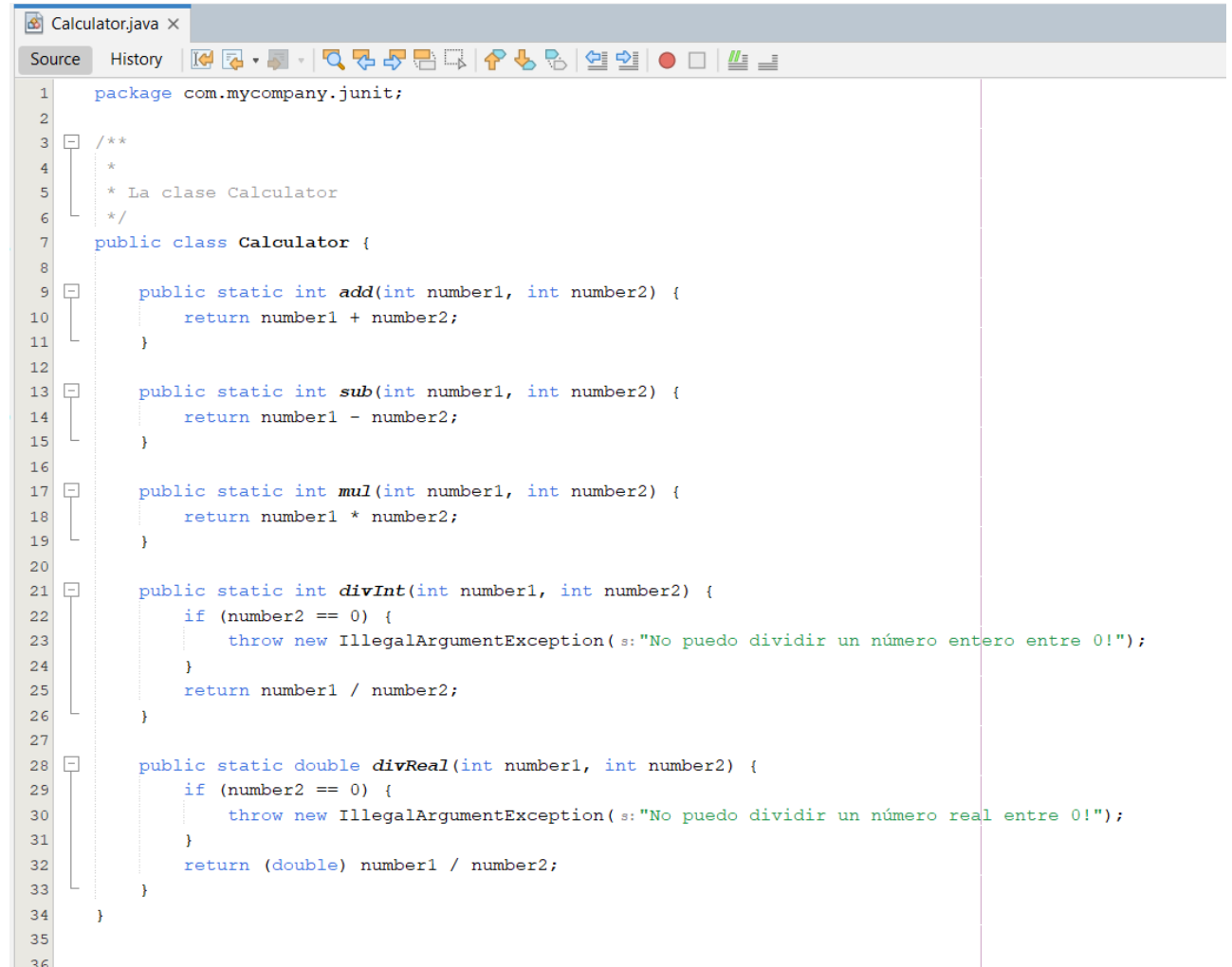


COMENZANDO CON UN EJEMPLO

Crea un nuevo proyecto Java en NetBeans de nombre "JUnit".

Supongamos que deseamos realizar pruebas unitarias a partir de la siguiente clase "Calculator", que utiliza métodos para realizar operaciones aritméticas con dos números enteros.

