# Actividad 5: Pruebas Automáticas

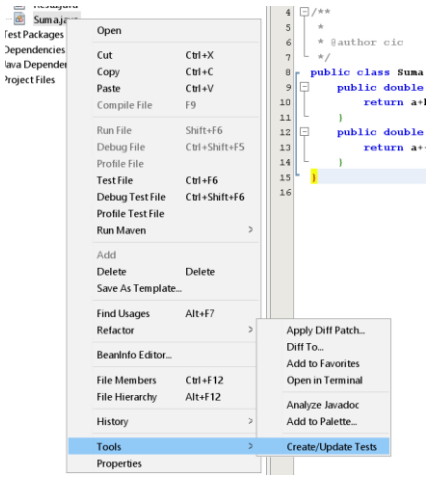
En actividades anteriores hemos realizado pruebas unitarias de forma manual, pero existen herramientas que nos permiten realizar estas pruebas de forma automática. La herramienta que vamos a utilizar es JUnit, que se encuentra integrado en NetBeans.

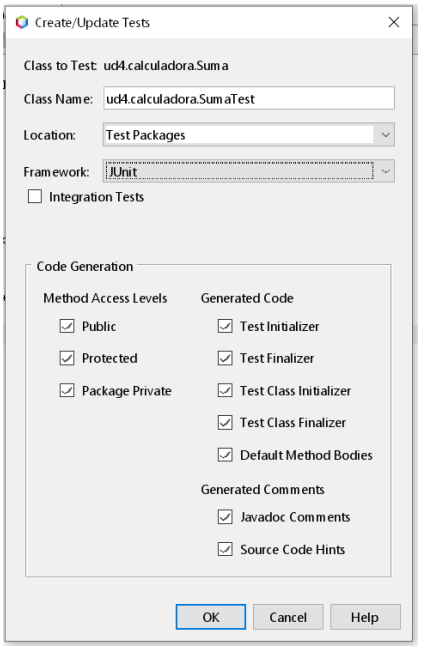
En esta actividad se realizan dos ejercicios guiados con varios apartados para practicar.

**EJERCICIO1**

Para probar el funcionamiento de JUnit vamos a probar un programa Calculadora con tres clases; un main llamado Calculadora y dos clases Suma y Resta con las operaciones de nuestra calculadora. Se adjunta el proyecto de NetBeans llamado Calculadora.

Ahora vamos a testear la clase Suma. Pulsamos botón derecho sobre la clase Suma y en el menú contextual seleccionamos **Tools -> Create/Update tests**.




Dejamos las opciones marcadas por defecto.

Nos ha creado una nueva clase de prueba que tiene como nombre, el nombre de nuestra clase, añadido la palabraTest.

Se produce un error al generar esta clase ya que se duplican cuatro métodos. Borramos los 4 primeros métodos para que nuestro proyecto de prueba compile correctamente.

Comprobamos que nuestro fichero **pom.xml** contenga la estructura que figura en los apuntes y que la **importación** de las librerías sea la correcta.

Se ha creado la clase SumaTest, que incluye los métodos para probar la clase Suma. Esta clase se crea con código de testeo, lo vamos a modificar para adaptarlo a nuestros casos de prueba.

Nuestros casos de prueba para getSuma e incrementa son:

| **Método a testear** | **Entrada** | **Salida esperada** |
|---|---|---|
| **GetSuma** | a=1, b=1 | 2 |
| **Incrementa** | a=1 | 2 |

Con estos casos de prueba modificamos el código como se muestra a continuación:

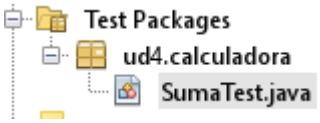
```
/**
 * Test of getSuma method, of class Suma.
 */
@org.junit.jupiter.api.Test
public void testGetSuma() {
    System.out.println("getSuma");
    double a = 1;
    double b = 1;
    Suma instance = new Suma();
    double expResult = 2;
    double result = instance.getSuma(a, b);
    assertEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to f
    //fail("The test case is a prototype.");
}

/**
 * Test of incrementa method, of class Suma.
 */
@org.junit.jupiter.api.Test
public void testIncrementa() {
    System.out.println("incrementa");
    double a = 1;
    Suma instance = new Suma();
    double expResult = 2;
    double result = instance.incrementa(a);
    assertEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to f
    //fail("The test case is a prototype.");
}
```

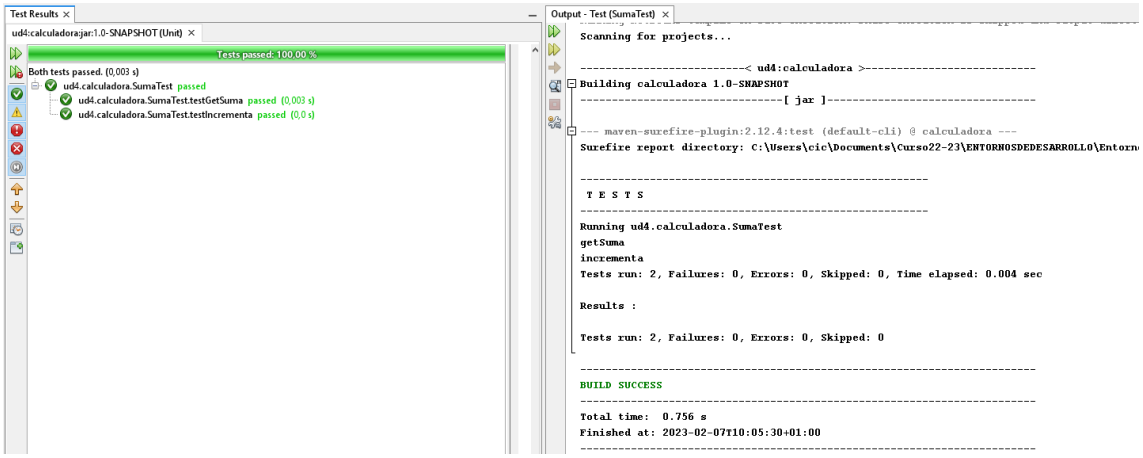
Ahora vamos a ver la funcionalidad de las instrucciones de la clase SumaTest.

- La anotación **@Test** identifica el método como método de testeo.
- **assertEquals(esperado,real)** comprueba si el valor esperado es igual al que nos devuelve el método testeado. En nuestro código hemos añadido un tercer parámetro, un 0, este parámetro es la sensibilidad del assertEquals, que determina la diferencia tolerable entre los valores a evaluar.
- Comentamos el método fail porque es una instrucción que fuerza un test fallido.

Vamos a probar el funcionamiento de nuestro test. Para ello pulsamos botón derecho sobre SumaTest, que se ha creado en un paquete nuevo, y seleccionamos Test File.



