Práctica 3.7 Pruebas Unitarias (CE3.g)

Nombre: Mateo Daniel Ortuño Ovando | Asignatura: Entornos de Desarrollo

Fecha de inicio: 27/01/2026 | Fecha de finalización: 07/02/2026

INDICE

1. INTRODUCCIÓN ..... Pág. 3

- 1.1. Objetivos de la práctica
- 1.2. Entorno de desarrollo utilizado (NetBeans y JUnit 5)

2. IMPLEMENTACIÓN DEL CÓDIGO ..... Pág. 4/5/6

- 2.1. Clase base: Calculadora.java
- 2.2. Clase de pruebas: CalculadoraTest.java

3. PRUEBAS UNITARIAS: RESULTADOS Y ANÁLISIS ..... Pág. 7/8/9/10

- 3.1. Ejecución inicial (- Barra Verde)
- 3.2. Evaluación con valores negativos: ¿Qué ocurre?
- 3.3. Simulación de errores: Fallos intencionados (Barra Roja)

4. CONCLUSIONES PERSONALES ..... Pág. 10(final)

1. INTRODUCCIÓN.

PRÁCTICA JUNIT – INTRODUCCIÓN A LAS PRUEBAS UNITARIAS.

Vas a crear una clase Java sencilla y una clase de pruebas con JUnit para comprobar que sus métodos funcionan correctamente.

1.1. Objetivos de la práctica.

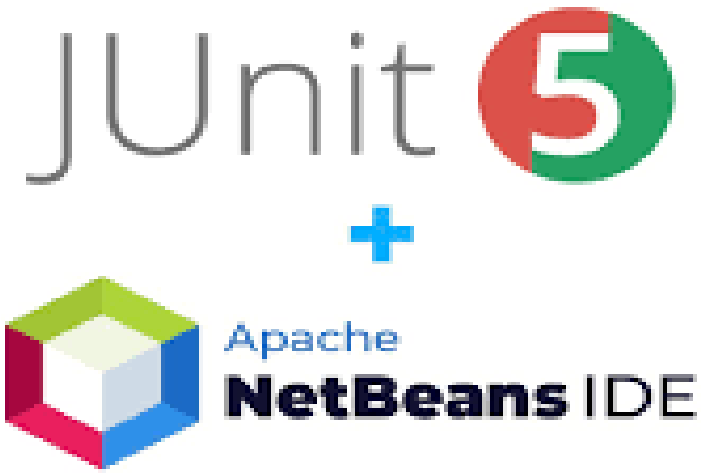
_Aprender a configurar, ejecutar e interpretar pruebas unitarias en un entorno real (NetBeans).

Que se hace en la practica?

1. Capturas de pantalla de ejecutar los Test. Explicar cuando un test sale bien que muestra Netbeans.
2. Añadir un método Test para probar valores negativos. ¿Qué ocurre?. Capturas y descríbelo,
3. Crear Test para que fallen intencionadamente. ¿Qué ocurre?. Capturas y descríbelo,

