






# Introducción a JUnit

- Javier García Guzmán
- Lisardo Prieto González
- Carmen Carrero
- José Luis López Cuadrado
- Mat Max Montalvo Martínez

1



## Introducción (I)

- JUnit es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.
- JUnit es un conjunto de clases (Framework) que permite ejecutar las clases Java de manera controlada, permitiendo evaluar si el funcionamiento de cada uno de los métodos de la clase es el que se espera.

En función de algún **valor de entrada** se evalúa el **valor de retorno esperado**.

En función de este valor JUnit devolverá **si la clase pasó la prueba o no**.

Grado en Ingeniería Informática – Principios de Desarrollo Software

2

2

## Introducción (II)

- Compara valores esperados y valores generados:
  - Si son diferentes, la ejecución de la prueba no tiene éxito.
  - Si son iguales, la ejecución de la prueba tiene éxito.
- Permite controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores
- Herramientas con plug-in de JUnit:
  - NetBeans y Eclipse
  - Facilita al programador enfocarse en la prueba y el resultado esperado
  - Permiten la generación de plantillas para la creación de las pruebas de una clase Java
  - Dejan a la herramienta la creación de las clases que permiten coordinar las pruebas
  - Se realiza de manera automática

3

## ¿Qué se necesita para trabajar con JUnit 5.6.0?

- Java Development Kit 8 o superior.
- Eclipse IDE.

4

## Introducción (III)

### ■ ¿Qué es un **caso de prueba**?

- Es una clase o módulo que disponen de métodos para probar los métodos de otra clase o módulo concreto.
- Para cada clase que se quiera probar definiremos su correspondiente clase de caso de prueba
  - Para cada método, uno o más métodos que prueben cada uno de los definidos en la clase a probar.
- Mediante las **suites** podemos organizar los casos de prueba, de forma que cada suite agrupa los casos de prueba de módulos que están funcionalmente relacionados.

## Introducción (IV)

- Las pruebas que se van construyendo se estructuran en forma de árbol, de modo que las hojas son los casos de prueba, y podemos ejecutar cualquier subárbol (suite).
  - Construimos código que sirven para probar nuestros componentes (o clases), y que podremos ejecutar de forma automática
- Según el componente (o aplicación) vaya avanzando se dispondrá de un conjunto importante de casos de prueba, que servirá para hacer pruebas de regresión
  - *Importante:* cuando cambiamos un módulo que ya ha sido probado, el cambio puede haber afectado a otros módulos, y sería necesario volver a ejecutar las pruebas para verificar que todo sigue funcionando

## El Framework JUnit (I)

uc3m

 Universidad  
Carlos III  
de Madrid


- Open Source, disponible en <http://www.junit.org>
- Adecuado para el **Desarrollo dirigido por las pruebas** (Test-driven development)
- Consta de un conjunto de clases que se pueden utilizar para construir los casos de prueba y ejecutarlos automáticamente
- Los casos de prueba son programas Java que quedan archivados y pueden ser re-ejecutados tantas veces como sea necesario

Grado en Ingeniería Informática – Principios de Desarrollo Software

7

7

## El Framework JUnit (II)

uc3m

 Universidad  
Carlos III  
de Madrid


- JUnit permite construir “árboles de casos de prueba” (“suites”) que permiten organizar y gestionar mejor los casos de prueba sobre una o varias clases
- Componentes principales de las clases para programar los casos en JUnit
  - Paquete principal (Tiene que ser importado en los casos de prueba):
    - **org.junit.jupiter.api.Test**: Paquete que contiene las **clases principales** para la **realización de los casos de prueba**
    - **org.junit.jupiter.api.Assertions.\***: Paquete que contiene los **métodos principales** para la **realización de las asecciones**

Grado en Ingeniería Informática – Principios de Desarrollo Software

8

8

## El Framework JUnit (II)

uc3m

 Universidad  
Carlos III  
de Madrid


- Componentes principales de las clases para programar los casos en JUnit
  - Paquete principal (Tiene que ser importado en los casos de prueba):
    - **org.junit.jupiter.api.AfterEach**: Paquete que contiene las **clases principales** para indicar la ejecución de métodos después de **realización de cada caso de prueba**.
    - **org.junit.jupiter.api.AfterAll**: Paquete que contiene las **clases principales** para indicar la ejecución de métodos después de **realización de TODOS los casos de prueba**.
    - **org.junit.jupiter.api.BeforeEach**: Paquete que contiene las **clases principales** para indicar la ejecución de métodos antes de **realización de cada caso de prueba**.
    - **org.junit.jupiter.api.BeforeAll**: Paquete que contiene las **clases principales** para indicar la ejecución de métodos antes de **realización de TODOS los casos de prueba**.