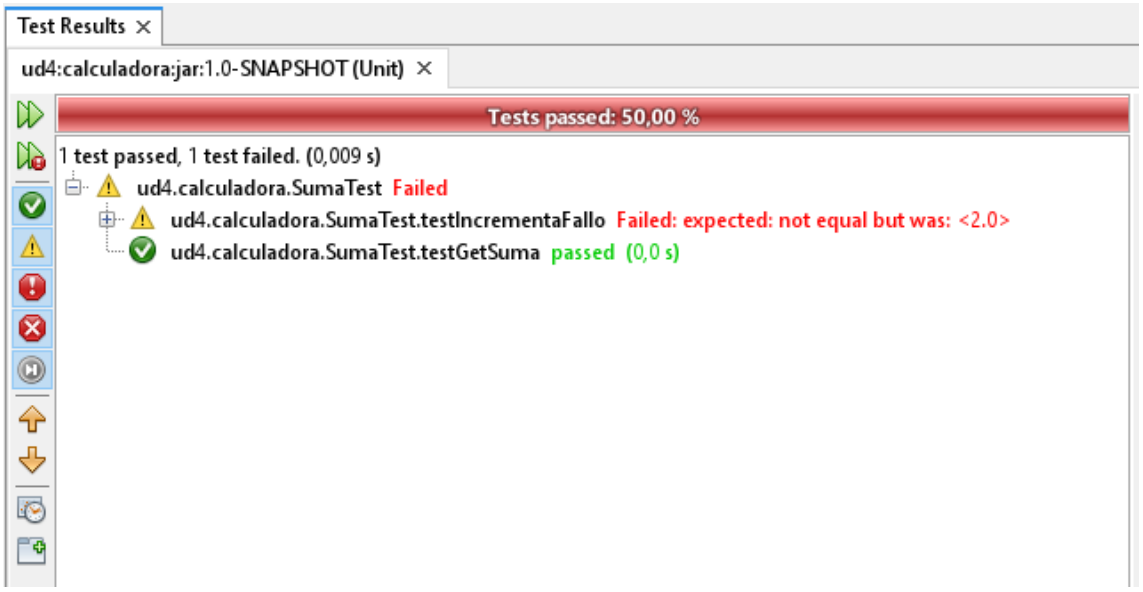

Si los valores introducidos son correctos el resultado será que se pasa el test.




También podemos testear casos de prueba con otras sentencias assert por ejemplo, si lo que queremos detectar es el caso, de que no sean iguales el resultado esperado con el resultado de la función podemos usar la aserción **assertNotEquals**, que hará que se pase el test cuando los valores a comparar no coincidan.

Para realizar una prueba vamos a crear otro método de test denominado testIncrementaFallo simulando los creados.

```
/**
 * Test of incrementa method, of class Suma.
 */
@org.junit.jupiter.api.Test
public void testIncrementaFallo() {
    System.out.println("incrementa fallo");
    double a = 1;
    Suma instance = new Suma();
    double expResult = 2;
    double result = instance.incrementa(a);
    assertNotEquals(expResult, result, 0.0);
    // TODO review the generated test code and remove the default call to fa
    //fail("The test case is a prototype.");
}
```



Realiza el test de la clase Resta

**EJERCICIO 2**

Se trata de unas clases que definen un carrito de compras usado en las aplicaciones de comercio electrónico. Proyecto CarritoCompra

La funcionalidad del carrito de compras va a estar constituida por dos clases: Producto y Carrito.

- La clase Producto representa los productos que se pueden comprar añadiendo al carrito de compras.
- la clase Carrito representa al carrito de compras donde se pueden introducir productos a comprar como un array de productos.

La clase **Producto** tiene:

- Atributos: el código, nombre, descripción y precio.
- Constructor con los cuatro atributos.
- Métodos get de los atributos.
- Método toString con la información del producto.

La clase **Carrito** tiene:

- Atributos: el array de productos y el contador de productos
- Constructor donde crea el array de productos e inicializa el contador de productos
- Métodos:
    - Obtener el total de la compra, obtenerPrecio()
    - Añadir un ítem de producto al carrito, insertar( producto)
    - Quitar un ítem de producto al carrito, borrar( producto)
    - Obtener la cantidad de ítems de productos que hay en un carrito, getContProductos()
    - Obtener el array de productos del carrito, getProductos()

se deberá verificar que el funcionamiento del programa sea correcto, mediante la realización de unas pruebas unitarias más precisas.