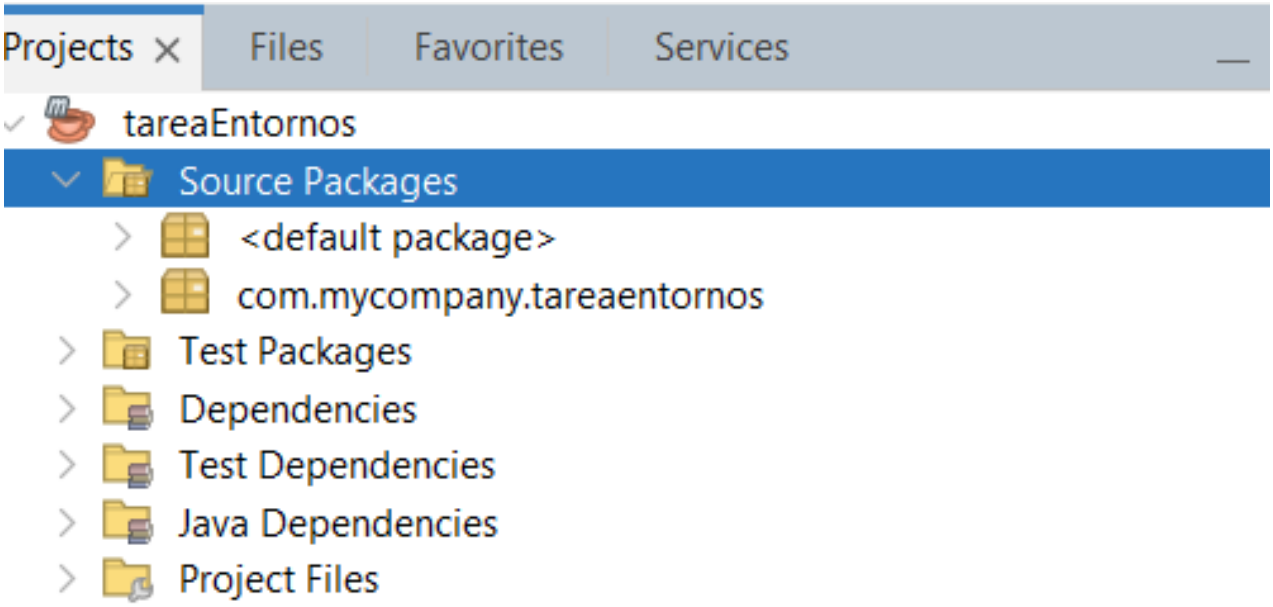
1.2. Entorno de desarrollo utilizado (NetBeans y JUnit 5).

_Para esta practica se va a utilizar el IDE Netbeans, con las funciones que ya tiene implementada (JUnit5), por lo que no es necesario instalar nada mas.

