UD04: Junit en Netbeans (Ejercicios)

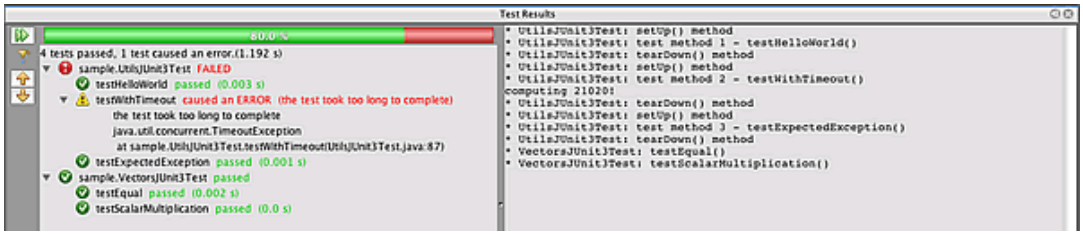
1. Introducción.

Las pruebas de software son parte esencial del ciclo de desarrollo. La elaboración y mantenimiento de unidad, pueden ayudarnos a asegurar que los los métodos individuales de nuestro código, funcionan correctamente. Los entorno de desarrollo, integran frameworks, que permiten automatizar las pruebas.

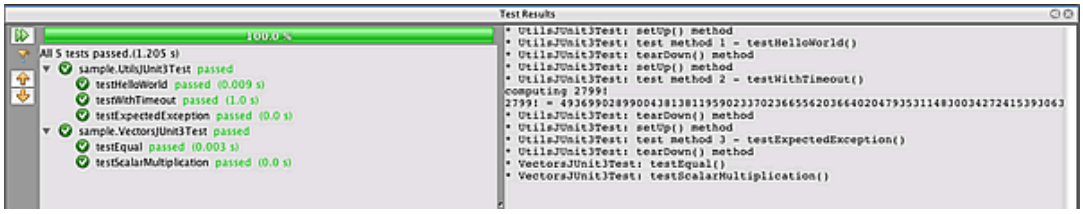
En el caso de entornos de desarrollo para Java, como NetBeans y Eclipse, nos encontramos con el framework JUnit. JUnit es una herramienta de automatización de pruebas que nos permite de manera rápida y sencilla, elaborar pruebas. La herramienta nos permite diseñar clases de prueba, para cada clase diseñada en nuestra aplicación. Una vez creada las clases de prueba, establecemos los métodos que queremos probar, y para ello diseñamos casos de prueba. Los criterios de creación de casos de prueba, pueden ser muy diversos, y dependerán de lo que queramos probar.

Una vez diseñados los casos de prueba, pasamos a probar la aplicación. La herramienta de automatización, en este caso Junit, nos presentará un informe con los resultados de la prueba. En función de los resultados, deberemos o no, modificar el código.

- Algunos test no superados:



- Todos los tests superados:



Los entornos de desarrollo más extendidos, que se utilizan para implementar aplicaciones Java, tales como NetBeans o Eclipse, incorporan un plugin para trabajar con Junit nos va a servir para realizar pruebas unitarias de clases escritas en Java, dentro de un entorno de pruebas. Es un framework con muy pocas clases fácil de aprender y de utilizar.

Una vez que hemos diseñado nuestra aplicación, y la hemos depurado, procedemos a probarla. En el caso del ejemplo, disponemos de una clase, de nombre `Ccuenta`, donde se han definido una serie de métodos.

