

ÍNDICE

1. INTRODUCCIÓN 2

2. PROBANDO JUNIT 2

2.1. CÓDIGO PROPUESTO..... 2

2.2. ETIQUETAS JUNIT. 4

2.3. ASSERTIONES..... 7

3. SESIÓN DE PRUEBAS..... 10

1. Introducción

En apartados anteriores se veía que entre los diferentes tipos de pruebas a los que someter los desarrollos software están las **pruebas unitarias**.

JUnit es una herramienta que ayuda a pasar de **forma automática** estas pruebas a los métodos y clases de los programas Java.

Para ello, habrá que identificar los casos de uso, datos de entrada y resultados esperados con las técnicas de caja blanca y negra ya conocidos. Una vez esté disponible esta información, toca lanzar con JUnit las ejecuciones programadas y valorar si los resultados obtenidos son exitosos (de acuerdo a lo previsto).

2. Probando JUnit

Este manual describe de forma introductoria conceptos relacionados con JUnit en su versión 5, tales como:

- Etiquetas disponibles para configurar pruebas en Java (anotaciones).
- Llamadas a métodos de prueba para comprobar el funcionamiento del código (assertions).
- Aprender a utilizar el plugin JUnit disponible en Eclipse.

En las siguientes urls podrás encontrar información útil referida a JUnit y su uso en el IDE Eclipse:

Propósito	URL
Guía de uso de JUnit 5	https://junit.org/junit5/docs/current/user-guide/
Uso de JUnit 5 en Eclipse	https://www.eclipse.org/eclipse/news/4.7.1a/#junit-5- support

2.1. Código propuesto

Se va a crear un proyecto con una única clase que llamaremos **ClaseUnoJUnit**, que será probada con la clase de prueba **ClaseUnoJUnitTest** basada en JUnit 5.

1. Crearemos la clase **ClaseUnoJUnit**

El código de la ClaseUnoJUnit.java es el siguiente:

```
public class ClaseUnoJUnit {
    // Metodo que siempre devuelve true
    public boolean DevuelveTrue()
    {
        return true;
    }
    // Metodo que devuelve el valor entero recibido como parametro
    public int DevuelveEnt(String sPrueba, int iValEnt)
    {
        return iValEnt;
    }
}
```

2. Crearemos la clase de prueba **ClaseUnoJUnitTest**:

Clase de prueba ClaseUnoJUnitTest.java

Pulsar el botón derecho sobre la clase ClaseUnoJUnit. En el menú contextual que aparece seleccionar: *Nuevo->Otras->Java->JUnit->Caso de prueba JUnit*.

Seleccionar la versión Jupiter de JUnit.

Trabajaremos con la versión más actual.

Caso de prueba JUnit

No se recomienda utilizar el paquete predeterminado.

☐ Prueba JUnit 3.8.1 nueva ☐ Prueba JUnit 4 nueva ☒ New JUnit Jupiter test

Carpeta fuente: EntornosJUnit/src Examinar...

Paquete: (predeterminado) Examinar...

Nombre: ClaseUnoJUnitTest

Superclase: java.lang.Object Examinar...

¿Qué apéndice de método desea crear?

☐ setUpBeforeClass() ☐ tearDownAfterClass()
☐ setUp() ☐ tearDown()
☐ constructor

¿Desea añadir comentarios tal como se configuran en las [propiedades](#) del proyecto actual?

☐ Generar comentarios

Clase sometida a prueba: ClaseUnoJUnit Examinar...

Indicar que considere los dos métodos de la clase ClaseUnoJUnit. Queremos validar ambos.

Caso de prueba JUnit nuevo

Métodos de prueba

Seleccione métodos para los que deben crearse apéndice de método de prueba.

Métodos disponibles:

- ☒ ClaseUnoJUnit
- ☒ DevuelveTrue()
- ☒ DevuelveEnt(String, int)

Al crear la primera clase de prueba en cada proyecto, **Eclipse** nos ofrece la opción de añadir JUnit al path del sistema.

Aceptamos el ofrecimiento.