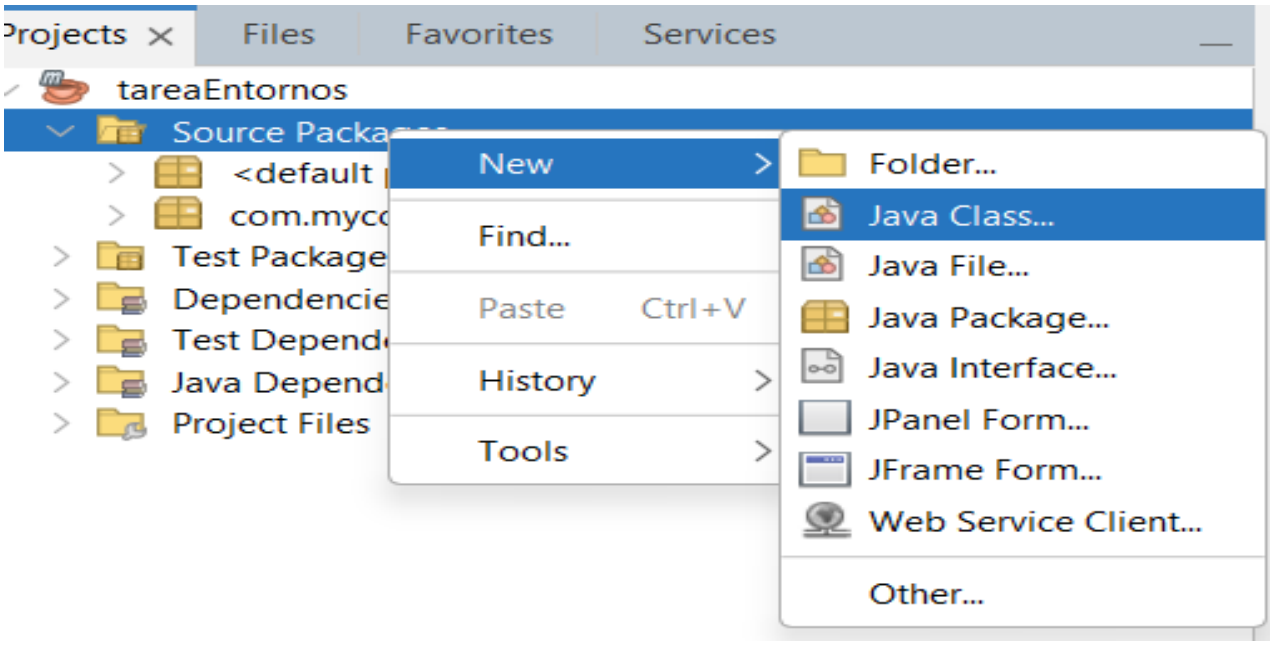


2. IMPLEMENTACIÓN DEL CÓDIGO

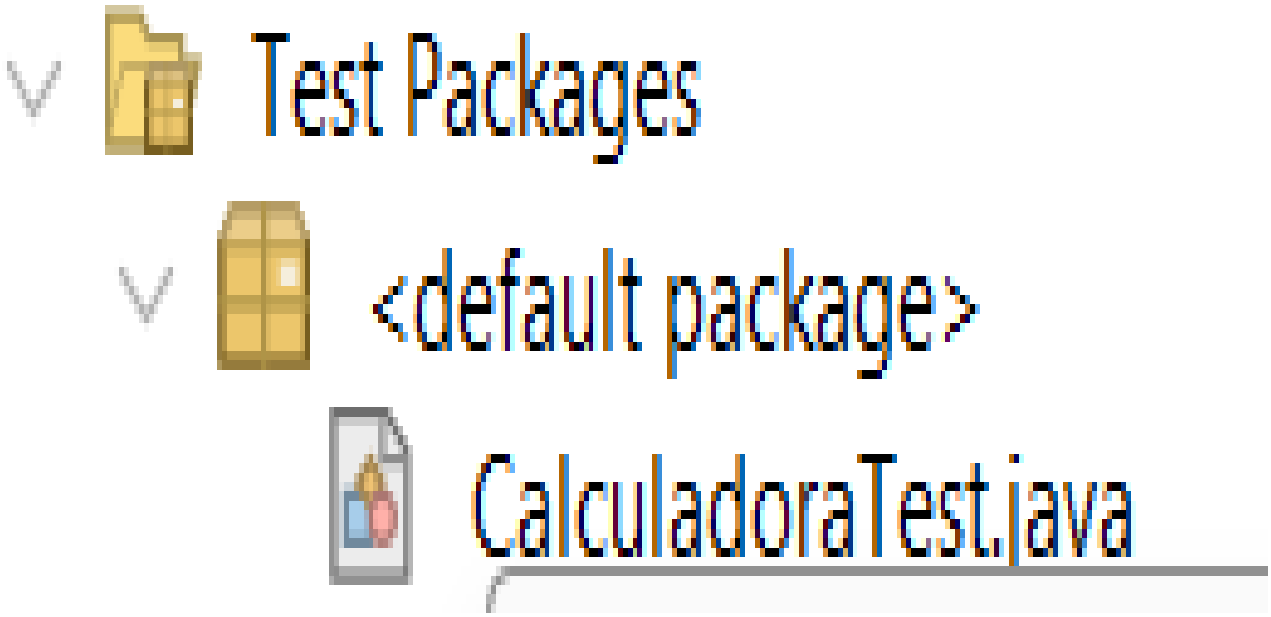
_Implementamos el codigo en un nuevo proyecto de java, lo creamos en NetBeans(IDE).