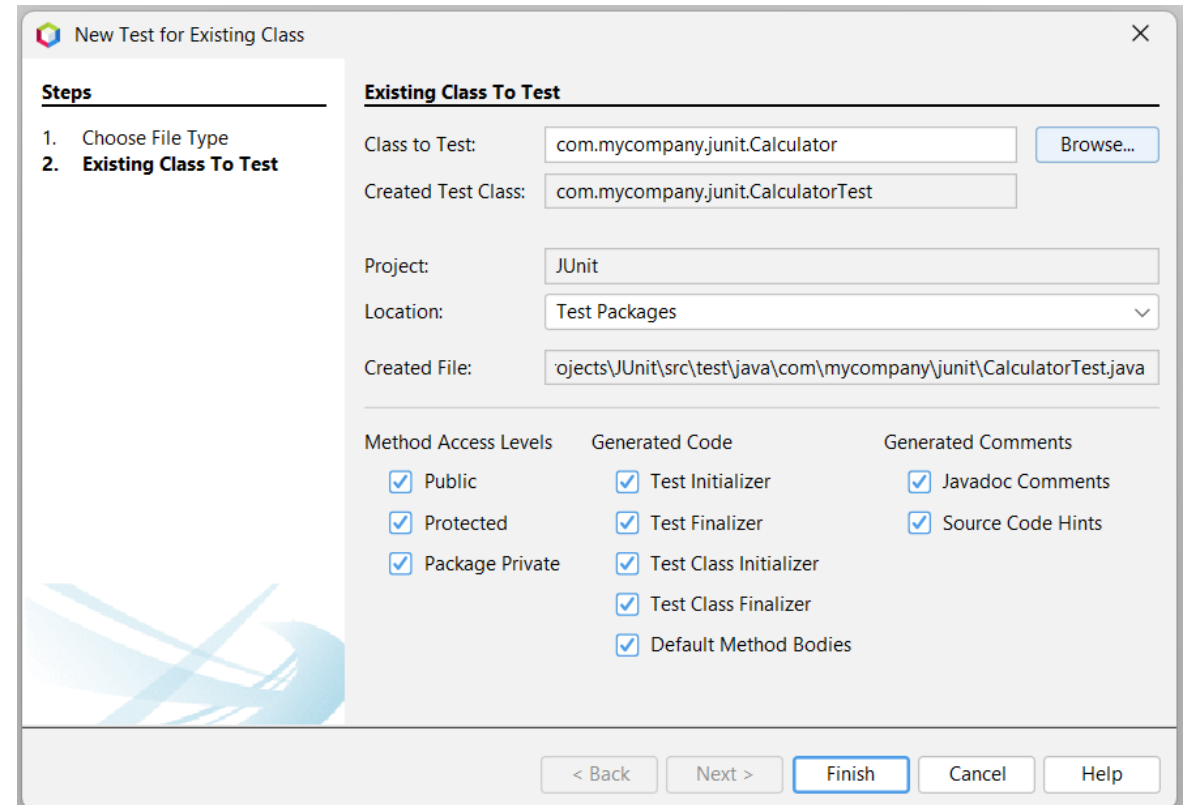
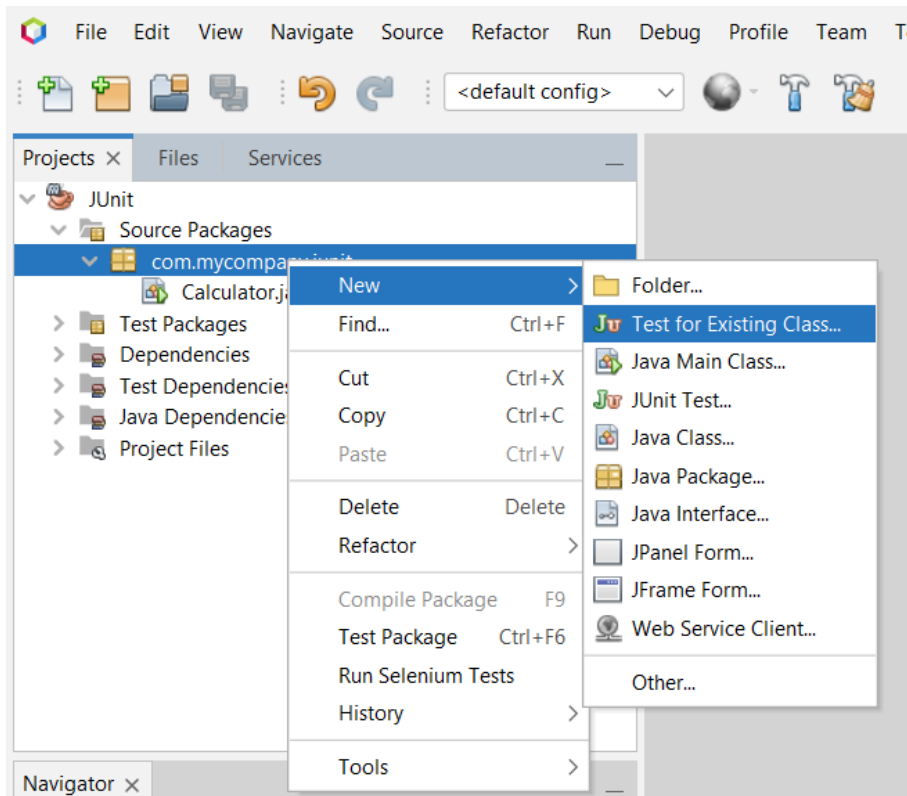
Ten en cuenta que los dos métodos para dividir arrojan una `IllegalArgumentException` cuando se intenta dividir entre cero.



```
1 package com.mycompany.junit;
2
3 /**
4  *
5  * La clase Calculator
6  */
7 public class Calculator {
8
9     public static int add(int number1, int number2) {
10         return number1 + number2;
11     }
12
13     public static int sub(int number1, int number2) {
14         return number1 - number2;
15     }
16
17     public static int mul(int number1, int number2) {
18         return number1 * number2;
19     }
20
21     public static int divInt(int number1, int number2) {
22         if (number2 == 0) {
23             throw new IllegalArgumentException(s:"No puedo dividir un número entero entre 0!");
24         }
25         return number1 / number2;
26     }
27
28     public static double divReal(int number1, int number2) {
29         if (number2 == 0) {
30             throw new IllegalArgumentException(s:"No puedo dividir un número real entre 0!");
31         }
32         return (double) number1 / number2;
33     }
34 }
35
36
```

