Tema 3
Testing

JORGE BARÓN ABAD

JORGE.BARON@ADAITS.ES

3.2

JUnit


JUnit 5

JAVA Unit Testing

+ JUnit 5

JUnit

Es un framework(conjunto de clases y librerías) para hacer pruebas unitarias, automáticas y repetibles en Java

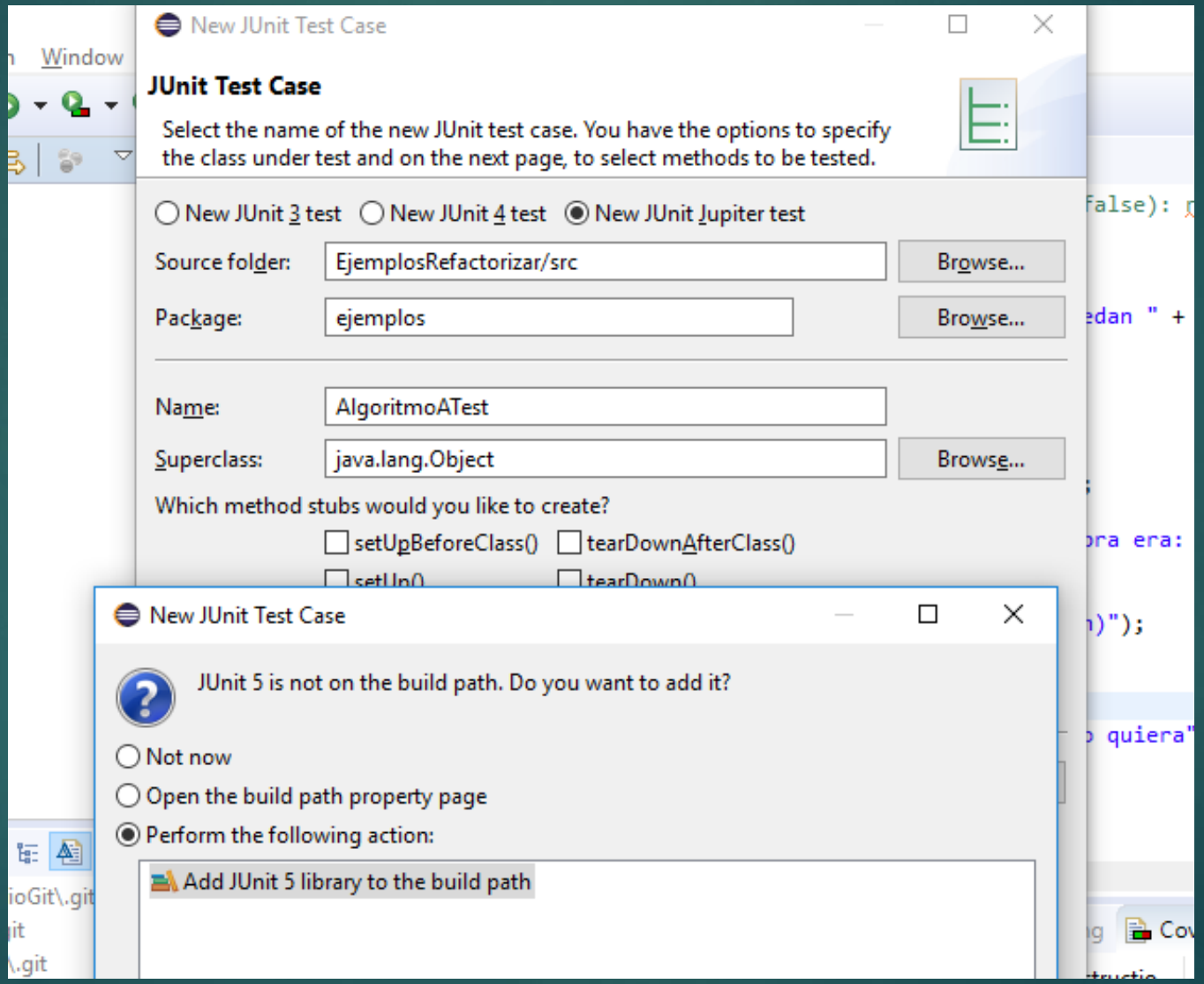
JUnit

- Ventajas:
  - Facilidad para programar y realizar pruebas.
  - Pruebas automáticas.
  - Validación y verificación de la aplicación.
  - Dotar de calidad a la aplicación.
  - Es gratuito.
  - Se encuentra bien mantenido y en constante mejora

JUnit

- Para usar Junit debemos seguir los siguientes pasos:
  - Pinchamos en la clase que queremos hacer pruebas
  - Pulsamos botón derecho
  - Seleccionamos New>Junit Test Case
  - SI no apareciera el punto anterior, Debemos seleccionar Other>Java>Junit>Junit Test Case