_Una vez creado nuestro proyecto en NetBeans(IDE), en la carpeta de Source Packages creamos un nuevo file, pero de formato Class, le damos click derecho a Source Packages, y le damos a donde dice new java Class



_Luego para generar Nuestra carpeta de Test, Hacemos click derecho en **Calculadora.java** el arbol de nuestro proyecto, selecciona **Tools > Create/Update Tests**. Se abre una ventana y nos aseguramos que en frameworks este JUnit 5. Y se creara nuestra carpeta de test Y un archivo **CalculadoraTest.java** de nuestro archivo class **Calculadora.java.**



2.1. Clase base: Calculadora.java

_En clase base ponemos este codigo:

```
public class Calculadora {

    public int sumar(int a, int b) {
        return a + b;
    }

    public int restar(int a, int b) {
        return a - b;
    }

    public int multiplicar(int a, int b) {
        return a * b;
    }

    public int dividir(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("No se puede dividir entre 0");
        }
        return a / b;
    }
}
```

2.2. Clase de pruebas: **CalculadoraTest.java.**

_En la clase para Test ponemos este codigo:

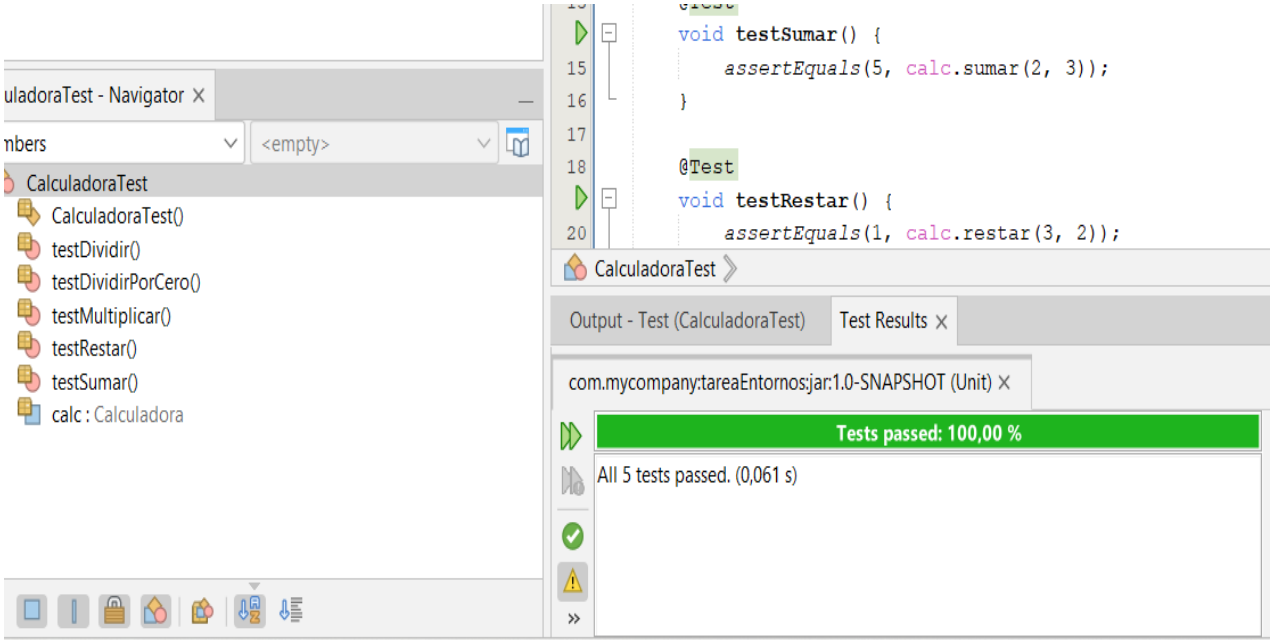
```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class CalculadoraTest {

    Calculadora calc = new Calculadora();

    @Test
    void testSumar() {
        assertEquals(5, calc.sumar(2, 3));
    }

    @Test
    void testRestar() {
        assertEquals(1, calc.restar(3, 2));
    }

    @Test
    void testMultiplicar() {
        assertEquals(6, calc.multiplicar(2, 3));
    }

    @Test
    void testDividir() {
        assertEquals(2, calc.dividir(4, 2));
    }

    @Test
    void testDividirPorCero() {
        assertThrows(IllegalArgumentException.class, () -> {
            calc.dividir(4, 0);
        });
    }
}
```

3. PRUEBAS UNITARIAS: RESULTADOS Y ANÁLISIS.

_Verificamos y cumplimos los objetivos de la practica.