Grado en Ingeniería Informática – Principios de Desarrollo Software

9

9

## JUnit 5: Componentes necesarios

uc3m

 Universidad  
Carlos III  
de Madrid


- Clases a utilizar:

Clases	Que se deben heredar para / Contiene
Test	la realización de los casos de prueba
SelectPackages SelectClasses	construir el(los) árbol(es) de casos de prueba (suites) ( <a href="https://howtodoinjava.com/junit5/junit5-test-suites-examples/">https://howtodoinjava.com/junit5/junit5-test-suites-examples/</a> )
Assert	métodos para determinar si la condición definida para el caso de prueba es cierta: si el valor obtenido al ejecutar el método y el valor esperado coinciden en el caso de prueba

Grado en Ingeniería Informática – Principios de Desarrollo Software

10

10

## JUnit 5: Componentes necesarios

## ■ Clase Test:

Métodos	Contiene
<code>public void setUp()</code>	el código que queremos que se <b>ejecute siempre antes de la ejecución</b> de cualquiera de los casos de prueba definidos. <b>Se usa con @BeforeAll</b>
<code>public void tearDown()</code>	el código que se ejecutará <b>siempre después de la ejecución</b> de los casos de prueba. Se emplea para liberar recursos o dejar el sistema en el estado en el que se encontraba antes de la realización de las pruebas Ej: si hemos insertado registros de prueba en una base de datos, para borrarlos. <b>Se usa con @AfterAll</b>

## JUnit 5: Componentes necesarios

## ■ Algunos términos empleados.

- El nombre de todos los métodos que implementen casos de prueba sobre uno de los métodos de una clase deben finalizar por XXXTest o XXXTests.
- Métodos `assertXXX` se emplean para determinar si el resultado obtenido del caso de prueba coincide con el esperado. Por ejemplo:
  - `assertEquals(Object, Object)` : Devuelve true si los dos objetos son iguales.
- `fail`: Permite probar si en el caso de haber un error se lanza una determinada excepción que se había introducido en el código

## JUnit 5

- Utilizaremos la perspectiva de la versión 5.6.0 de JUnit, que presenta ciertas diferencias con las versiones 3 y 4.
- Se siguen utilizando las clases de prueba que contendrán un conjunto de métodos de prueba.
- Ejemplo: Para probar los métodos de una clase *Calculadora* creamos la clase *CalculadoraTest*.

```
package tests;
import static org.junit.jupiter.api.Assertions.*;

class CalculadoraTest {
    @Test
    public void test3() {
        double primero = 3;
        double segundo = 5;
        Calculadora calculasuma = new Calculadora();
        double res = calculasuma.suma(primero, segundo);
        assertEquals (res, 8, 1, "El método suma ha fallado");
    }
}
```

13

## JUnit 5: Métodos de prueba

- Los métodos de prueba deben indicarse con la anotación `@Test`.

```
package tests;
import static org.junit.jupiter.api.Assertions.*;

class CalculadoraTest {
    @Test
    public void test3() {
        double primero = 3;
        double segundo = 5;
        Calculadora calculasuma = new Calculadora();
        double res = calculasuma.suma(primero, segundo);
        assertEquals (res, 8, 1, "El método suma ha fallado");
    }
}
```

- Tienen que ser públicos, sin parámetros y devolver void.