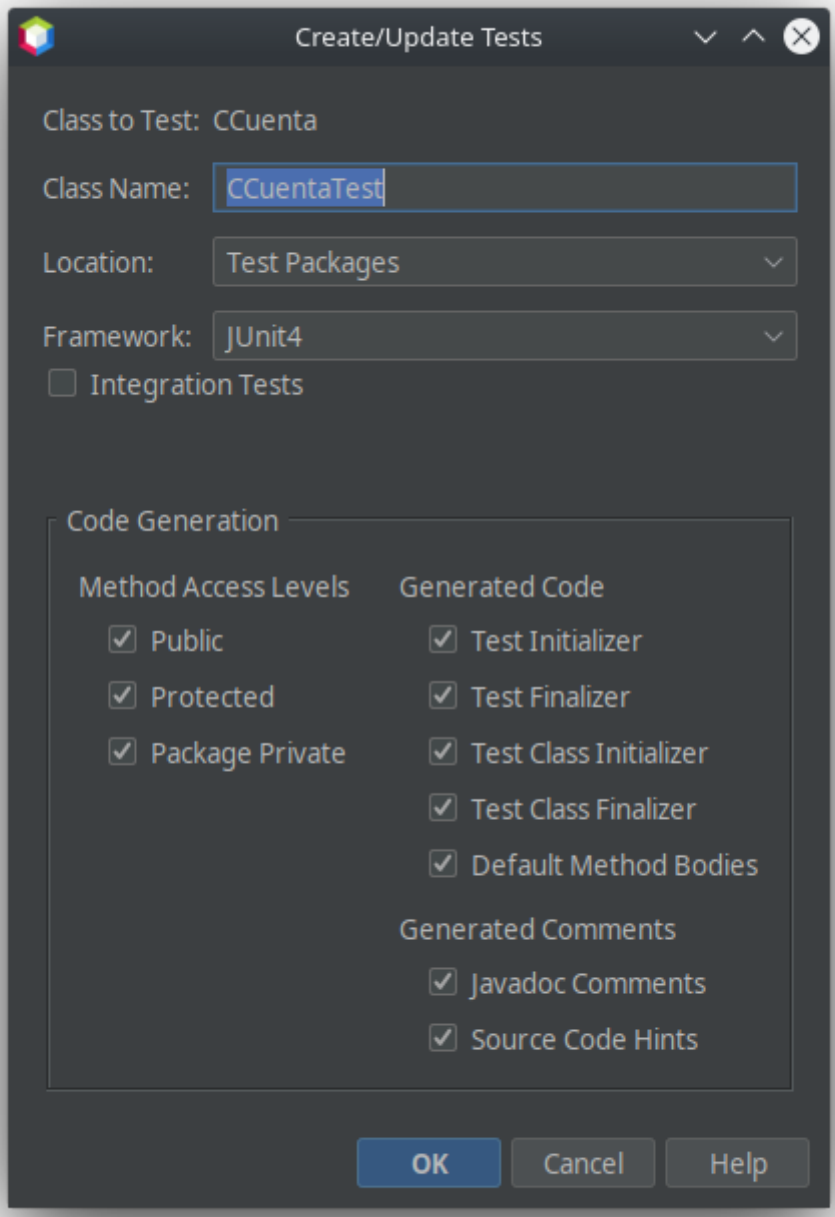
El objetivo va a ser el diseño y ejecución de algunos casos de prueba.

2. Inicio de Junit

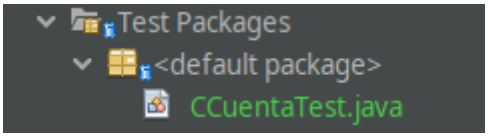
Para iniciar Junit, seleccionada en la ventana de proyectos la clase a probar, abrimos el menú contextual y seleccionamos `Tools` > `Create/Update Tests`.

Nos aparece un formulario donde debemos indicar el nombre de la clase. Puesto que vamos a probar la clase `Ccuenta`, por convenio es recomendable llamar a la clase de prueba `CcuentaTest`. Esta clase se va a insertar en un nuevo paquete de nuestro proyecto, denominado `Test Packages` (Paquetes de pruebas). Nos da a elegir entre `JUnit`, `TestNG` y `JUnit4`. Son las dos versiones de JUnit disponibles en NetBeans 12.X. En nuestro caso, elegimos `JUnit4`, desmarcamos la opción `Integration Tests`.