3.1. Ejecución inicial (- Barra Verde).

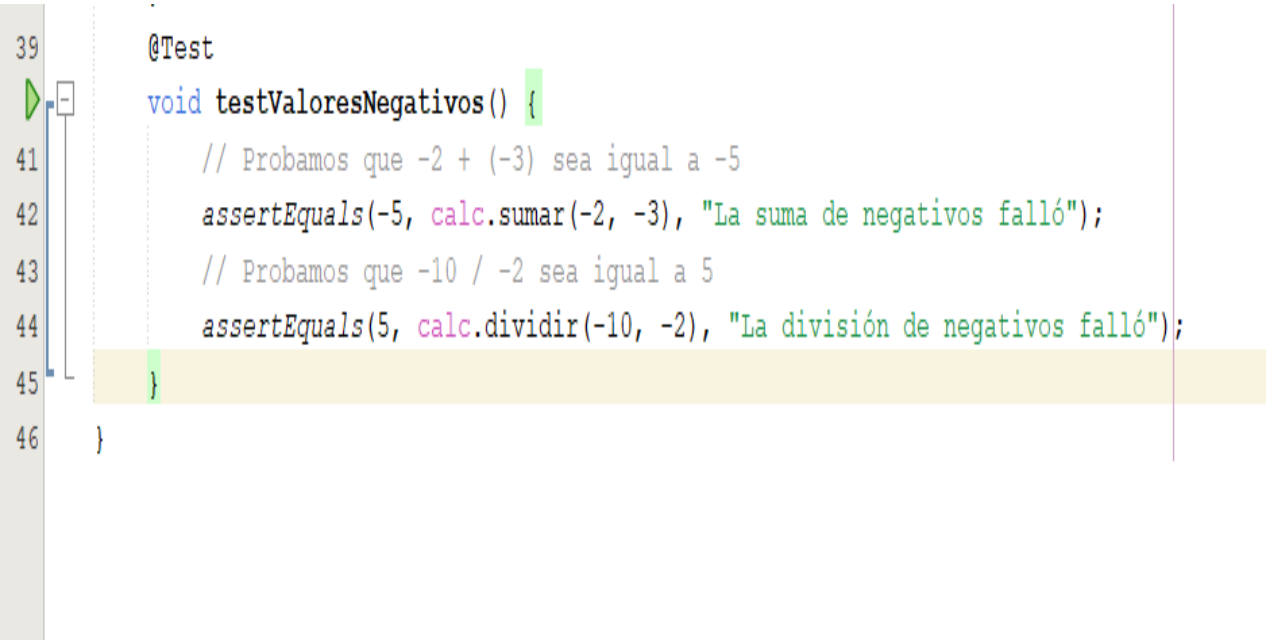
_Como primer objetivo tenemos que: Ejecutar nuestro primer Test, y ver que es lo que aparece cuando el teste es 100% Correcto.