14

## JUnit 5: Métodos de prueba

- Los métodos de test los creamos siguiendo el patrón AAA:

1. Arrange (Preparar)
2. Act (Actuar)
3. Assert (Afirmary)

```
@Test
public void testSuma() {
    // Arrange
    Calculadora calculadora = new CalculadoraImpl();
    // Act
    double res = calculadora.suma(1, 1);
    // Assert
    assertEquals(2, res, 0);
}
```

## JUnit 5: Condiciones 'assert'

- Clase Assert: para realizar validaciones en los métodos de prueba, se utilizan las condiciones de aceptación assertXXX():

Métodos	Validar
assertEquals(A, B, "message")	la igualdad de los objetos A y B, utiliza el método equals()
assertSame(A, B, "message")	que A y B son el mismo objeto, utiliza el operador ==
assertTrue(A, "message")	que la condición A es true
assertNotNull(A, "message")	que el objeto A no es nulo



## JUnit 5: métodos accesorios

- Las clases de tests tienen que programarse eficientemente y refactorizarse cuando sea necesario.
- JUnit nos proporciona anotaciones para definir métodos adicionales pre/post test (fixture methods), mediante los que podemos inicializar o finalizar elementos comunes, evitar duplicidades, preparar datos, etc:
  - @BeforeAll
  - @BeforeEach
  - @AfterEach
  - @AfterAll

## JUnit 5: Manejo excepciones

- Si estamos probando un método que puede lanzar excepciones, puede resultar interesante poder verificar, que para las condiciones oportunas, el código lanza la excepción esperada:

- Para ello utilizamos:

```
@Test
void testTransferFundsException() {
    Assertions.assertThrows(BankException.class, () -> {
        source.deposit(125.00f);
        source.transferFunds(destination, 200.00f);
    });
}
```

- El test fallará, si no se produce la excepción *BankException*.

## JUnit 5: Ignorar un test

- JUnit proporciona una anotación para indicar que un determinado test no se ejecute.

- Para ello utilizamos el parámetro @Disabled

```
@Test
@Disabled("No implementado - aunque sea mentira")
public void test3() {
    double primero = 3;
    double segundo = 5;
    Calculadora calculasuma = new Calculadora();
    double res = calculasuma.suma(primero, segundo);
    assertEquals(res, 8, 1, "El método suma ha fallado");
}
```

- Evitar un test no es una buena práctica, pero si en alguna ocasión es necesario es mejor utilizar la anotación @Disabled a simplemente comentar código.

## JUnit 5: Test Runners (II)

Ejecución exitosa de los casos de prueba

Nombre de la clase que contiene los casos de prueba

Casos de prueba ejecutados

Traza del error en caso de que se haya detectado alguno

## JUnit 5: Test Runners (III)

**Errores:** Fallos ocurridos en la ejecución del programa.

**Fallos:** Casos de prueba en los que el resultado no coincide con el valor esperado.

Ejecución fallida de alguno de los casos de prueba

Casos de prueba ejecutados

Casos de prueba fallido

Traza del error aparecido en el caso de prueba

## JUnit 5: Test Runners (IV)

- En resumen el TestRunner nos permite:
  - Ejecutar los casos de prueba definidos.
  - Conocer los resultados de la ejecución de los casos de prueba.
  - En caso de resultado no satisfactorio:
    - Determinar qué errores se han encontrado en la ejecución de los casos de prueba.
    - Conocer los fallos que ha habido en la ejecución, indicando que casos de prueba no han proporcionado los resultados esperados.

## JUnit 5: Suites de pruebas

- A medida que vamos acumulando métodos de prueba, nos podría interesar organizarlos o agruparlos.
- Las suites permiten asignar métodos de prueba a grupos. Para ello, añadimos a nuestra clase de prueba un método suite() que devuelva una instancia de la clase "org.junit.platform.suite".
- Permite seleccionar y ejecutar clases del mismo paquete (@SelectClasses) o de distintos paquetes seleccionándolos (@SelectPackages).
- Solo incluye clases con nombres que comienzan con "Test" o terminan con "Test" o "Tests".

## RESUMEN: Anotaciones

- Se utilizan para interpretar las pruebas, indicando al motor de ejecución de JUnit qué debe hacer con un determinado método.

Anotaciones	
@Test	identifies a method as a test method.
@BeforeAll	is executed once, before the start of all tests. It is used to perform time intensive activities, for example to connect to a database. need to be defined as static.
@AfterAll	is executed once, after all tests have been finished. It is used to perform clean-up activities, for example to disconnect from a database. need to be defined as static
@BeforeEach	is executed before each test. It is used to prepare the test environment (e.g. read input data, initialize the class).
@AfterEach	is executed after each test. It is used to cleanup the test environment (e.g. delete temporary data, restore defaults).
@Disabled	ignores the test method. This is useful when the underlying code has been changed and the test case has not yet been adapted or if the execution time of this test is too long to be included.