COMENZANDO CON UN EJEMPLO

Copia el código anterior y de la siguiente forma crea los casos de prueba para la clase existente Calculator:



COMENZANDO CON UN EJEMPLO

Observa como se habrá creado bastante código de pruebas por defecto. Nosotros vamos a crear cuatro casos de prueba en la clase generada **CalculatorTest**, dos métodos para la suma y otros dos métodos para la resta.

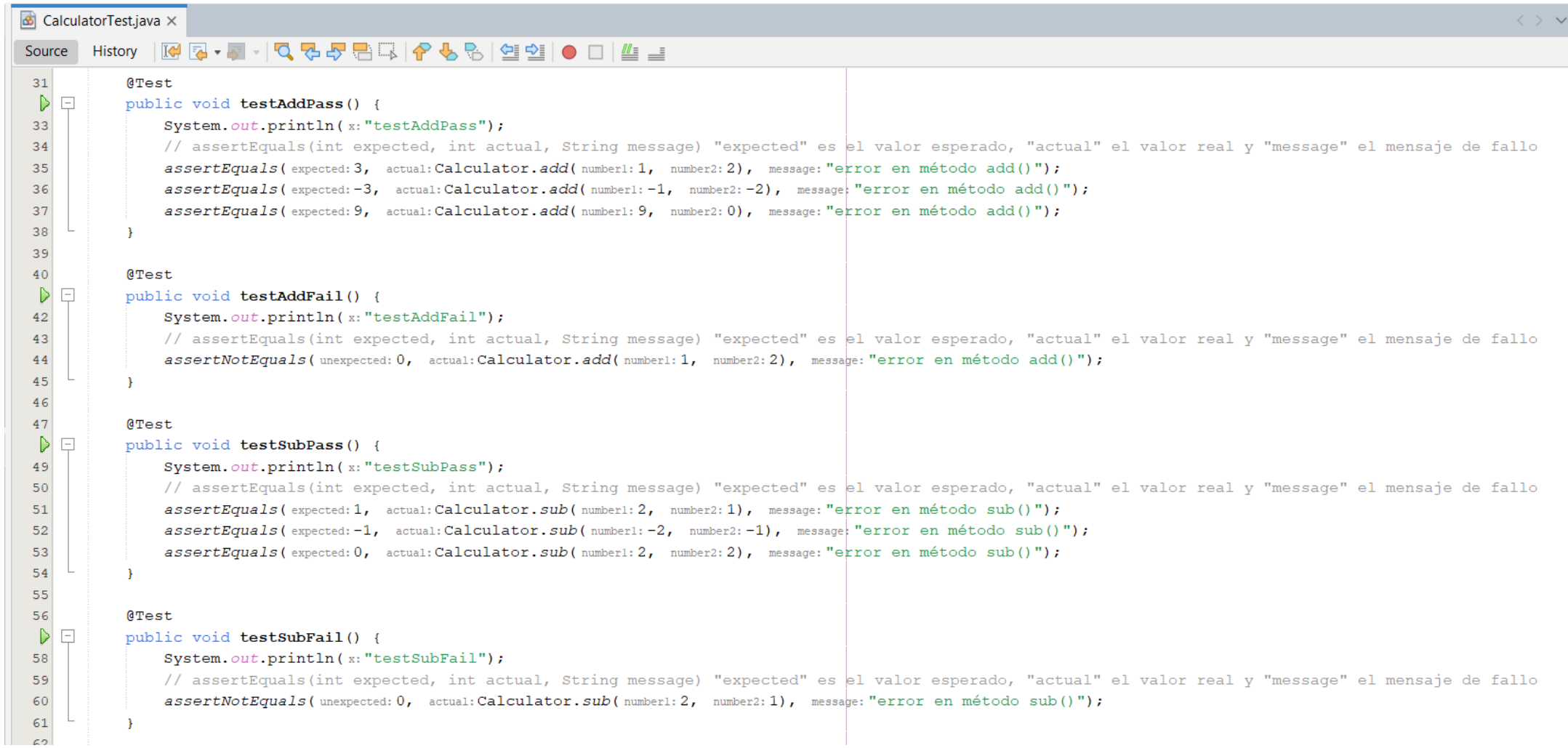
Existen dos conceptos muy importantes en JUnit que debes dominar: **anotaciones y assertions**.