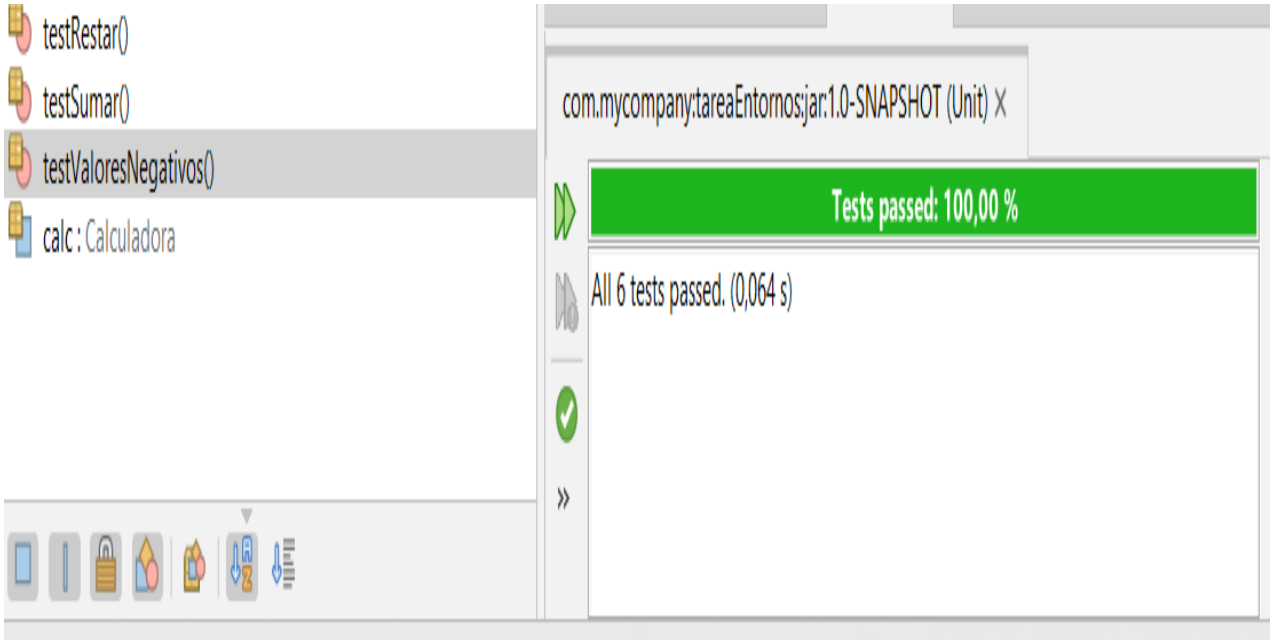
_Como podemos ver se nos abre una ventana Test Results, en la que la barra verde indica que el teste no ha detectado fallos y nuestro programa funciona correctamente.

3.2. Evaluación con valores negativos: ¿Qué ocurre?.

_Como segundo objetivo tenemos que añadir un metodo test, para probar valores negativos, en los que en este test se van a probar si los valores negativos tienen el resultado esperado.



_Creamos la clase con nombre **testValoresNegativos{}** esta es estatica por lo que no recibe ningun parametro.



_Como podemos apreciar, al darle denuevo a test, la barra correctamente aparece en verde con un 100% de exito.

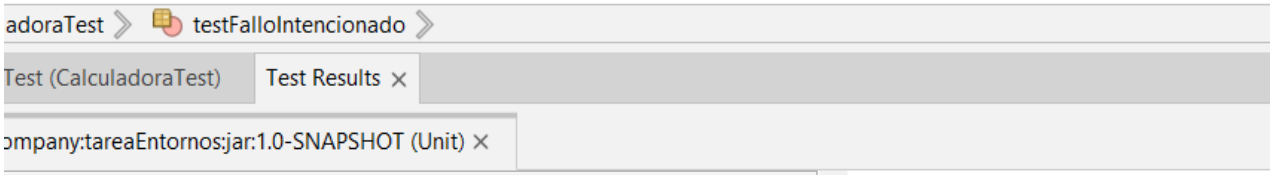
¿Que ocurre?.

_ Al añadir el test de valores negativos, el resultado sigue siendo exitoso (barra verde). Esto ocurre porque los operadores aritméticos de Java (+, -, *, /) están diseñados para manejar números enteros con signo según las reglas matemáticas. El test confirma que nuestra clase `Calculadora` es robusta y procesa correctamente valores bajo cero sin necesidad de código adicional.