JUnit


New JUnit Test Case
JUnit Test Case
Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.
New JUnit 3 test
New JUnit 4 test
New JUnit Jupiter test
Source folder:
EjemplosRefactorizar/src
Browse...
Package:
ejemplos
Browse...
Name:
AlgoritmoATest
Superclass:
java.lang.Object
Browse...
Which method stubs would you like to create?
setUpBeforeClass()
tearDownAfterClass()
setUp()
tearDown()
New JUnit Test Case
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

JUnit

Nuestra primera Prueba:

Tenemos la siguiente clase con una función:

```
public class Operaciones {

    public int sumar(int numero1,int numero2){
      return numero1 + numero2;
      }

}
```

Tenemos la siguiente clase para pruebas

```
Import static
org.junit.jupiter.api.Assertions.*;

class OperacionesTest {


}
```

JUnit

- Vamos a probar la función sumar:

- Debemos crear una función en la clase OperacionesTest con un nombre similar a la función que vamos a probar. Sin parámetro alguno

```
void sumarTest(){


}
```

JUnit

- Junit se sirve de etiquetas para definir como debe comportarse el framework para efectuar las pruebas.
- **@Test =>** Sirve para definir una prueba
- Debemos usar dicha etiqueta e importar la clase correspondiente a esa etiqueta.
- En nuestro ejemplo quedaría así:

```
@Test
void sumarTest(){

}
```

JUnit

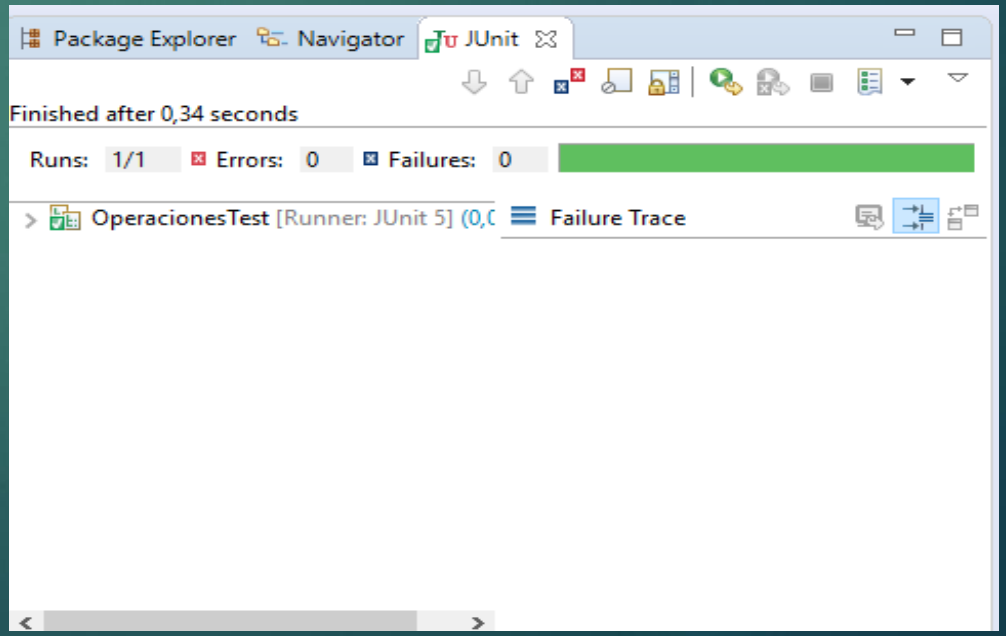
- Ahora debemos incluir una sentencia que sirva para ejecutar la función a probar con unos parámetros de entrada que nosotros definamos y a la vez confirmar el resultado que esperamos de dicha función.

- Para ello debemos usar el método estático con el siguiente proptotipo:
  - **assertEquals(resultadoEsperadoExpresion,expresion)**
- En nuestro caso se usaría de la siguiente forma:

```
void sumarTest(){
    Operaciones o = new Operaciones();
    assertEquals(10, o.sumar(6,4)); //Test Correcto
}
```

JUnit

- Ahora debemos probar que nuestro Test funciona bien, por lo tanto debemos ejecutarlo.
- Tenemos que pulsar en el botón derecho encima de la clase de Test y seleccionamos Run As > Junit Test Case.
- A continuación debería aparecernos en la parte izquierda del IDE, el panel de control de las pruebas. Indicando los test que se han superado y los que no:

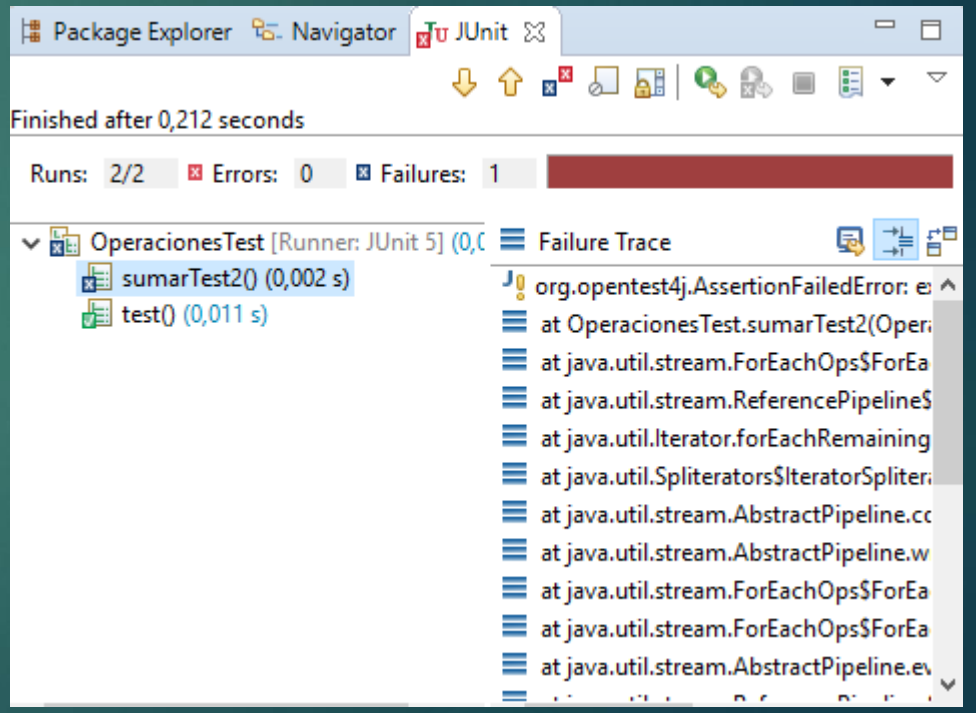


JUnit

- Ahora probamos a crear un test que no vaya a superarse:

```
@Test
    void sumarTest2(){
        Operaciones o = new Operaciones();
        assertEquals(10, o.sumar(6,6));
    }
```

- El resultado debería ser el siguiente:



JUnit: Assertions

- Ahora ya podemos empezar a hacer pruebas. Pero antes vamos a conocer otros métodos **assertions**, para realizar pruebas:
  - **assertNotEquals(resultadoEsperadoExpresion,expresion) =>** Se supera la prueba si el resultado esperado no es igual a la expreisón indicada.
  - **assertFalse(expresion) y assertTrue(expresion)=>** La prueba se supera si la expresión que se le pasa por parámetro devuelve falso o true, respectivamente.
  - **assertNotNull(expresion) y assertNull(expresion)=>** La prueba se supera si la expresión que se le pasa por parámetro devuelve Null o no, respectivamente

JUnit: Assertions

- **assertNotSame(expresion1,expresion2) y assertSame(expresion1, expresion2)=>** La prueba se supera si la expresión primera no es igual o si es igual, respectivamente, a la segunda expresión. Este método se diferencia de assetNotEquals y assertEquals, porque se usa cuando las expresiones devuelven objetos.

- **assertArrayEquals(arrayEsperado,arrayExoresion) =>** Se supera la prueba si el array esperado es igual al array de la expresion.

- Existen más tipos de **assertions**, creados por otras librerías que pueden añadirse a Junit, pero con este grupo de métodos, tenemos suficiente.

JUnit

- Si queremos que al ejecutar nuestro conjunto de pruebas, el texto que identifique a la prueba no sea el nombre de la función, sino uno específico, debemos usar la siguiente etiqueta:

```
@DisplayName('Nombre del Test')
void sumarTest(){
}
```

Junit: Preparación

- Junit ofrece una serie de etiquetas, para definir varias funciones que sirvan para preparar todos los elementos que necesitan todos los test de la clase antes de ejecutarse, y para gestionar como deben comportarse si fallan o superan los assertions indicados:
  - @**BeforeAll** => Sirve para crear una función que va a preparar los elementos que vamos a necesitar antes de ejecutar todos los test. Debe ser un método estático
  - @**BeforeEach** => Sirve para que preparar los elementos que vamos a necesitar antes de ejecutar cada Test. No puede ser un método estático
  - @**AfterAll** => Sirve para ejecutar una o varias operaciones cuando terminan todos los test. Debe ser un método estático.
  - @**AfterEach**=> Sirve para ejecutar una o varias operaciones cuando terminan todos cada Test. No puede ser un método estático.


THE
END