Justo antes de cada método usaremos las **anotaciones**, que son etiquetas que añaden funcionalidad al código de manera muy concisa, haciéndolo muy fácil de leer. Comienzan siempre con una @. Por ejemplo, la anotación **@Test** representa un test que se debe ejecutar.

Por otro lado, los métodos de prueba deben llamarse **testXXX()**. Esto se debe a que JUnit usa el mecanismo de reflexión para encontrar y ejecutar estos métodos.

Dentro de cada método de prueba, usaremos las **assertions**, que son métodos estáticos para probar ciertas condiciones. Haremos uso de las distintas variaciones del método `assert()` para comparar los resultados esperados y reales. Por ejemplo: `assertEquals(int expected, int actual)` → Afirma que lo esperado y lo real son iguales. También tenemos `assertTrue()`, `assertFalse()`, etc. Estudiaremos todos estos métodos en una diapositiva posterior.

COMENZANDO CON UN EJEMPLO



The screenshot shows an IDE window titled "CalculatorTest.java". The code defines four test methods: `testAddPass()`, `testAddFail()`, `testSubPass()`, and `testSubFail()`. Each method uses `assertEquals` or `assertNotEquals` to verify the results of `Calculator.add` and `Calculator.sub` operations. Comments explain the parameters of the assertion methods: "expected" is the expected value, "actual" is the actual value, and "message" is the failure message.

```
31  @Test
32  public void testAddPass() {
33      System.out.println(x: "testAddPass");
34      // assertEquals(int expected, int actual, String message) "expected" es el valor esperado, "actual" el valor real y "message" el mensaje de fallo
35      assertEquals( expected: 3,   actual: Calculator.add( number1: 1, number2: 2), message: "error en método add()");
36      assertEquals( expected: -3,  actual: Calculator.add( number1: -1, number2: -2), message: "error en método add()");
37      assertEquals( expected: 9,   actual: Calculator.add( number1: 9, number2: 0), message: "error en método add()");
38  }
39
40  @Test
41  public void testAddFail() {
42      System.out.println(x: "testAddFail");
43      // assertEquals(int expected, int actual, String message) "expected" es el valor esperado, "actual" el valor real y "message" el mensaje de fallo
44      assertEquals( unexpected: 0,  actual: Calculator.add( number1: 1, number2: 2), message: "error en método add()");
45  }
46
47  @Test
48  public void testSubPass() {
49      System.out.println(x: "testSubPass");
50      // assertEquals(int expected, int actual, String message) "expected" es el valor esperado, "actual" el valor real y "message" el mensaje de fallo
51      assertEquals( expected: 1,   actual: Calculator.sub( number1: 2, number2: 1), message: "error en método sub()");
52      assertEquals( expected: -1,  actual: Calculator.sub( number1: -2, number2: -1), message: "error en método sub()");
53      assertEquals( expected: 0,   actual: Calculator.sub( number1: 2, number2: 2), message: "error en método sub()");
54  }
55
56  @Test
57  public void testSubFail() {
58      System.out.println(x: "testSubFail");
59      // assertEquals(int expected, int actual, String message) "expected" es el valor esperado, "actual" el valor real y "message" el mensaje de fallo
60      assertEquals( unexpected: 0,  actual: Calculator.sub( number1: 2, number2: 1), message: "error en método sub()");
61  }
62
```

