Temporalización:
27 oct – 17 nov
Autora: Cristina Álvarez Villanueva
Revisado por:
Fco. Javier Valero Garzón
M.ª Carmen Safont

1. JUSTIFICACIÓN DE LA PRÁCTICA

En las prácticas anteriores hemos hecho pruebas de forma manual a partir de un código. En esta práctica vamos a hacerlas **automáticamente con la herramienta JUnit.** Se trata de una herramienta que permite hacer pruebas unitarias automatizadas. Está integrada en **Eclipse y en Netbeans**, por lo que no es necesario descargarse ningún paquete para usarla. Recuerda que las pruebas se hacían sobre una clase para observar su comportamiento independientemente del resto de clases de la aplicación. Pero hemos de tener en cuenta que a veces una clase depende de otra.

2. CREACIÓN DE UNA CLASE DE PRUEBA

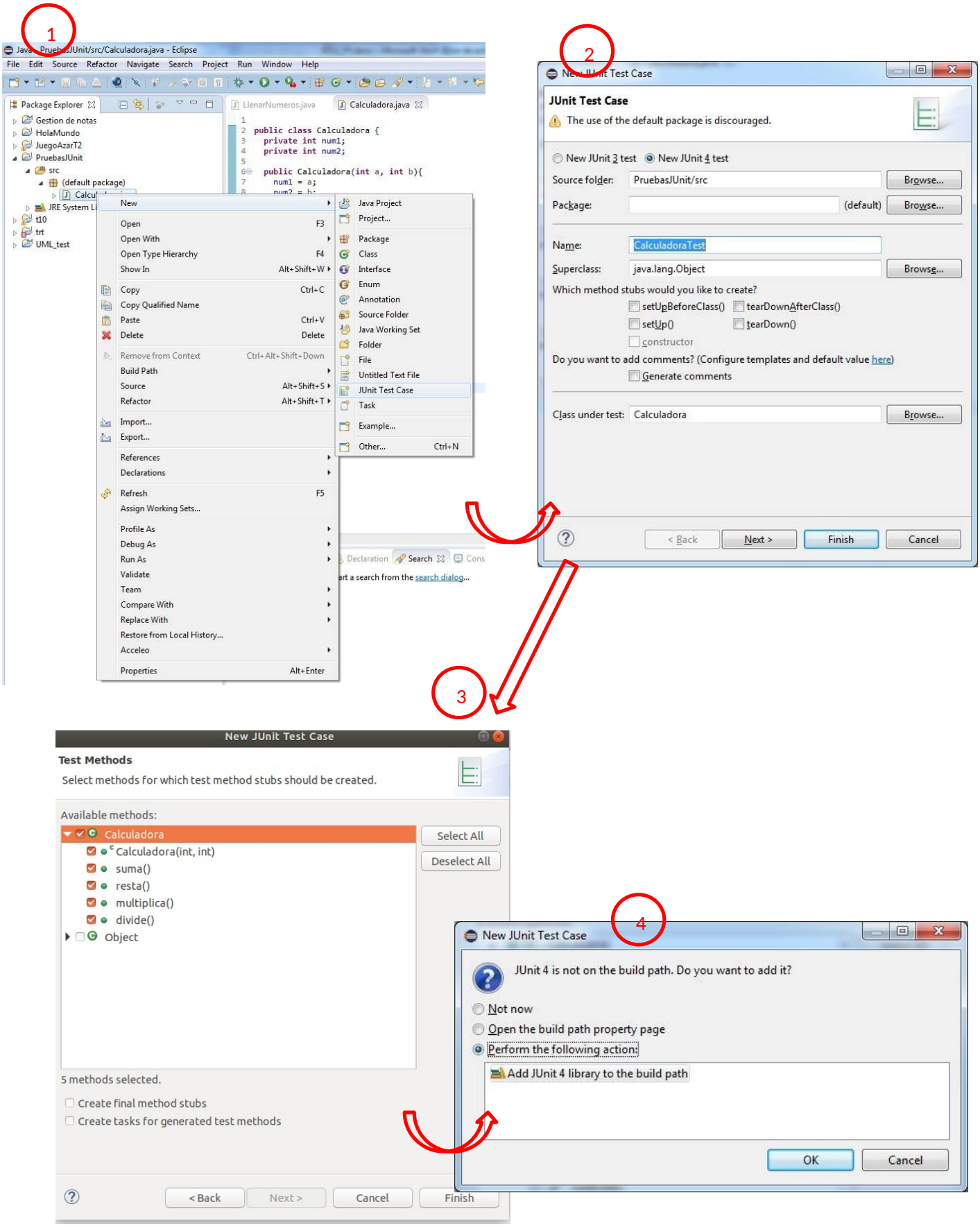
Crea un proyecto nuevo en Eclipse (**PruebaJUnit**) y guarda en él la clase **Calculadora.java** que será así:

Acordaros de desmarcar Create module-info.java file y cuando creeis la clase ponerle nombre al paquete

```
public class Calculadora {

	private int num1;
	private int num2;

	public Calculadora(int a, int b) {
		num1 = a;
		num2 = b;
	}

	public int suma() {
		int resul = num1 + num2;
		return resul;
	}

	public int resta() {
		int resul = num1 - num2;
		return resul;
	}

	public int multiplica() {
		int resul = num1 * num2;
		return resul;
	}

	public int divide() {
		int resul = num1 / num2;
		return resul;
	}
} // fin clase
```

Ahora vamos a crear la clase de prueba. Con la clase *Calculadora* seleccionada, pulsa botón derecho del ratón y elige ***New > JUnit Test Case.*** Otra manera es desde el **menú *File > New > JUnit Test Case.***

Debemos seleccionar ***New JUnit 4 test*** y dejamos el resto de opciones por defecto. El nombre de la clase que se generará será ***CalculadoraTest***. Pulsa el botón ***Next*** y selecciona los métodos a probar **(marca los 5)** y pulsa **Finish**. Se abrirá una ventana indicando que la librería JUnit 4 no está incluida en el proyecto, pulsa **OK** para que se incluya y acabará. Se muestra a continuación:



La clase de prueba se crea automáticamente, y se observan una serie de características:

- Se crean 5 **métodos de prueba**, uno para cada uno de los elegidos
- Los métodos son públicos, no devuelven nada y no reciben ningún argumento
- El nombre de cada **método incluye la palabra *test*** al principio (*testSuma()*, *testResta()*, *testMultiplica()*...)
- Sobre cada uno de los métodos aparece la anotación *@Test* que indica al compilador que es un método de prueba
- Cada uno de los métodos de prueba tiene una llamada al método ***Fail()* con un mensaje** indicando que todavía no se ha implementac el método. Fail hace que el test termine con un fallo lanzando un mensaje.

```
1  package uno;
2
3  import static org.junit.Assert.*;
6
7  public class CalculadoraTest {
8
9      @Test
10     public void testCalculadora() {
11         fail("Not yet implemented");
12     }
13
14     @Test
15     public void testSuma() {
16         fail("Not yet implemented");
17     }
18
19     @Test
20     public void testResta() {
21         fail("Not yet implemented");
22     }
23
24     @Test
25     public void testMultiplica() {
26         fail("Not yet implemented");
27     }
28
29     @Test
30     public void testDivide() {
31         fail("Not yet implemented");
32     }
33
34 }
```

3. PREPARACIÓN Y EJECUCIÓN DE LAS PRUEBAS

Recuerda que con **JUnit** algunos de los métodos principales son:

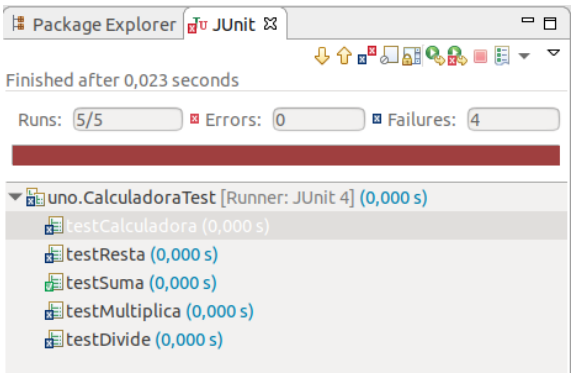
MÉTODOS	MISIÓN
assertTrue(boolean expresión) assertTrue(String mensaje, boolean expresión)	Comprueba que la expresión se evalúe a *true*. Si no es *true* y se incluye el String, al producirse error se lanzará el *mensaje*
assertFalse(boolean expresión) assertFalse(String mensaje, boolean expresión)	Comprueba que la expresión se evalúe a *false*. Si no es *false* y se incluye el String, al producirse error se lanzará el *mensaje*
assertEquals(valorEsperado,valorReal), assertEquals(String mensaje, valorEsperado,valorReal)	Comprueba que el *valorEsperado* sea igual al *valorReal*. Si no son iguales y se incluye el String, entonces se lanzará el *mensaje*. *ValorEsperado* y *valorReal* pueden ser de diferentes tipos
assertNull(Object objeto), assertNull(String mensaje, Object objeto)	Comprueba que el *objeto* sea null. Si no es null y se incluye el String al producirse error se lanzará el *mensaje*.
assertNotNull(Object objeto), assertNotNull(String mensaje, Object objeto)	Comprueba que el *objeto* no sea null. Si es null y se incluye el String, al producirse error se lanzará el mensaje
assertSame(Object objetoEsperado, Object objetoReal) assertSame(String mensaje, Object objetoEsperado, Object objetoReal)	Comprueba que *objetoEsperado* y *objetoReal* sean el mismo objeto. Si no son el mismo y se incluye el String, al producirse error se lanzará el *mensaje*
assertNotSame(Object objetoEsperado, Object objetoReal) assertNotSame(String mensaje, Object objetoEsperado, Object objetoReal)	Comprueba que *objetoEsperado* no sea el mismo objeto que *objetoReal*. Si son el mismo y se incluye el String, al producirse error se lanzará el *mensaje*
fail() fail(String mensaje):	Hace que la prueba falle. Si se incluye un String la prueba falla lanzando el *mensaje*

Vamos a crear el código de prueba para el método ***testSuma()*** que probará el método *suma()* de la clase *Calculadora()*. Lo primero que vamos a hacer es crear una instancia de la clase *Calculadora*, llamamos al método *suma()* llevando los valores a sumar, por ejemplo 20 y 10, y comprobamos los resultados con el método ***assertEquals()*.** En el primer parámetro de este último método escribimos el **resultado esperado** al realizar el método *suma()*, en este caso 30, y como segundo parámetro asignamos el **resultado obtenido** al llamar a dicho método. **Quedaría así:**

```
@Test
public void testSuma() {
	Calculadora calcu = new
Calculadora(20,10);
	int resultado=calcu.suma();
	assertEquals(30,resultado);
}
```

```
1  package uno;
2
3  import static org.junit.Assert.*;
6
7  public class CalculadoraTest {
8
9      @Test
10     public void testCalculadora() {
11         fail("Not yet implemented");
12     }
13
14     @Test
15     public void testSuma() {
16         Calculadora calcu = new Calculadora(20,10);
17         int resultado=calcu.suma();
18         assertEquals(30,resultado);
19     }
20
21     @Test
22     public void testResta() {
23         fail("Not yet implemented");
24     }
25
26     @Test
27     public void testMultiplica() {
28         fail("Not yet implemented");
29     }
30
31     @Test
32     public void testDivide() {
33         fail("Not yet implemented");
34     }
35
36 }
```