Como se aprecia en el formulario, JUnit va a generar los métodos que aparecen seleccionados. En nuestro caso lo vamos a dejar tal cual, aunque luego van a ser modificados en el código.



Al pulsar el botón `OK` nos aparecen un nueva clase de nombre `CcuentaTest`, que contiene los métodos que estaban seleccionados en el formulario anterior, con un código prototipo. Es en ese código en el que el programador creará sus casos de prueba.



El diseño de los casos de prueba, requiere que se establezcan criterios que garanticen que esa prueba tiene muchas probabilidades de encontrar algún error no detectado hasta el momento.

3. Casos de prueba

El paso anterior genera una serie de métodos que van ligados a una serie de anotaciones. Comenzamos por conocer los inicializadores y finalizadores:

- Para la clase:
    - `setUpClass()` (`@BeforeClass`): Sólo puede haber un método con este marcador, es invocado una vez al principio de todos los test. Se suele usar para inicializar atributos.
    - `tearDown()` (`@AfterClass`): Sólo puede haber un método con este marcador y se invoca al finalizar todos los test.
- Para cada uno de los tests:
    - `setUp()` (`@Before`): Se ejecuta antes de de cada test.
    - `tearDown()` (`@After`): Se ejecuta después de cada test.

  Se utilizan para inicializar y finalizar las condiciones de prueba, como puede ser la creación de un objeto, inicialización de variables, etc. En algunos casos, no es necesario utilizar estos métodos, pero siempre se suelen incluir.

A continuación, debemos conocer las anotaciones:

- `@Ignore`: Los métodos marcados con esta anotación no serán ejecutados.
- `@Test`: Representa un test que se debe ejecutar.
- `@Test (timeout=X)` el test será válido si se ejecuta antes de `X` milisegundos.

Y por último, es necesario conocer las aserciones. Los método `assertXXX()`, se utilizan para hacer las pruebas. Estos métodos, permiten comprobar si la salida del método que se está probando, concuerda con los valores esperados. Las principales son:

- `assertTrue()` evalúa una expresión booleana. La prueba pasa si el valor de la expresión es true.
- `assertFalse()` evalúa una expresión booleana. La prueba pasa si el valor de la expresión es false.
- `assertNull()` verifica que la referencia a un objeto es nula.
- `assertNotNull()` verifica que la referencia a un objeto es no nula.
- `assertSame()` compara dos referencias y asegura que los objetos referenciados tienen la misma dirección de memoria. La prueba pasa si los dos argumentos son el mismo objeto o pertenecen al mismo objeto.
- `assertNotSame()` Compara dos referencias a objetos y asegura que ambas apuntan a diferentes direcciones de memoria. La prueba pasa si los dos argumentos suplidos son objetos diferentes o pertenecen a objetos distintos.
- `assertEquals()` Se usa para comprobar igualdad a nivel de contenidos. La igual de tipos primitivos se compara usando "==", la igual entre objetos se compara con el método `equals()`. La prueba pasa si los valores de los argumentos son iguales.
- `fails()` causa que la prueba falle inmediatamente. Se puede usar cuando la prueba indica un error o cuando se espera que el método que se está probando llame a una excepción.