## RESUMEN: Aserciones

- Son métodos del Framework de JUnit (métodos estáticos de `org.junit.Assert`) para comprobar y comparar valores.  
(<https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html>)

Assertion	What does it assert?
<b><code>assertTrue</code></b> (boolean condition) <b><code>assertTrue</code></b> (boolean condition, String message)	Asserts that the supplied condition is true.
<b><code>assertFalse</code></b> (boolean condition) <b><code>assertFalse</code></b> (boolean condition, String message)	Asserts that the supplied condition is not true.
<b><code>assertEquals</code></b> (short expected, short actual) <b><code>assertEquals</code></b> (short expected, short actual, String message)	Asserts that expected and actual are equal.
<b><code>assertNotEquals</code></b> (Object unexpected, Object actual)	Asserts that expected and actual are not equal.
<b><code>assertNull</code></b> (Object actual)	Asserts that actual is null.
<b><code>assertNotNull</code></b> (Object actual)	Asserts that actual is not null.
<b><code>fail</code></b> (String message)	Fails a test with the given failure message.

## RESUMEN: Pruebas con JUnit

- Paquetes a importar:

```
import static org.junit.Assertions.*;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Test;
```

- Los métodos deben ser públicos y no devolver nada.
- Los métodos anotados con `@BeforeAll` y `@AfterAll` además deben ser definidos como estáticos.

## RESUMEN: Probando excepciones esperadas

- Para probar que salta una excepción cuando debe:

```
@Test
public void testExceptions() {
    Assertions.assertThrows(Exception.class, () -> {
        m.miMetodo();
    });
}
```

- Para comprobar el estado después de la excepción:

```
try {
    m.miMetodo();
    fail("Debería haber saltado una excepción");
}
catch(Exception excepcion) {
}
```

## RESUMEN: Suites de pruebas

- Permiten ejecutar conjuntamente todos los métodos de varias clases de pruebas de forma ordenada.

```
import org.junit.runner.RunWith;
import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectClasses;

@SelectClasses({
    BookTest1.class,
    BookTest2.class
})

public class BookSuite {
}
```

## RESUMEN: Fallos vs. Errores en JUnit

- **Fallo (*failure*)** es un caso de prueba en el que el resultado obtenido no coincide con el resultado esperado.
  - La aparición de un fallo indica un problema que ya se preveía en el código que se está probando y para el cual se creó una aserción.
- **Error (*error*)** es un problema inesperado ocurrido durante la ejecución de las pruebas e indicado por una excepción.
  - La aparición de un error indica que hay que arreglar algo más que el código que se está probando, probablemente la implementación de la prueba.

## JUnit4 vs JUnit 5

FEATURE	JUNIT 4	JUNIT 5
Declare a test method	@Test	@Test
Execute before all test methods in the current class	@BeforeClass	@BeforeAll
Execute after all test methods in the current class	@AfterClass	@AfterAll
Execute before each test method	@Before	@BeforeEach
Execute after each test method	@After	@AfterEach
Disable a test method / class	@Ignore	@Disabled
Test factory for dynamic tests	NA	@TestFactory
Nested tests	NA	@Nested
Tagging and filtering	@Category	@Tag
Register custom extensions	NA	@ExtendWith

## INSTALACIÓN JUNIT

- Instalar Eclipse.
- Agregar Librería JUnit 5 ([www.junit.org](http://www.junit.org)).
- Si no se tiene **JDK** descargar e instalar.
- Asegurarse de tener **Java 8** o superior.

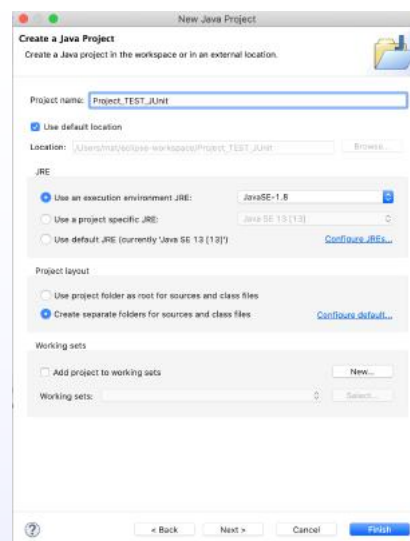
## JUnit con Eclipse

1.- Creamos un proyecto Java.