COMENZANDO CON UN EJEMPLO

En el código de pruebas que se ha generado automáticamente verás métodos con la instrucción fail.

Es importante saber que el método **fail(String)** hace fallar el test.

Deberás borrar este método para que los test puedan funcionar, en caso contrario aparecerá el test como fallado con el mensaje: "The test case is a prototype."

```
62
63 ☐
64
65
66 ☒
67
68
69
70
71
72
73
74
75
76
77
```

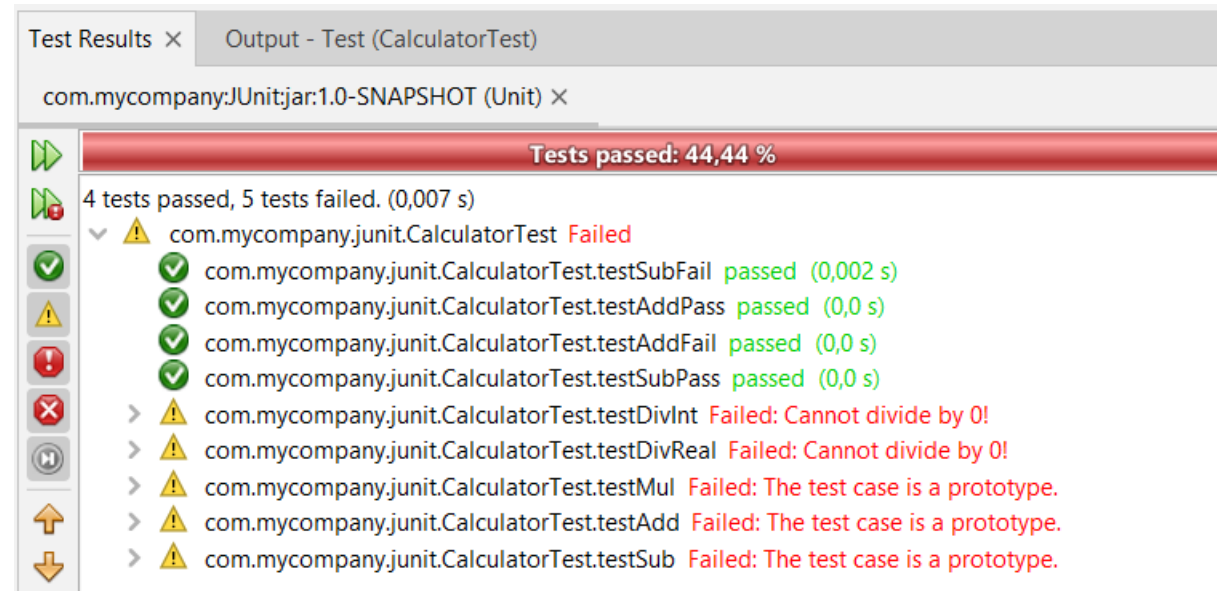
```
/**
 * Test of add method, of class Calculator.
 */
@Test
public void testAdd() {
    System.out.println( x: "add");
    int number1 = 0;
    int number2 = 0;
    int expectedResult = 0;
    int result = Calculator.add(number1, number2);
    assertEquals( expected: expectedResult, actual: result);
    // TODO review the generated test code and remove the default call to fail.
    fail( message: "The test case is a prototype.");
}
```

COMENZANDO CON UN EJEMPLO

Para ejecutar el caso de prueba, haz clic derecho en el archivo y luego selecciona **Test File**.

El resultado de la prueba se muestra en la **pestaña Test Results** indicándonos que prueba ha sido exitosa y cual no, diciéndonos en que línea estaría el error.

En la figura se ejecutaron 9 pruebas y 4 tuvieron éxito. Analiza los resultados de cada prueba.



