JUnit 5 - Tests

```java
package org.iesbelen;

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

```java
package org.iesbelen;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.Test;

public class CalculadoraTest {
	Calculadora calculadora;

	@BeforeEach
	void setUp() {
		calculadora = new Calculadora();
	}

	@Test
	@DisplayName("Multiplicación sencilla debería ir")
	void testMultiplicar() {
		assertEquals(20, calculadora.multiplicar(4, 5), "Multiplicación normal debe funcionar");
	}

	@RepeatedTest(5)
	@DisplayName("Manejo del cero")
	void testMultiplicarPorCero() {
		assertEquals(0, calculadora.multiplicar(0, 5), "Multiplicar por cero debe ser cero");
		assertEquals(0, calculadora.multiplicar(5, 0), "Multiplicar por cero debe ser cero");
	}
}
```

Explicación de anotaciones y métodos JUnit 5

- El método anotado con @BeforeEach se ejecuta antes de cada test.

- Un método anotado con @Test define un método de test.

- @DisplayName se puede utilizar para definir el nombre de la test que se muestra al usuario.

- assertEquals es una declaración de afirmación que valida que el valor esperado y real es el mismo, si no, se muestra el mensaje al final del método.

- @RepeatedTest define que este método de prueba se ejecutará varias veces, en este ejemplo 5 veces.

Nota: en un test con múltiples *asserts*, cuando falla uno se considera que falla el test y se quedan sin comprobar los *asserts* siguientes. JUnit también instanciará un objeto de Test nuevo para invocar cada método de test de forma implícita.

Patrón para decidir si una clase es una clase de test:

**/Test*.java
Incluye todos sus subdirectorios y todos los nombres de archivo de Java que comienzan con Test.

**/* Test.java
Incluye todos sus subdirectorios y todos los nombres de archivo de Java que terminan con Test.

**/* Tests.java
Incluye todos sus subdirectorios y todos los nombres de archivos de Java que terminan en Tests.

**/*TestCase.java
Incluye todos sus subdirectorios y todos los nombres de archivos Java que terminan en TestCase.

Es una práctica común usar el sufijo Test o Tests al final de los nombres de las clases de prueba.

Assert (Aserciones)	Ejemplos
assertEquals	assertEquals(4,calculadora.multiplicar(2, 2),“mensaje opcional de fallo");
assertTrue	assertTrue('a' < 'b', () → "mensaje opcional de fallo");
assertFalse	assertFalse('a' > 'b', () → "mensaje opcional de fallo");
assertNotNull	assertNotNull(tuObjeto, "mensaje opcional de fallo");
assertNull	assertNull(tuObjeto, "mensaje opcional de fallo");

A continuación se destacan algunas funcionalidades de testeo especiales.

- Testear que se envía excepción por un bloque de código:

```
@Test
void testExpectedException() {
	//Testeas el código que lanza excepción mediante el paso por un lambda
	NumberFormatException thrown = assertThrows(NumberFormatException.class
	                                            , () -> { Integer.parseInt("One"); }
	                                            , "Se espera NumberFormatException");

	assertEquals("Para cadena: \"One\"", thrown.getMessage());
}
```

```
@Test
void exceptionTesting() {
	//Testeas el código que lanza excepción mediante el paso por un lambda
	Throwable exception = assertThrows(IllegalArgumentException.class, () ->calculadora.setMemoria("dos"));
	assertEquals("Memoria debe ser numérico",  exception.getMessage());
}
```

- Tests de tiempo no excedido:

```
@Test
void timeout NoExcedido() {
        assertTimeout(ofMinutes(1), () -> service.doBackup());
}
```

```
@Test
void timeoutNoExcedidoConResultado() {
        //Pasado ofSeconds(1) sin que la lambda termine, entonces falla el test
        String actualResult = assertTimeout(ofSeconds(1), () -> {
                return restService.request(request);
        });
        assertEquals(200, request.getStatus());
}
```

```
@Test
void timeoutNoExcedidoConResultado() {
    //Pasado ofSeconds(1) el test se interrumpe con fallo del test
    String actualResult = assertTimeoutPreemptively(ofSeconds(1), () -> {
        return restService.request(request);
    });
    assertEquals(200, request.getStatus());
}
```

- Flujos (streams) de Tests:

```
@TestFactory
  Stream<DynamicTest> testDiferentesOperacionesDeMultiplicar() {
    Calculadora tester = new Calculadora();
    int[][] data = new int[][] { { 1, 2, 2 }, { 5, 3, 15 }, { 121, 4, 484 } };
    return Arrays.stream(data).map(entry -> {
      int m1 = entry[0];
      int m2 = entry[1];
```

```
            int esperado = entry[2];
            return dynamicTest(m1 + " * " + m2 + " = " + esperado, () -> {
                assertEquals(esperado, tester.multiplicar(m1, m2));
            });
        });
    }
```

- Directorios temporales para manejo con ficheros

```
    @Test
    @DisplayName("Asegurar que 2 determinados ficheros existen")
    void dosDirectoriosConMismoContenido(@TempDir Path tempDir, @TempDir Path tempDir2) throws IOException {

        Path file1 = tempDir.resolve("myfile.txt");

        List<String> input = Arrays.asList("input1", "input2", "input3");
        Files.write(file1, input);

        assertTrue(Files.exists(file1), "El fichero existe");

        Path file2 = tempDir2.resolve("myfile.txt");

        Files.write(file2, input);
        assertTrue(Files.exists(file2), "El fichero existe");
    }
```

```
@Test
@DisplayName("Asegurar que un fichero existe y contiene las líneas con las que fue creado")
void ficheroNumerosExisteYContieneLasLineas(@TempDir Path tempDir)
  throws IOException {
    Path numeros = tempDir.resolve("numeros.txt");

    List<String> lineas = Arrays.asList("1", "2", "3");
```

```
    Files.write(numeross, lineas);

    assertAll(
      () -> assertTrue("El fichero existe", Files.exists(numeros)),
      () -> assertLinesMatch(lineas, Files.readAllLines(numeros)));
}
```