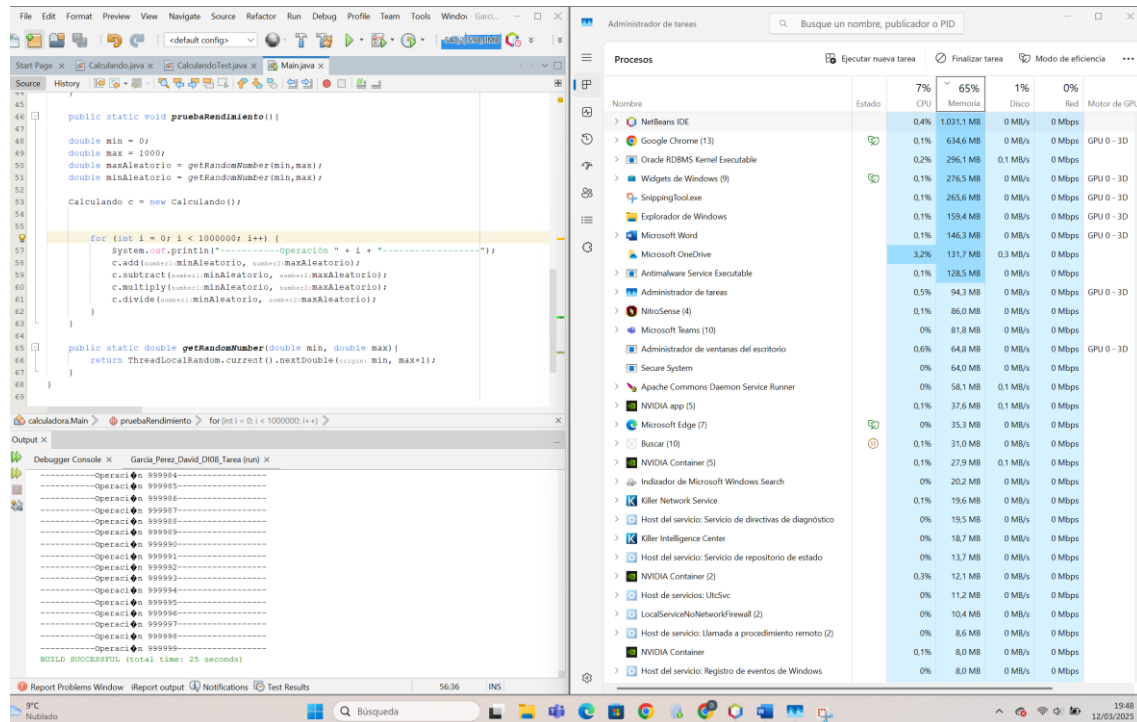
### - De uso de recursos.

Esta prueba se realiza monitoreando el sistema durante la ejecución del código para observar el consumo de recursos durante la ejecución del programa. Para ello crearemos una clase en Main que realice operaciones y las muestre por consola y a su vez monitorearemos el sistema:

#### ○ Consumo de recursos durante la ejecución del programa



## ○ Consumo de recursos al finalizar el programa:



Como hemos podido observar, la aplicación no usa más de un 15% de nuestra CPU al realizar 1 millón de operaciones, por lo que podemos decir que realiza un consumo eficiente de recursos de nuestro sistema.

**7-. Supuestas exitosas las pruebas, documenta el resultado (Calificación 2 puntos).**

Esta documentación se ha ido realizando en cada ejercicio.