En este punto, nos disponemos a diseñar los métodos que necesitemos para los casos de prueba.

```
1 /*
2  * To change this license header, choose License Headers in Project Properties.
```

```
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
import org.junit.BeforeClass;

/**
 *
 * @author David Martínez (wwww.martinezpenya.es|iesmre.com)
 */
public class CCuentaTest {

    //Las variables que usaremos para hacer los tests, ojo! son static!.
    static CCuenta cuentaSinParmetros;
    static CCuenta cuentaDavid;
    static CCuenta cuentaPepe;
    static CCuenta cuentaSinSaldo;
    static CCuenta cuentaSaldoMil;

    //Aunque el constructor no es obligatorio, Junit lo añade, observa que no
    //tiene ninguna anotación.
    public CCuentaTest() {
    }

    @BeforeClass
    public static void setUpClass() throws Exception {
        //Este método se ejecutará una sola vez antes de todos los tests
        //en nuestro caso imprimimos el comienzo del TEST.
        System.out.println("INICIO TEST");
    }

    @AfterClass
    public static void tearDownClass() throws Exception {
        //Este método se ejecutará una sola vez al terminar todos los tests
        //en nuestro caso no lo usamos.
    }

    @Before
    public void setUp() {
        //Este método se ejecutará al comienzo de cada Test
        //en nuestro caso imprimiremos el comentario de que comienza la prueba
        //y crearemos aquí los objetos que vamos a necesitar para hacer pruebas.
        System.out.print("Comienza la prueba ");
        cuentaSinParmetros = new CCuenta();
        cuentaDavid = new CCuenta("David", "1234", 50, 0.5);
        cuentaPepe = new CCuenta("Pepe", "5678", 200, 1);
    }

    @After
    public void tearDown() {
        //Este método se ejecutará al finalizar cada Test
        //en nuestro caso imprimiremos el comentario de que ha terminado la prueba
        //Añadimos la variable fin y el calculo respecto al comienzo para saber
```

```
61          //los ms empleados en la prueba.
62          System.out.println("Fin de la prueba\n");
63      }
64
65      //Comenzamos con los tests o pruebas.
66      /**
67       * Test of getNombre method, of class CCuenta.
68       */
69      @Test
70      public void testGetNombre() {
71          System.out.println("getNombre");
72          //CCuenta instance = new CCuenta();
73          //String expResult = "";
74          //String result = instance.getNombre();
75          //assertEquals(expResult, result);
76          // TODO review the generated test code and remove the default call to
    fail.
77          //fail("The test case is a prototype.");
78
79          //Tal y como indica el TODO, debemos modificar el código y sustituirlo
80          //por nuestros tests en este caso podemos definir dos casos de prueba:
81          //nombre nulo
82          assertNull(cuentaSinParmetros.getNombre());
83
84          //nombre "David"
85          assertEquals("David", cuentaDavid.getNombre());
86
87          //nombre "Pepe"
88          assertEquals("Pepe", cuentaPepe.getNombre());
89      }
90
91      /**
92       * Test of setNombre method, of class CCuenta.
93       */
94      @Test
95      public void testSetNombre() {
96          System.out.println("setNombre");
97
98          //Cambiamos el nombre a la cuenta David
99          cuentaDavid.setNombre("David2");
100         assertEquals("David2", cuentaDavid.getNombre());
101
102         //Cambiamos el nombre a la cuenta sin parámetros
103         cuentaSinParmetros.setNombre("Anonimo");
104         assertEquals("Anonimo", cuentaSinParmetros.getNombre());
105     }
106
107     /**
108      * Test of getCuenta method, of class CCuenta.
109      */
110     @Test
111     public void testGetCuenta() {
112         System.out.println("getCuenta");
113
114         //cuenta nulo
115         assertNull(cuentaSinParmetros.getCuenta());
116
117         //cuenta "1234" David
```

```
118         assertEquals("1234", cuentaDavid.getCuenta());
119 
120         //saldo 200 Pepe
121         assertEquals("5678", cuentaPepe.getCuenta());
122     }
123 
124     /**
125      * Test of setCuenta method, of class CCuenta.
126      */
127     @Test
128     public void testSetCuenta() {
129         System.out.println("setCuenta");
130 
131         //Cambiamos la cuenta David
132         cuentaDavid.setCuenta("0000");
133         assertEquals("0000", cuentaDavid.getCuenta());
134 