1.2.- Agregamos el nombre del proyecto.

1.3.- Verificamos que estemos usando la versión de JRE 8 o superior.

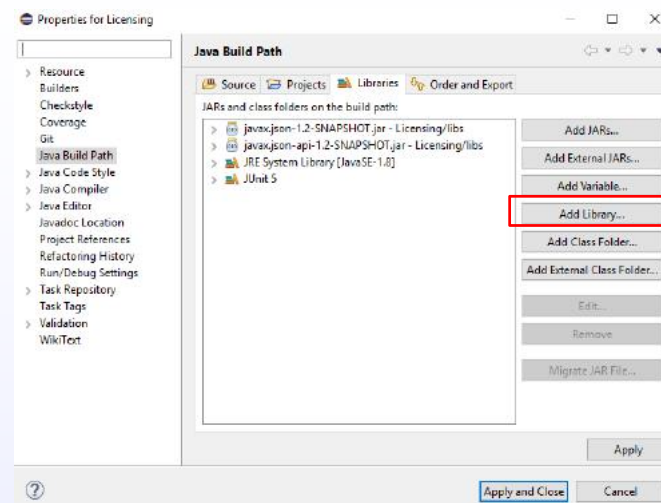
1.4.- Damos clic en Next.





# JUnit con Eclipse

## 2.- Añadir Librerías

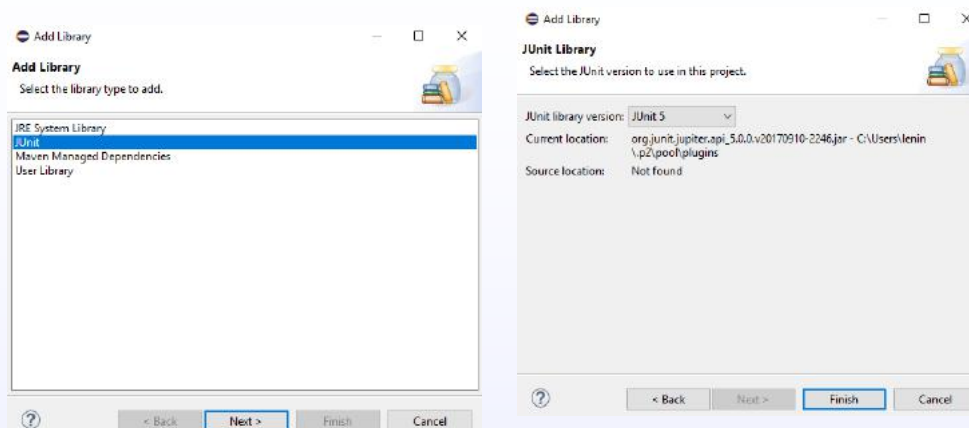


Grado en Ingeniería Informática – Principios de Desarrollo Software

33

33

# JUnit con Eclipse



Grado en Ingeniería Informática – Principios de Desarrollo Software

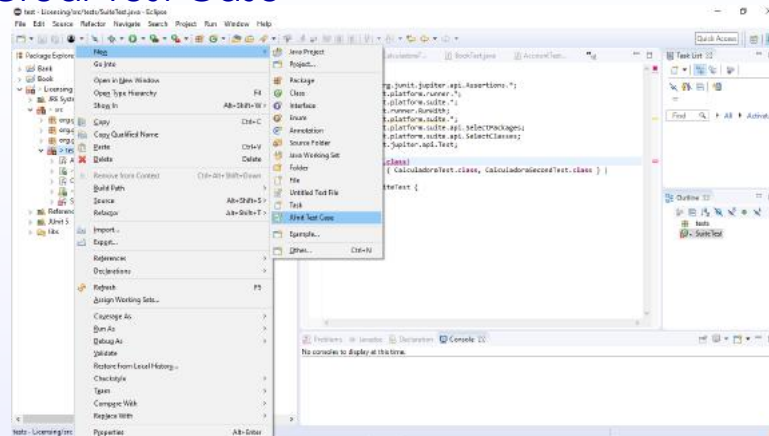
34

34

## JUnit con Eclipse

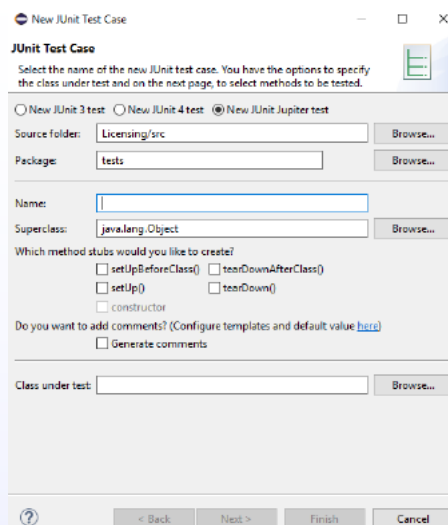
## 3.- Escribir Clase

## 4.- Crear Test Case



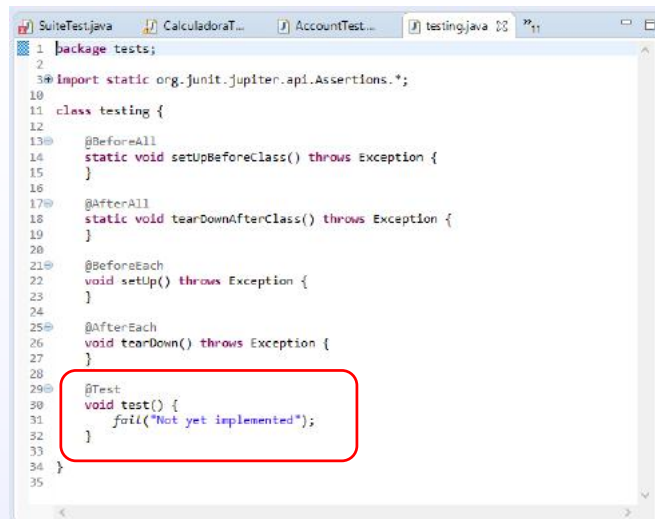
35

## JUnit con Eclipse



36

## JUnit con Eclipse



```

1 package tests;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
10 class testing {
11
12     @BeforeAll
13     static void setUpBeforeClass() throws Exception {
14     }
15
16     @AfterAll
17     static void tearDownAfterClass() throws Exception {
18     }
19
20     @BeforeEach
21     void setUp() throws Exception {
22     }
23
24     @AfterEach
25     void tearDown() throws Exception {
26     }
27
28     @Test
29     void test() {
30         fail("Not yet implemented");
31     }
32 }
33
34
35

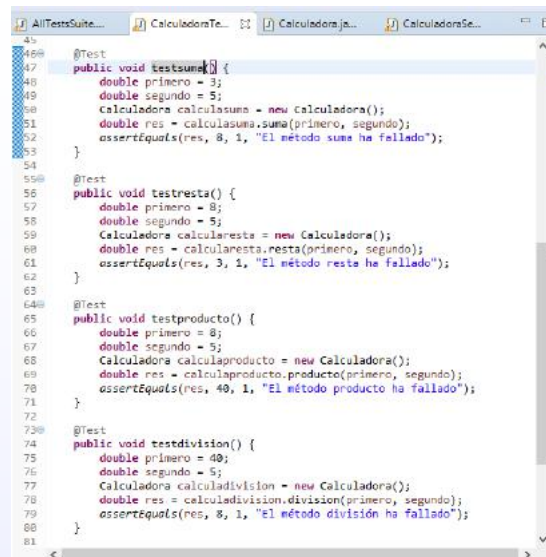
```

Grado en Ingeniería Informática – Principios de Desarrollo Software

37

37

## JUnit con Eclipse: Test



```


45
46 @Test
47 public void testsuma() {
48     double primero = 3;
49     double segundo = 5;
50     Calculadora calculadora = new Calculadora();
51     double res = calculadora.suma(primero, segundo);
52     assertEquals(res, 8, 1, "El método suma ha fallado");
53 }
54
55 @Test
56 public void testresta() {
57     double primero = 8;
58     double segundo = 5;
59     Calculadora calculadora = new Calculadora();
60     double res = calculadora.resta(primero, segundo);
61     assertEquals(res, 3, 1, "El método resta ha fallado");
62 }
63
64 @Test
65 public void testproducto() {
66     double primero = 8;
67     double segundo = 5;
68     Calculadora calculadora = new Calculadora();
69     double res = calculadora.producto(primero, segundo);
70     assertEquals(res, 40, 1, "El método producto ha fallado");
71 }
72
73 @Test
74 public void testdivision() {
75     double primero = 40;
76     double segundo = 5;
77     Calculadora calculadora = new Calculadora();
78     double res = calculadora.division(primero, segundo);
79     assertEquals(res, 8, 1, "El método división ha fallado");
80 }
81


