

3.D. JUnit

2. Probando JUnit.

2.3. Assertions (afirmaciones).

En el apartado anterior se indica que las llamadas a los métodos de prueba son configurables en diferentes aspectos, tales como: qué hacer antes/después de cada prueba, cómo pasar parámetros a las llamadas o cómo repetir de forma automática un determinado test. Pero en realidad, no hemos lanzado prueba alguna a los métodos del código del proyecto implementado en la clase ClaseUnoJUnit.

JUnit utiliza la **clase Assertions** para lanzar los **test**, que básicamente está compuesta por una serie de métodos, que una vez llamados ejecutan los métodos a probar y analizan su comportamiento comparándolos con los resultados que se espera de ellos.

Así, hay métodos que nos permiten comprobar si dos valores son o no iguales, si el valor del parámetro pasado se puede resolver como true o false, si el tiempo consumido en ejecutar un método supera el previsto

Además, los métodos están sobrecargados, permitiendo en algunos casos indicar el mensaje que ha de devolver si la comprobación no resulta exitosa, o definir un margen que valide dos números como iguales si su diferencia es inferior a dicha tolerancia

Para entender mejor los **métodos assert**, se va a modificar la clase ClaseUnoJUnitTest nuevamente, sustituyendo los mensajes por pantalla que se mostraban en la sección anterior por llamadas assert que permitan probar los métodos de la clase ClaseUnoJUnit.

Nuevo código de la ClaseUnoJUnitTest.java:

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.CsvSource;
import java.time.Duration;

class ClaseUnoJUnitTest {
    ClaseUnoJUnit miObjUno = new ClaseUnoJUnit();
    // Test que comprueba que el metodo devuelve true. Exitoso
    @Test
    @DisplayName("Comprueba True")
    void testDevuelveTrue() {
        assertTrue(miObjUno.DevuelveTrue());
    }
    // Test que comprueba que el metodo devuelve false. Error
    @Test
    @DisplayName("Comprueba False")
    void testDevuelveFalse() {
        assertFalse(miObjUno.DevuelveTrue());
    }
    // El metodo de prueba recibe varios parametros de entrada.
    // Se ejecuta tantas veces como tuplas de parametros se indican.
    // Para valor 1 exitoso, para valor 2 error.
    @ParameterizedTest
    @CsvSource({"HOLA,1","ADIOS,2"})
    @DisplayName("Comprueba val. Numerico")
    void testParameterizedIntTest(String a, int b) {
        assertEquals(1,miObjUno.DevuelveEnt(a,b));
    }
    // Metodo de prueba desactivado.
    @Test
    @Disabled
    void testDesactivado() {
        fail("No se muestra porque esta desactivado");
    }

    @Test
    @DisplayName("Tiempo de duracion")
    void testDevuelveTimeOut() {
        assertTimeoutPreemptively(Duration.ofMillis(200), () -> {miObjUno.DevuelveTrue();});
    }
}
```

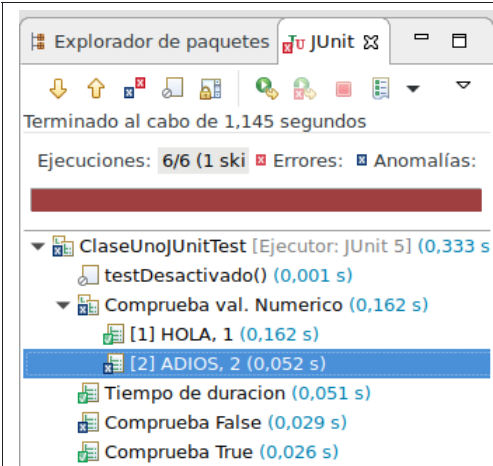
}

Como se puede ver, no se ha considerado necesaria actuación alguna antes de iniciar/finalizar los test, por lo que las etiquetas BeforeAll, AfterAll, BeforeEach y AfterEach no aparecen en el código.

A continuación se presenta una descripción de las pruebas realizadas.

assertTrue(BooleanSupplier booleanSupplier);	
Valida que la condición boolean devuelta por el booleanSupplier es true.	<pre>@Test @DisplayName("Comprueba True") void testDevuelveTrue() { assertTrue(miObjUno.DevuelveTrue()); }</pre>
assertFalse(BooleanSupplier booleanSupplier);	
Valida que la condición boolean devuelta por el booleanSupplier es false.	<pre>@Test @DisplayName("Comprueba False") void testDevuelveFalse() { assertFalse(miObjUno.DevuelveTrue()); }</pre>
assertEquals(int expected, int actual);	
<p>Compara el valor actual y el esperado. Si son iguales notifica éxito en la prueba, sino error.</p> <p>El método assertEquals ofrece alternativas, en cuanto a tipos a comparar, tolerancia aceptable en la comparación, personalización del mensaje a enviar ...</p>	<pre>@ParameterizedTest @CsvSource({"HOLA,1","ADIOS,2"}) @DisplayName("Comprueba val. Numerico") void testParameterizedIntTest(String a, int b) { assertEquals(1,miObjUno.DevuelveEnt(a,b)); }</pre>
fail(String message);	
Devuelve que el resultado de la prueba es errónea sin llevar a cabo comprobación alguna. En nuestro ejemplo esta prueba está desactivada.	<pre>@Test @Disabled void testDesactivado() { fail("No se muestra porque esta desactivado"); }</pre>
assertTimeoutPreemptively(Duration timeout, ThrowingSupplier<T> supplier);	
Valida que el tiempo consumido en la ejecución de la prueba no supere el tiempo previsible.	<pre>@Test @DisplayName("Tiempo de duracion") void testDevuelveTimeout() { assertTimeoutPreemptively(Duration.ofMillis(10), () -> {miObjUno.DevuelveTrue();}); }</pre>

Eclipse resume los resultados de las pruebas en la **vista JUnit**. A continuación podemos ver el resultado de las pruebas del código anterior.

	
<p>Analizando los resultados, podemos concluir:</p> <ul style="list-style-type: none"> Nos informa que hay un test que está desactivado. Al comprobar el método DevuelveEnt, que devuelve el valor recibido como parámetro, una vez el resultado es satisfactorio ya que el valor comprobado es igual al parámetro pasado, mientras que en la segunda ejecución somos advertidos de un error. El test del tiempo de ejecución del método DevuelveTrue fue exitoso. Por último al validar el método DevuelveTrue, el resultado es correcto cuando su respuesta se compara con true y erróneo cuando se hace con false. 	

Esta es sólo una muestra de todos los métodos assert disponibles. Pincha en este enlace [clase assertions](#) para ver una documentación mucho más extensa .