Ahora pulsamos el botón **Run** para ejecutar el test. Se mostrarán algunos errores, porque no hemos implementado todos los test de pruebas. También podemos ejecutar la clase de prueba pulsando sobre la clase con el botón derecho del ratón y seleccionando ***Run As > JUnit Test***. En ambos casos se abre la pestaña de **JUnit** donde se muestran los resultados de ejecución de las pruebas:

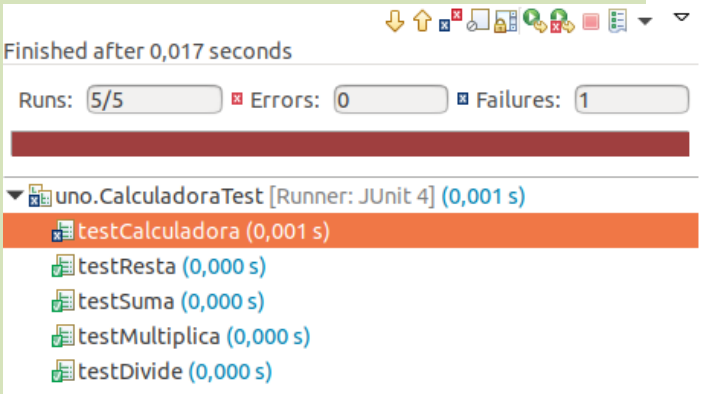


Al lado de cada prueba aparece un icono con una marca: una marca de verificación **verde** indica **prueba exitosa,** un **aspa azul indica fallo** y una **roja** indica **error**. El resultado de nuestra ejecución muestra *Runs: 5/5 Errors: 0 Failures: 4*; Esto indica que se han realizado 5 pruebas, ninguna de ellas ha dado error y 4 han dado fallo.

En JUnit, **fallo** es una comprobación que no se cumple mientras que **error** es una excepción durante la ejecución del código. En esta prueba solo se ha procedido bien con el método *testSuma()* (tiene aspa verde) y el resto han fallado (aspa azul) con lo que han llamado al método *fail()*.

Ahora rellena el resto de casos como se muestra:

```
	@Test
	public void testResta() {
		Calculadora calcu = new Calculadora(20,10);
		int resultado = calcu.resta();
		assertEquals(10, resultado);

	}

	@Test
	public void testMultiplica() {
		Calculadora calcu = new Calculadora(20,10);
		int resultado = calcu.multiplica();
		assertEquals(200, resultado);

	}

	@Test
	public void testDivide() {
		Calculadora calcu = new Calculadora(20,10);
		int resultado = calcu.divide();
		assertEquals(2, resultado);

	}
```