Juego de datos de producto:

- Código ->"P001"
- Nombre ->"Plasma TV LG 32"
- Descripción ->"Plasma TV with TDT Decod. and high resolution Screen"
- Precio ->699.99

Para probar el método añadir un producto al carro, utilizar el siguiente producto:

- Código ->"P002"
- Nombre ->"DVD player Sanmsung"
- Descripción ->"DVD player with remote control and programming features"
- Precio ->39.99

- Crear y ejecutar los casos de prueba para las funcionalidades de la clase Producto y de la Clase CarritoCompra, utilizando los métodos setUp() y tearDown() para evitar la duplicación de código y poder reutilizar los datos de prueba en cada caso de prueba.

Se deberán comentar los resultados y capturando las correspondientes pantallas sobre las pruebas realizadas.

En este caso queremos tener definido un producto para realizar todas las pruebas de los métodos asociados al producto

```
public class ProductoTest {
    private Producto prod1;
    public ProductoTest() {
    }

    @BeforeAll
    public static void setUpClass() {
    }

    @AfterAll
    public static void tearDownClass() {
    }

    @BeforeEach
    public void setUp() {
        prod1 = new Producto("P001", "Plasma TV LG 32", "Plasma TV with TDT Dec
    }

    @AfterEach
    public void tearDown() {
        prod1=null;
    }
```

En todos los métodos podrás utilizar el mismo producto, así quedaría testGetCodigo

```
@Test
public void testGetCodigo() {
    System.out.println("getCodigo");
    Producto instance = prod1;
    String expResult = "P001";
    String result = instance.getCodigo();
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fa
    //fail("The test case is a prototype.");
}
```

Para realizar las pruebas de la clase carrito vamos a partir de la situación del carrito con el producto primero cargado y para la prueba de insertar utilizaremos el producto2

```
private Carrito cart1;
private Producto prod1;
private Producto prod2;

public CarritoTest() {
}

@BeforeAll
public static void setUpClass() {
}

@AfterAll
public static void tearDownClass() {
}

@BeforeEach
public void setUp() {
    cart1 = new Carrito(5);
    prod1 = new Producto("P001", "Plasma TV LG 32", "Plasma TV with TDT Deco
    cart1.insertar(prod1);
    prod2 = new Producto("P002", "DVD player Sanmsung", "DVD player with rem
}

@AfterEach
public void tearDown() {
    cart1 = null;
    prod1 = null;
    prod2 = null;
}
```

El método de testObtenerPrecio puede ser

```
49      @Test
50      public void testObtenerPrecio() {
51          System.out.println("obtenerPrecio");
52          Carrito instance = cart1;
53          double expResult = 669.99;
54          double result = instance.obtenerPrecio();
55          assertEquals(expResult, result, 0.0);
56          // TODO review the generated test code and remove the default call to fai
57          //fail("The test case is a prototype.");
58      }
```

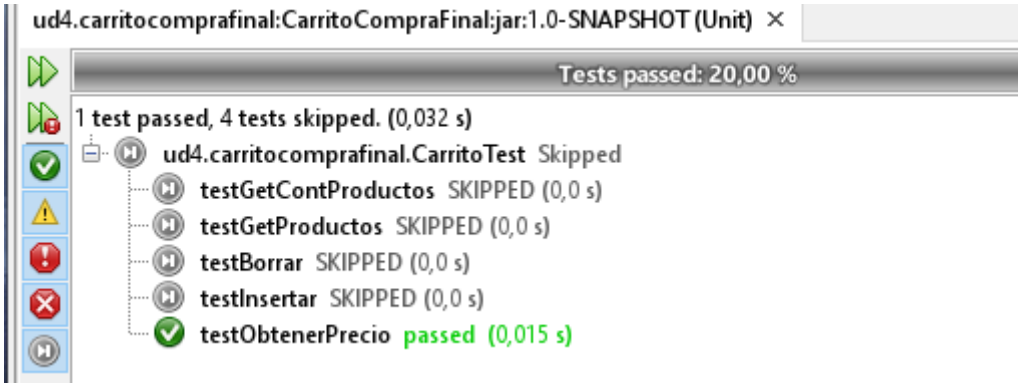
Si solo queremos ejecutar la prueba de este método con pulsar en el icono verde que aparece en la línea 50 es suficiente.