```

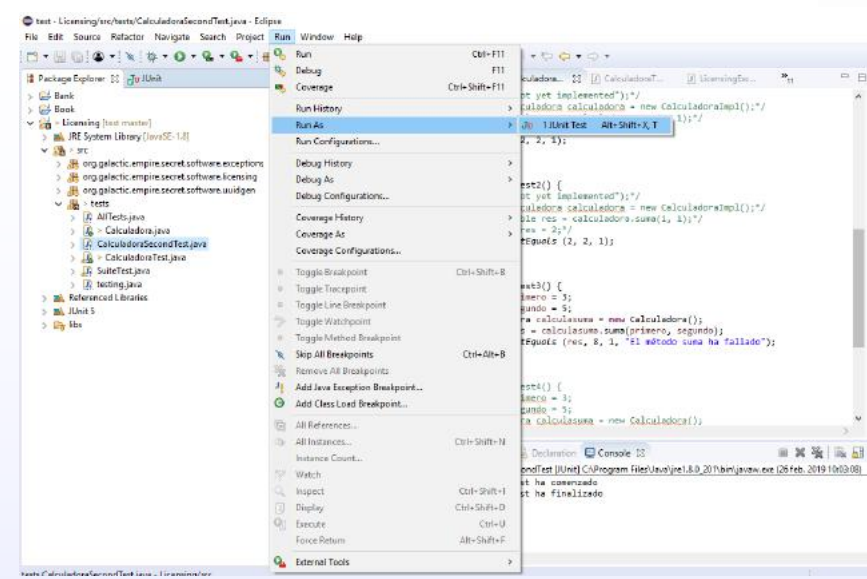
Grado en Ingeniería Informática – Principios de Desarrollo Software

38

38




**Universidad  
Carlos III  
de Madrid**





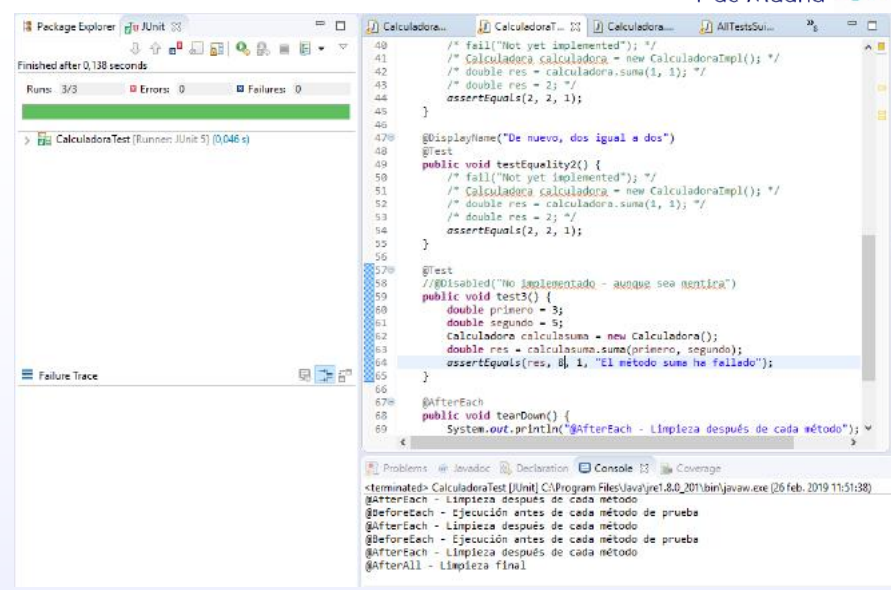
Grado en Ingeniería Informática – Principios de Desarrollo Software