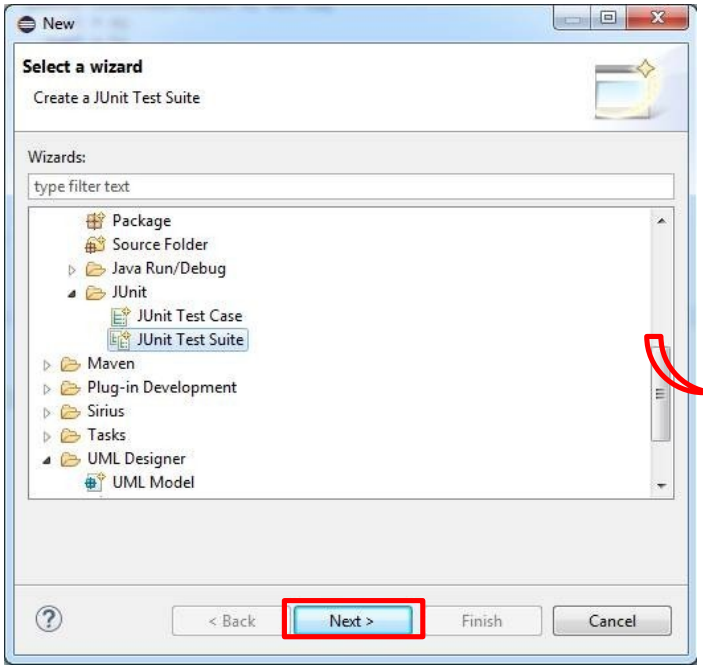


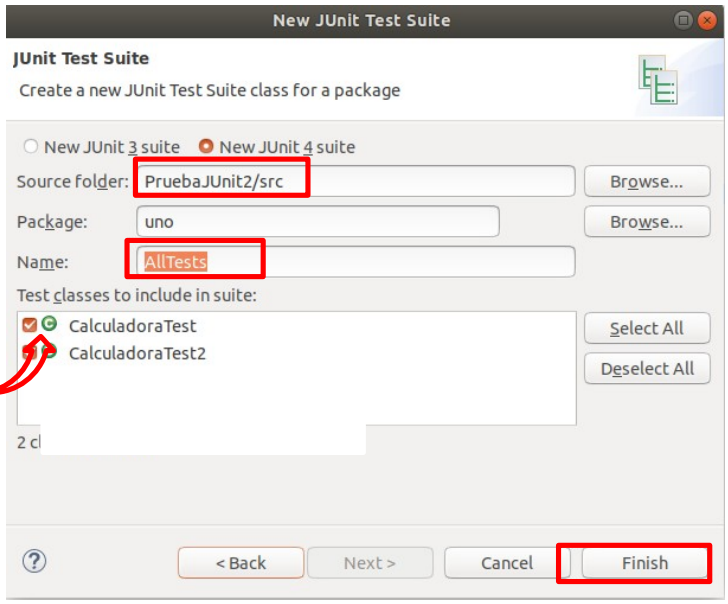
Si ejecutamos ahora la prueba nos devolverá *Runs: 5/5, Errors: 0, Failures: 1;* Esto nos indica que **se han hecho 5 pruebas, ninguna ha dado error y 1 ha dado fallo.**

4. SUITE DE PRUEBAS

A veces nos interesa ejecutar varias clases de prueba una tras otra. Para ello, tenemos el mecanismo **Test Suites** que agrupa varias clases de prueba para que se ejecuten una tras otra.
Crear un CalculadoraTest2.java rellenando los métodos con otras pruebas.

Hasta ahora tenemos las clases de pruebas CalculadoraTest.java, CalculadoraTest2.java,
Vamos a suponer que son pruebas que queremos hacer a la vez. Entonces, para crear la suite de pruebas pulsamos en la opción de menú: ***File > New > Other > Java > JUnit > JUnit Test Suite***. Pulsamos luego el botón *Next* y elegimos las clases que formarán parte de la suite de pruebas, marcamos la opción *New JUnit 4 suite* y le damos un nombre ("AllTests") dejando marcadas las dos clases que queremos que se prueben, y *Finish*.





Se generará automáticamente la clase AllTest.java:

```
1  package uno;
2
3  import org.junit.runner.RunWith;
4  import org.junit.runners.Suite;
5  import org.junit.runners.Suite.SuiteClasses;
6
7  @RunWith(Suite.class)
8  @SuiteClasses({ CalculadoraTest.class, CalculadoraTest2.class })
9  public class AllTests {
10
11 }
12
```

En ella vemos que @RunWith(Suite.class) indica a JUnit que la clase es una suite de pruebas. Por otro lado, @SuiteClasses() muestra las clases que forman parte del conjunto de pruebas y que se ejecutarán. Dentro de la clase no se genera ningún código. Ejecutar el fichero **AllTests.java** y entender los resultados.

5. BIBLIOGRAFÍA Y ENLACES

Álvarez, C. (2014): "Entornos de desarrollo", CEEDCV
Ramos, A.; Ramos, MJ (2014): *Entornos de desarrollo*, Garceta, Madrid