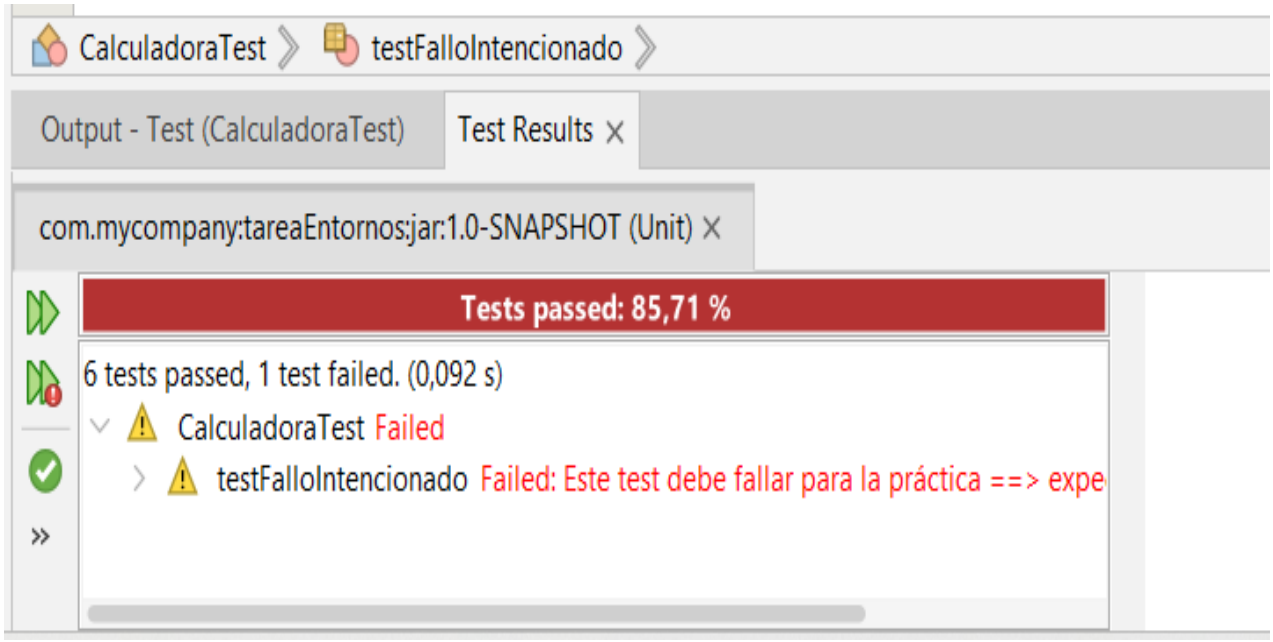
3.3. Simulación de errores: Fallos intencionados (Barra Roja).

_En este tercer objetivo, tenemos que realizar un fallo intencionado para reconocer que pasa cuando nuestro codigo falla en el test utilizando JUnit.

```java
@Test
void testFalloIntencionado() {
    // Forzamos el error: decimos que 5 + 5 es 11
    // (Claramente la calculadora devolverá 10)
    assertEquals(11, calc.sumar(5, 5), "Este test debe fallar para la práctica");
}
```



_Esta es la clase estatica que he creado para verificar que el programa falle correctamente.



_ Al ejecutar un test donde el resultado esperado no coincide con el devuelto por el método, NetBeans muestra una **barra roja** indicando un fallo de aserción (**`AssertionFailedError`**). En el panel de resultados, el IDE nos informa exactamente de la discrepancia: muestra el valor que nosotros esperábamos frente al valor real que calculó el programa . Esto permite al desarrollador localizar rápidamente errores de lógica en el código.

4. CONCLUSIONES PERSONALES

_ He comprobado que las pruebas unitarias son fundamentales para asegurar que, al realizar cambios futuros en el código (como añadir nuevas funciones), no 'rompamos' lo que ya funcionaba correctamente.

_ Gracias a la respuesta visual de NetBeans (barra verde/roja), es mucho más rápido localizar un error matemático mediante un `assertEquals` que ejecutando el programa manualmente y revisando la consola paso a paso.

_ La implementación de JUnit permite validar casos críticos, como la división por cero o el uso de números negativos, garantizando que el software sea robusto y responda adecuadamente ante entradas de datos inesperadas.