135         //Cambiamos la cuenta sin parámetros
136         cuentaSinParmetros.setCuenta("4321");
137         assertEquals("4321", cuentaSinParmetros.getCuenta());
138     }
139 
140     /**
141      * Test of getSaldo method, of class CCuenta.
142      */
143     @Test
144     public void testGetSaldo() {
145         System.out.println("getSaldo");
146 
147         //cuenta cero
148         assertEquals(0, cuentaSinParmetros.getSaldo(), 0);
149 
150         //saldo 50
151         assertEquals(50, cuentaDavid.getSaldo(), 0);
152 
153         //nombre 200
154         assertEquals(200, cuentaPepe.getSaldo(), 0);
155 
156         /*
157          * Cuando desea comparar tipos de punto flotante (double o float),
158          * necesita un parámetro adicional para evitar errores de redondeo.
159          * La afirmación se evalúa como se indica a continuación:
160          *  Math.abs (esperado - real) <= delta
161          * Por ejemplo:
162          *  afirmarEquals( unValorDoble, otroValorDoble, 0.001 )
163          */
164     }
165 
166     /**
167      * Test of setSaldo method, of class CCuenta.
168      */
169     @Test
170     public void testSetSaldo() {
171         System.out.println("setSaldo");
172 
173         //Cambiamos el saldo a la cuenta David
174         cuentaDavid.setSaldo(0);
175         assertEquals(0, cuentaDavid.getSaldo(), 0);
```

```
176
177         //Cambiamos el saldo a la cuenta sin parámetros
178         cuentaSinParmetros.setSaldo(1000.0001);
179         assertEquals(1000.0001, cuentaSinParmetros.getSaldo(), 0);
180     }
181
182     /**
183      * Test of getInteres method, of class CCuenta.
184      */
185     @Test
186     public void testGetInteres() {
187         System.out.println("getInteres");
188
189         //interes cero
190         assertEquals(0, cuentaSinParmetros.getInteres(), 0);
191
192         //interes 0.5
193         assertEquals(0.5, cuentaDavid.getInteres(), 0);
194
195         //interes 1
196         assertEquals(1, cuentaPepe.getInteres(), 0);
197     }
198
199     /**
200      * Test of setInteres method, of class CCuenta.
201      */
202     @Test
203     public void testSetInteres() {
204         System.out.println("setInteres");
205
206         //Cambiamos el interes a la cuenta David
207         cuentaDavid.setInteres(0);
208         assertEquals(0, cuentaDavid.getInteres(), 0);
209
210         //Cambiamos el interes a la cuenta sin parámetros
211         cuentaSinParmetros.setInteres(10.01);
212         assertEquals(10.01, cuentaSinParmetros.getInteres(), 0);
213     }
214
215     /**
216      * Test of ingresar method, of class CCuenta.
217      */
218     @Test
219     public void testIngresar() throws Exception {
220         System.out.println("ingresar");
221
222         //ingresamos 100 a la que estaba vacia
223         cuentaSinParmetros.ingresar(100);
224         assertEquals(100, cuentaSinParmetros.getSaldo(), 0);
225
226         //ingresamos 0 a la que tenia 50
227         cuentaDavid.ingresar(0);
228         assertEquals(50, cuentaDavid.getSaldo(), 0);
229     }
230
231     //El caso de Ingresar es un poco especial, porque puede lanzar una excepción
232     //cuando la cantidad es negativa, esos casos lo tratamos con una anotación
233     //especial donde identificas el tipo de excepcion esperada.
```

```
234     @Test(expected = Exception.class)
235     public void testIngresaExcepcion() throws Exception {
236         System.out.println("Excepción ingresar");
237         //intentamos ingresar una cantidad negativa
238         cuentaPepe.ingresar(-200);
239     }
240 
241     /**
242      * Test of retirar method, of class CCuenta.
243      */
244     @Test
245     public void testRetirar() throws Exception {
246         System.out.println("retirar");
247 
248         //retiramos 0 a la que tenia 50
249         cuentaDavid.retirar(0);
250         assertEquals(50, cuentaDavid.getSaldo(), 0);
251 
252         //retiramos 50 a la que tenia 200
253         cuentaPepe.retirar(50);
254         assertEquals(150, cuentaPepe.getSaldo(), 0);
255     }
256 
257     //Lo mismo para la excepción al intentar retirar una cantidad mayor que el
    saldo
258     @Test(expected = Exception.class)
259     public void testRetirarExcepcion() throws Exception {
260         System.out.println("Excepción retirar");
261         //intentamos retirar cuando no hay saldo
262         cuentaSinParmetros.retirar(200);
263     }
264 }
```