Caso de prueba JUnit nuevo

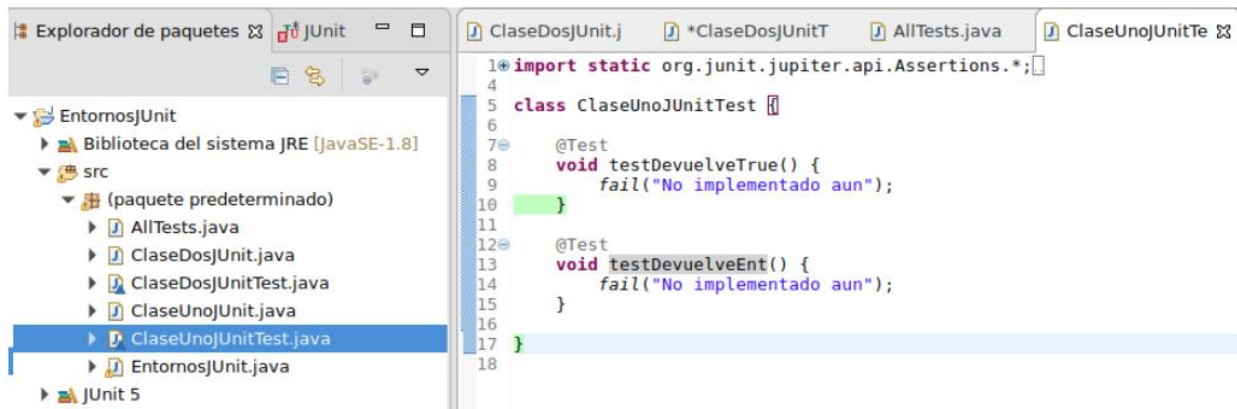
JUnit 5 is not on the build path. Do you want to add it?

☐ Not now
☐ Open the build path property page
☒ Perform the following action:

Add JUnit 5 library to the build path

Cancelar Aceptar

La siguiente figura muestra la clase de prueba que automáticamente ha creado Eclipse.



2.2. Etiquetas JUnit.

Las **etiquetas** o **anotaciones** permiten configurar las pruebas a realizar. En esta sección se describen las utilizadas con mayor frecuencia.

Tras una breve descripción de varias de las anotaciones en la siguiente tabla, se propone una nueva versión de la clase de prueba ClaseUnoJUnitTest, que incluye ejemplos para que puedas ponerlas en práctica.

Etiqueta @Test	
Indica que el siguiente método será llamado para ser probado.	<pre> @Test void testDevuelveTrue() { System.out.println("Llamando a testDevuelveTrue"); } </pre>
Etiqueta Repeatedtest	
Indica que el siguiente método será llamado las veces que la etiqueta recibe como parámetro. Dos en este caso.	<pre> @RepeatedTest(2) void testRepiteTest() { System.out.println("Llamando a testRepiteTest"); } </pre>
Etiqueta @BeforeAll	
Indica que el siguiente método es llamado una sola vez en toda la ejecución del programa. A continuación son llamadas las pruebas etiquetadas con @test, @Repeatedtest, @ParameterizedTest o @TestFactory.	<pre> @BeforeAll static void MetodoBeforeAll() { System.out.println("Llamando a MetodoBeforeAll"); } </pre>
Etiqueta @AfterAll	
Indica que el siguiente método es llamado una sola vez en toda la ejecución del programa. Antes habrán sido llamadas todas las pruebas etiquetadas con @test, @Repeatedtest, @ParameterizedTest o @TestFactory.	<pre> @AfterAll static void MetodoAfterAll() { System.out.println("Llamando a MetodoAfterAll"); } </pre>
Etiqueta @BeforeEach	
Indica que el siguiente método es llamado antes de ejecutar cada una de las pruebas etiquetadas con @test, @Repeatedtest, @ParameterizedTest o @TestFactory.	<pre> @BeforeEach void MetodoBeforeEach() { System.out.println("Llamando a MetodoBeforeEach"); } </pre>
Etiqueta @AfterEach	
Indica que el siguiente método es llamado después de ejecutar cada una de las pruebas etiquetadas con @test, @Repeatedtest, @ParameterizedTest o @TestFactory.	<pre> @AfterEach void MetodoAfterEach() { System.out.println("Llamando a MetodoAfterEach"); } </pre>
Etiquetas @ParameterizedTest y @ValueSource	
Permite pasar al método de prueba parámetros a utilizar en la ejecución. Se lanza una vez por parámetro pasado.	<pre> @ParameterizedTest @ValueSource(strings = {"HOLA", "ADIOS"}) void testParameterizedTest(String sMiInput) { System.out.println("Llamando a testParameterizedTest " + sMiInput); } </pre>
Etiquetas @ParameterizedTest y @CsvSource	
@ValueSource sólo permite el paso de un parámetro al método de prueba. Si necesitamos pasar varios se utiliza la etiqueta CsvSource. Se lanza una vez por cada tupla de parámetros pasada.	<pre> @ParameterizedTest @CsvSource({"HOLA,1", "ADIOS,2"}) void testParameterizedIntTest(String a, int b) { System.out.println("Llamando a testParameterizedIntTest : " + a + " --- " + b); } </pre>
@Disabled	
Desactiva la llamada a un método de prueba.	<pre> @Test @Disabled void testDesactivado() { System.out.println("Desactivada la llamada a testDesactivado"); } </pre>
@DisplayName	Muestra un nombre personalizado al método bajo prueba en el reporte de pruebas. Más adelante se muestran ejemplos.