Otra forma es utilizar la anotación @Disabled

La anotación **@Disabled**, Se utiliza para deshabilitar una clase de prueba o un método de prueba. Recuerda que tiene que estar instalado el plugin Maven-surefire-plugin, figura en el resultado como SKIPPED. Ejemplo:

```
@Disabled ("No prueba testInsertar()")
@Test
public void testInsertar() {
    System.out.println("insertar");
    Producto producto = null;
    Carrito instance = null;
    instance.insertar(producto);
    // TODO review the generated test code and remove the default call to f
    fail("The test case is a prototype.");
}
```

Si realizo esta anotación para el resto de los métodos




Ahora realiza las pruebas del resto de métodos del carrito y de los métodos de la clase producto.

**EJERCICIO 3**

En este ejercicio vamos a testear la clase cuenta utilizando JUnit y NetBeans. La clase cuenta incluye como atributos el número de la cuenta y su saldo, un método constructor con un parámetro por cada uno de los atributos de la clase, métodos de consulta y acceso a cada uno de sus atributos, así como los siguientes métodos:

- ingresarDinero(), para incrementar el saldo de la cuenta con el importe recibido como parámetro.
- extraerDinero(), para disminuir el saldo de la cuenta con el importe recibido como parámetro.
- mostrarCuenta(), que muestra el número de la cuenta y su saldo.

Como en las pruebas unitarias afectan a clases y métodos de forma individual no necesitamos el programa al completo, nuestra tarea es testear únicamente los métodos de la clase Cuenta que se adjunta.

Crea la clase CuentaTest y el test para el método getSaldo que se llame testGetSaldo. Para hacer funcionar este método crea un objeto de la clase cuenta con un saldo inicial de 100€. Este método llamará a getSaldo y comprobará si el saldo de la cuenta es de 100€.

Implementa el test de las clases setSaldo, ingresarDinero y extraerDinero con los siguientes casos de prueba:

- setSaldo debe probarse con la cuenta con un saldo inicial de 100€ e introduciendo 100€ mediante el método setSaldo.
- ingresarDinero debe testearse con 100€ como saldo inicial de la cuenta e ingresando 300€ mediante el método ingresarDinero.
- extraerDinero debe probarse con un saldo inicial en la cuenta de 100€ y restando 40€ con el método extraerDinero.

Pon la anotación necesaria para el código de los test no implementados y comprueba que se supera el test de getSaldo, setSaldo, ingresarDinero y extraerDinero y salta los que tienen la anotación

## Tratamiento de excepciones

También podemos crear métodos de prueba que comprueben sí, en determinados casos, en un método, se ha producido una excepción si esta estaba prevista.

Para comprobar que se lanza una excepción cuando el saldo de la cuenta es negativo, se crea un nuevo método testExtraerDineroNegativo generando un caso de prueba que dé lugar a esta situación. Para ello, en el caso de prueba, se crea una cuenta con 100€ inicialmente y se extraen 250€ mediante el método extraerDinero.

Para realizar este test vamos a utilizar un try-catch y la instrucción fail, que fuerza el fallo de un test. Implementa el caso de uso mencionado como se muestra en la siguiente imagen y prueba sí se superan los test. El resultado debe ser de 5 test superados.

```
@Test
public void testExtraerDineroNegativo() {
    try{
        System.out.println("ingresarDinero - prueba negativo");

    Cuenta cuenta1 = new Cuenta ("ES1234567890", 100);
    cuenta1.extraerDinero(250);
    fail("ERROR. Se debería haber lanzado una excepción.");
    }catch(ArithmeticException ae){
        //test superado
    }
```

**EJERCICIO 4**

Utilizando JUnit, realiza el test unitario de los casos de prueba del ejercicio 5 y 6 de la actividad 1 de esta unidad.