Estos métodos intentan probar los métodos de la clase `Ccuenta`. Para ello, teniendo seleccionado el proyecto, accederemos al menú contextual y pulsamos la opción `Test`.

Como se puede comprobar, la prueba sobre el método retirar ha fallado, pero la resta de pruebas sobre los métodos han resultado exitosas. Con esta información, debemos comprobar que el caso de prueba está correctamente diseñado, en cuyo caso, lo que se ha encontrado es un error en el diseño del método `retirar`, y hay que corregirlo. La ventaja de utilizar herramientas automatizadas, es que se facilita la regresión, ya que tenemos diseñado el caso de prueba para el método, así que una vez recodificado el método retirar, podemos volver a probar todos los métodos de manera automatizada.

4. Clase `Ccuenta`

```java
/**
 *
 * @author David Martínez (wwww.martinezpenya.es|iesmre.com)
 */
public class CCuenta {

    // Propiedades de la Clase Cuenta
    private String nombre;
    private String cuenta;
    private double saldo;
    private double interes;

    // Constructor sin argumentos
    public CCuenta() {
    }

    // Constructor con parámetro para iniciar todas las propiedades de la clase
    public CCuenta(String nom, String cue, double sal, double tipo) {
        nombre = nom;
        cuenta = cue;
        saldo = sal;
        interes = tipo;
    }

    //getters & setters
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getCuenta() {
        return cuenta;
    }

    public void setCuenta(String cuenta) {
        this.cuenta = cuenta;
    }

    public double getSaldo() {
        return saldo;
    }

    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }

    public double getInteres() {
        return interes;
    }

```

```
54      public void setInteres(double interes) {
55          this.interes = interes;
56      }
57
58      //Método para ingresar cantidades en la cuenta. Modifica el saldo.
59      public void ingresar(double cantidad) throws Exception {
60          if (cantidad < 0) {
61              throw new Exception("No se puede ingresar una cantidad negativa");
62          }
63          saldo += cantidad;
64      }
65
66      // Método para retirar cantidades en la cuenta. Modifica el saldo.
67      public void retirar(double cantidad) throws Exception {
68          if (cantidad < 0) {
69              throw new Exception("No se puede retirar una cantidad negativa");
70          }
71          if (getSaldo()< cantidad) {
72              throw new Exception("No hay suficiente saldo");
73          }
74          saldo = cantidad;
75      }
76  }
```

5. Actividades

5.1. Junit_1

Tal cual tenemos ahora mismo la clase `Ccuenta` y `CcuentaTest`, hemos descubierto un problema en el método `retirar`. Explica como se lanzan las pruebas desde Netbeans (donde se vea el test que no es satisfactorio), justifica si el problema está en el Test o en el método `retirar`. Realiza las modificaciones oportunas (en el test o en el método `retirar`) para que el test sea satisfactorio explicando paso a paso y con capturas como realiza los test y resultan todos satisfactorios.

Envia la memoria en PDF a la tarea correspondiente de AULES.

5.2. Junit_2

Modifique los métodos de prueba (tests) que considere oportuno para conseguir que para cada uno de los tests se impriman los milisegundos empleados en la prueba. Hay que hacerlo de la manera más eficiente posible, y no repetir código en cada uno de los tests.

Envia la memoria en PDF a la tarea correspondiente de AULES.

6. Fuentes de información

- https://netbeans.apache.org/kb/docs/java/junit-intro.html
- https://www.discoduroderoer.es/como-hacer-una-aplicacion-de-prueba-con-junit/