El código de la ClaseUnoJUnitTest.java para esta prueba queda como sigue:

```
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.RepeatedTest;
import org.junit.jupiter.api.RepetitionInfo;
import org.junit.jupiter.params.ParameterizedTest; import
org.junit.jupiter.params.provider.CsvSource; import
org.junit.jupiter.params.provider.ValueSource;

class ClaseUnoJUnitTest {
    // Llamada al metodo una sola vez antes de todos los test
    @BeforeAll
    static void MetodoBeforeAll() {
        System.out.println("Llamando a MetodoBeforeAll");
    }

    // Llamada al metodo una vez despues de todos los test
    @AfterAll
    static void MetodoAfterAll() {
        System.out.println("Llamando a MetodoAfterAll");
    }

    // Llamada al metodo antes de cada metodo de prueba
    @BeforeEach
    void MetodoBeforeEach() {
        System.out.println("Llamando a MetodoBeforeEach");
    }

    // Llamada al metodo despues de cada metodo de prueba
    @AfterEach
    void MetodoAfterEach() {
        System.out.println("Llamando a MetodoAfterEach");
    }

    // Metodo etiquetado para ser probado
    @Test
    void testDevuelveTrue() {
        System.out.println("Llamando a testDevuelveTrue");
    }

    // Metodo etiquetado para ser probado 2 veces
    @RepeatedTest(2)
    void testRepiteTest() {
        System.out.println("Llamando a testRepiteTest");
    }
}
```

```

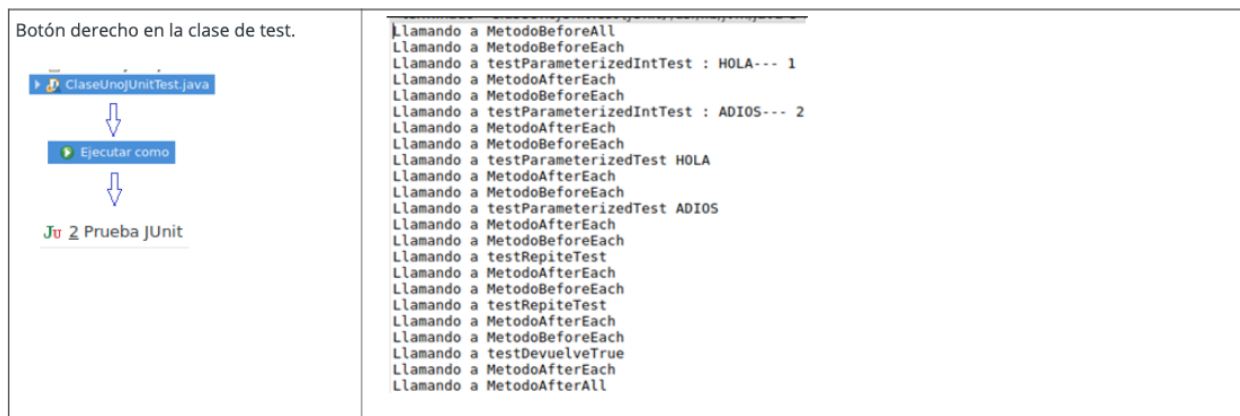
//El metodo de prueba recibe un parametro de entrada.
// Se ejecuta tantas veces como parametros se indican.
@ParameterizedTest
@ValueSource(strings = {"HOLA","ADIOS"})
void testParameterizedTest(String sMiInput) {
    System.out.println("Llamando a testParameterizedTest " +
        sMiInput );
}

// El metodo de prueba recibe varios parametros de entrada.
// Se ejecuta tantas veces como tuplas de parametros se indican.
@ParameterizedTest
@CsvSource({"HOLA,1","ADIOS,2"})
void testParameterizedIntTest(String a, int b) {
    System.out.println("Llamando a testParameterizedIntTest : " +a
        +"---"+b);
}

// Metodo de prueba desactivado.
@Test
@Disabled
void testDesactivado() {
    System.out.println("Desactivada la llamada a testDesactivado");
}
}

```

Las siguientes guras muestran como lanzar las pruebas definidas en el chero ClaseUnoJUnitTest.java y el resultado obtenido tras su ejecución.