39

39



**Universidad  
Carlos III  
de Madrid**




Grado en Ingeniería Informática – Principios de Desarrollo Software

40

40

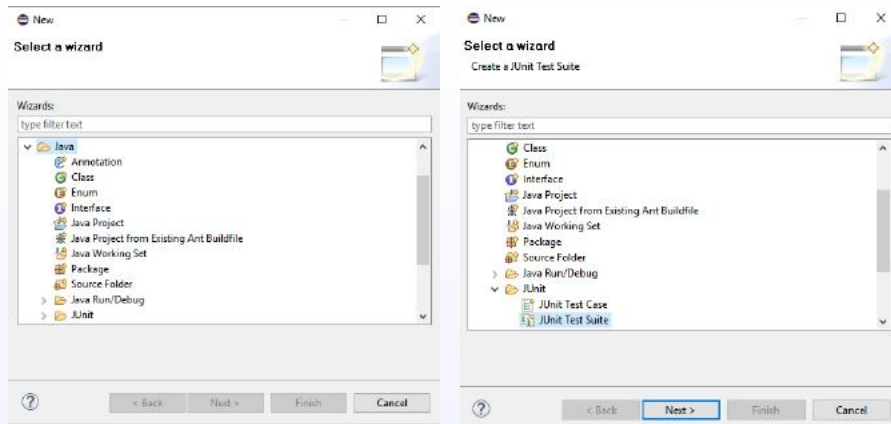
The screenshot shows the IntelliJ IDE with the Package Explorer on the left, the Run/Debug console at the top, and the Test Results window at the bottom. The Run/Debug console shows the test runner output, indicating that the tests passed. The Test Results window shows a table of test results, with the 'test30' test highlighted in green, indicating it passed. The test results are as follows:

Test Name	Duration	Status
test30 (0.032 s)	0.032 s	Passed
Dos igual a dos (0.000 s)	0.000 s	Passed
De nuevo, dos igual a dos (0.000 s)	0.000 s	Passed

41

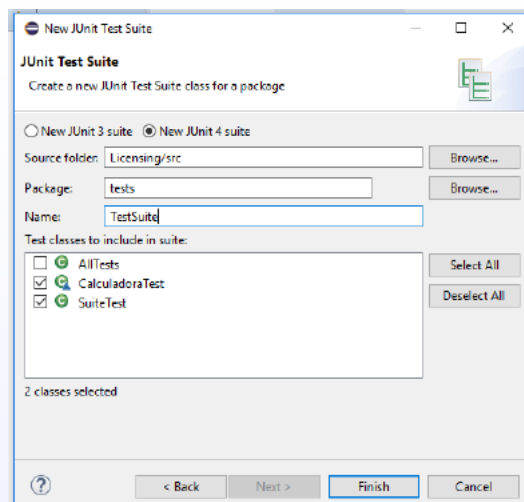
42

## JUnit con Eclipse: Suite



43

## JUnit con Eclipse: Suite



Aún usando JUnit 5, el asistente nos permite generar automáticamente suites asociadas a JUnit 4 que aún no son estables con JUnit 5. Suele dar problemas.

Por ello es mejor generar las suites manualmente usando las anotaciones @SelectClasses.

44

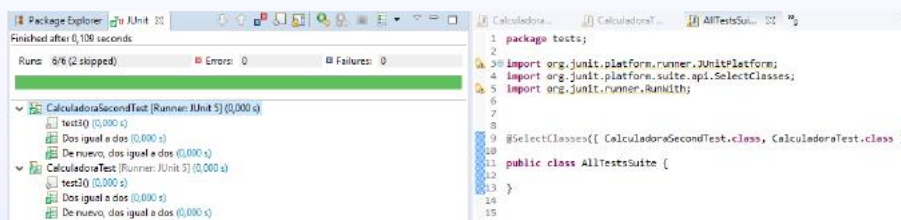
## JUnit con Eclipse: Suite

```
package tests;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.runner.RunWith;

@SelectClasses({ CalculadoraSecondTest.class, CalculadoraTest.class })
@RunWith(JUnitPlatform.class)
public class AllTestsSuite {

}
```



45

## Ejercicios

- Ejercicio Sumar y Multiplicar:
  - ☐ Datos y resultados válidos
  - ☐ Datos pero resultado incorrecto
  - ☐ Otro tipo de datos que el esperado
  - ☐ Cálculos anteriores/posteriores a la operación
  - ☐ Crear Suite
- Ejercicio Book
- Ejercicio Bank

46

## Referencias / Enlaces de interés

1. <https://www.eclipse.org>
2. <https://junit.org/junit5/>