COMENZANDO CON UN EJEMPLO

Modifica una de las pruebas correctas para forzar una fallo de la prueba y observa el resultado de la prueba, por ejemplo:

```
@Test
public void testAddPass() {
    System.out.println( x: "testAddPass");
    // assertEquals(int expected, int actual, String message) "expected" es el valor esperado, "actual" el valor real y "message" el mensaje de fallo
    assertEquals( expected: 0, actual: Calculator.add( number1: 1, number2: 2), message: "error en método add()");
    assertEquals( expected: -3, actual: Calculator.add( number1: -1, number2: -2), message: "error en método add()");
    assertEquals( expected: 9, actual: Calculator.add( number1: 9, number2: 0), message: "error en método add()");
}
```

Test Results × Output - Test (CalculatorTest)

com.mycompany.JUnit.jar:1.0-SNAPSHOT (Unit) ×

Tests passed: 33,33 %

3 tests passed, 6 tests failed. (0,016 s)

- com.mycompany.junit.CalculatorTest Failed
 - com.mycompany.junit.CalculatorTest.testAddPass Failed: error en método add() ==> expected: <0> but was: <3>
 - com.mycompany.junit.CalculatorTest.testSubFail passed (0,0 s)
 - com.mycompany.junit.CalculatorTest.testAdd Failed: The test case is a prototype.
 - com.mycompany.junit.CalculatorTest.testDivInt Failed: Cannot divide by 0!
 - com.mycompany.junit.CalculatorTest.testSub Failed: The test case is a prototype.
 - com.mycompany.junit.CalculatorTest.testSubPass passed (0,0 s)
 - com.mycompany.junit.CalculatorTest.testMul Failed: The test case is a prototype.
 - com.mycompany.junit.CalculatorTest.testAddFail passed (0,0 s)
 - com.mycompany.junit.CalculatorTest.testDivReal Failed: Cannot divide by 0!

COMENZANDO CON UN EJEMPLO

Resumen:

Un caso de prueba contiene varias pruebas, marcadas con la **anotación** de **"@Test"**. Cada una de las pruebas se ejecuta independientemente de las otras pruebas.

Dentro del método de prueba, usaremos las **assertions** a través de métodos estáticos **assertXXX()** (de la clase **org.junit.jupiter.api.Assertions**) para comparar los resultados de la prueba esperados y reales.

COMENZANDO CON UN EJEMPLO

Los métodos más utilizados de esta clase **org.junit.jupiter.api.Assertions** son:

- **assertEquals(resultado esperado, resultado actual):** Sirve para comparar dos tipos de datos u objetos y afirmar que son iguales.
- **assertNotEquals(resultado esperado, resultado actual):** Sirve para comparar dos tipos de datos u objetos y afirmar que no son iguales.
- **assertTrue(condición):** Si la condición pasada (puede ser una función que devuelva un booleano) es verdadera, el test será exitoso.
- **assertFalse(condición):** Si la condición pasada (puede ser una función que devuelva un booleano) es falsa, el test será exitoso.
- **assertNull(objeto):** Si un objeto es null el test será exitoso.
- **assertNotNull(objeto):** Si un objeto no es null el test será exitoso.
- **assertSame(objeto esperado, objeto actual):** Afirma que lo esperado y lo real se refieren al mismo objeto.
- **assertNotSame(Objeto inesperado, objeto actual):** Afirma que lo esperado y lo real no se refieren al mismo objeto.
- Puedes encontrar la lista completa de Assertions en el siguiente [enlace](#) de la API de JUnit 5.

COMENZANDO CON UN EJEMPLO

Tenemos varias **anotaciones** nuevas en JUnit 5 además de @Test:

- **@BeforeAll:** Solo puede haber un método con este marcador y se invoca una sola vez al principio de todos los test de la clase. Se utiliza junto a un método estático (static).
- **@BeforeEach:** Se ejecuta antes de cada uno de los test de la clase y después de @BeforeAll.
- **@AfterEach:** Se ejecuta después de cada uno de los test de la clase y antes de @AfterAll.
- **@AfterAll:** Solo puede haber un método con este marcador y se invoca una sola vez al finalizar todos los test de la clase. Se utiliza junto a un método estático (static).
- **@Disabled:** Los métodos marcados con esta anotación no serán ejecutados.

La anotación que mas usaremos será **@Test**, pero siempre nos puede venir bien usar alguna de las anotaciones anteriores.