2.3. Assertiones.

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

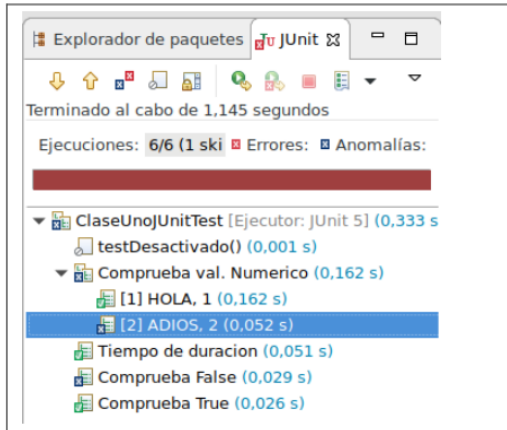
```

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.

<code>assertTrue(BooleanSupplier booleanSupplier);</code>	
Valida que la condición boolean devuelta por el booleanSupplier es true.	<pre> @Test @DisplayName("Comprueba True") void testDevuelveTrue() { assertTrue(miObjUno.DevuelveTrue()); } </pre>
<code>assertFalse(BooleanSupplier booleanSupplier);</code>	
Valida que la condición boolean devuelta por el booleanSupplier es false.	<pre> @Test @DisplayName("Comprueba False") void testDevuelveFalse() { assertFalse(miObjUno.DevuelveTrue()); } </pre>
<code>assertEquals(int expected, int actual);</code>	
<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>
<code>fail(String message);</code>	
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>
<code>assertTimeOutPreemptively(Duration timeout, ThrowingSupplier<T> supplier);</code>	
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>

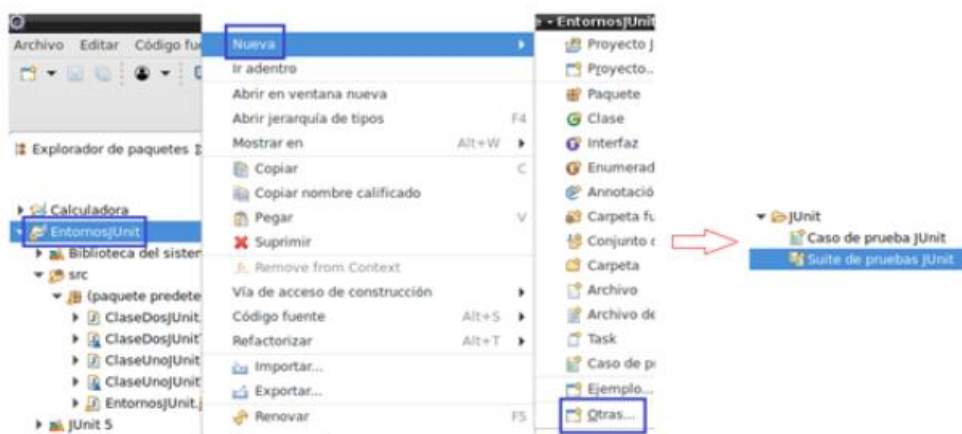
Eclipse resume los resultados de las pruebas en la **vista JUnit**. A continuación, podemos ver el resultado de las pruebas del código anterior.

	<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. En la página de JUnit podrás para ver una documentación mucho más extensa.

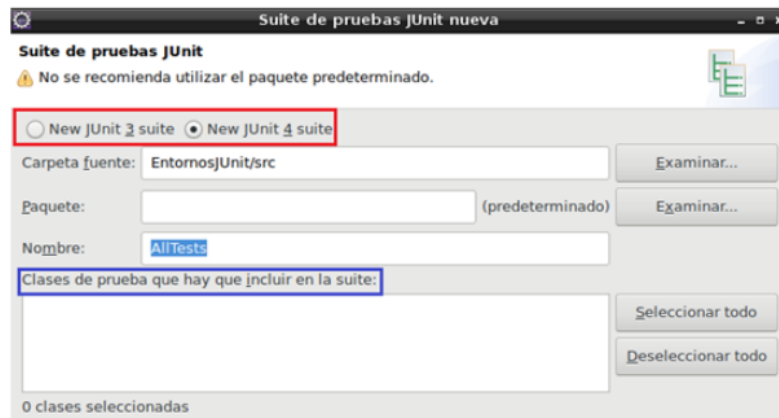
3. Sesión de pruebas

Las sesiones de pruebas se utilizan para configurar test sobre varias clases en una sola ejecución. La secuencia de ventanas que permiten crear la sesión de pruebas son las siguientes:



Observa en la siguiente ventana que Eclipse en su versión Photon Milestone 6 (4.8.0M6) todavía no incluye la creación de Suites para JUnit 5.0 (Júpiter), y como consecuencia no ofrece clases de prueba a incluir en la sesión.

Las etiquetas JUnit utilizadas en los casos de prueba de este proyecto son versión JUnit 5, y no son reconocidas por las versiones 4 y 3 de JUnit disponibles para generar una Suite de pruebas.



Nota: en futuras versiones de Eclipse habrá que comprobar si ya se encuentra disponible.