Práctica 7. Pruebas

Objetivos

Introducir conceptos básicos de pruebas unitarias en sistemas orientados a objetos.

Material Necesario

- Pruebas de caja negra con Junit. www.junit.org
Viene integrado en Eclipse, pero al crear el proyecto hay que incluir la librería.

- Pruebas de caja blanca con Eclemma. www.eclemma.org
Plugin de Eclipse a instalar. Disponible en el site http://update.eclemma.org

Sistema Bancario

Para esta práctica se proporciona parte de la implementación de un sistema bancario en el que hay *Cuentas* y *Tarjetas* asociadas cada una a una cuenta determinada. Las *Tarjetas* pueden ser de *Crédito* o de *Débito.* Las operaciones que se realizan sobre una *Cuenta* o las *Tarjetas de Crédito* quedan registradas en un vector de *Movimientos*. Para más detalle se adjunta un diagrama de clases.

Parte Básica

La parte básica de esta práctica corresponde a la creación de casos de prueba y su ejecución con JUnit (caja negra) y Eclemma (caja blanca).

Se proporciona la implementación del sistema a probar y además, un fichero con la implementación de varias pruebas para la clase *Cuenta*.

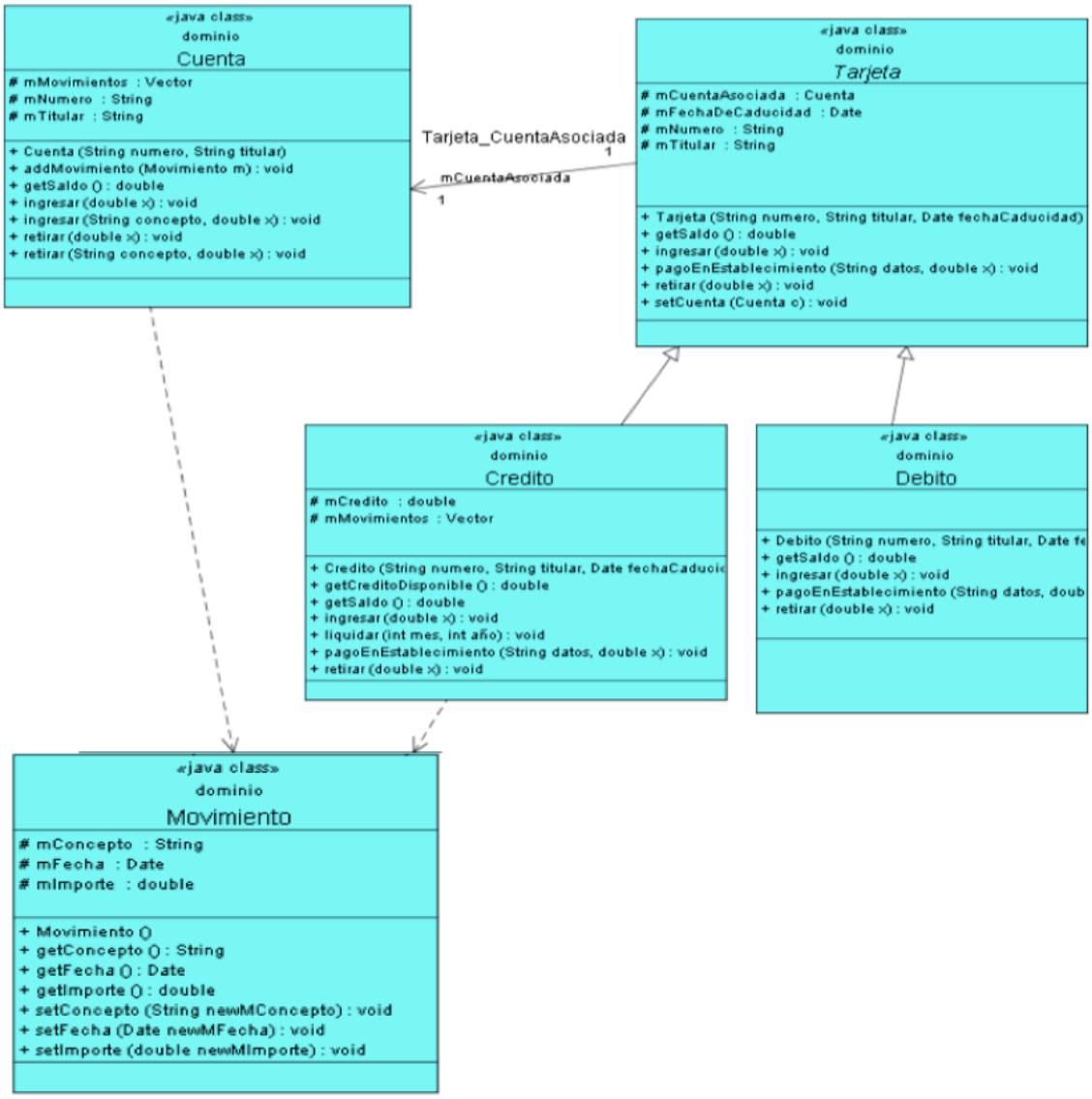
Las tareas a realizar son:

1. Revisar el fichero de pruebas proporcionado para la clase *Cuenta,* para familiarizarse con la sintaxis. Ejecutar las pruebas de la clase *Cuenta* con JUnit.

2. Realizar la implementación de varios casos de prueba para la clase *Crédito* y ejecutarlas con JUnit.

3. Ejecutar con Eclemma las pruebas definidas para las clases *Cuenta* y *Crédito*, y comprobar la cobertura alcanzada.

Práctica 7. Pruebas



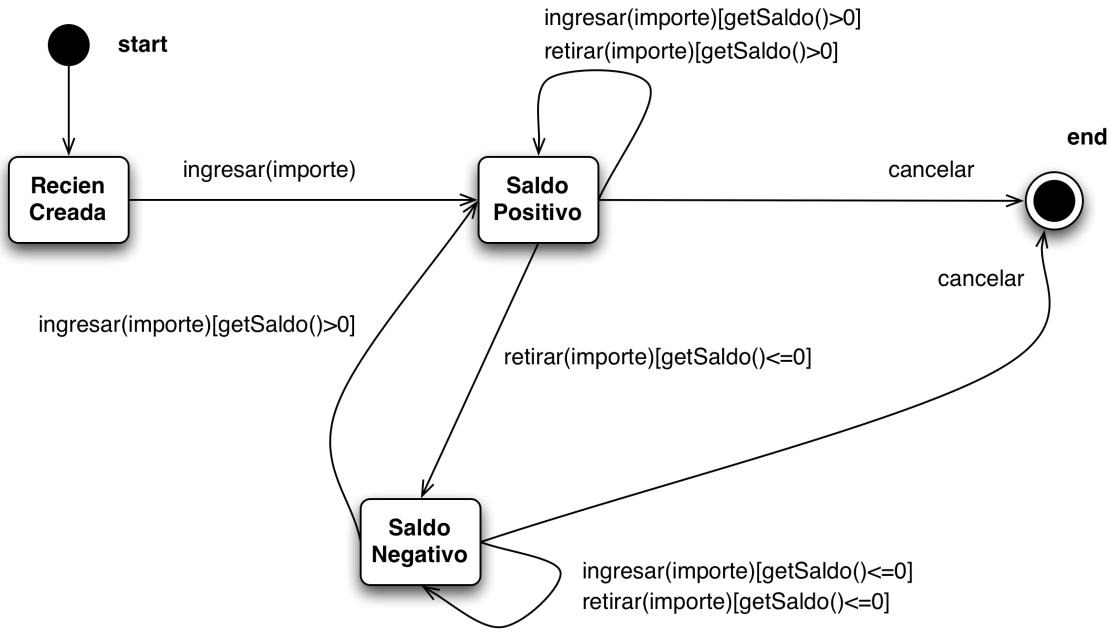
Para conocer más detalles sobre las clases, atributos y métodos, se puede consultar la documentación adjunta con el proyecto. Esto será útil, por ejemplo, para concer las excepciones que lanza cada método.

Práctica 7. Pruebas

Parte Complementaria

Como parte complementaria se pueden desarrollar las siguientes tareas.

1. Dado el siguiente diagrama de estados que representa el comportamiento de una cuenta corriente en nuestro sistema, implemente los casos de prueba necesarios para cumplir los criterios de cobertura de estados y cobertura de transiciones.



2. Detectar un error existente en la clase *Crédito* e intentar resolverlo.

3. Comprobar la cobertura alcanzada actualmente y en caso de ser necesario, crear casos de prueba adicionales para aumentarla.

Entrega

La entrega se realizará mediante una tarea de moodle creada a tal efecto. Se han de entregar los ficheros fuente desarrollados, comprimidos en un fichero zip cuyo nombre esté formado por el número de práctica y nombre del alumno. Ejemplo: p7JuanLopez.zip.
El plazo de entrega finaliza a las 23:55 del Miércoles anterior a la siguiente sesión de prácticas.

Práctica 7. Pruebas

Anexo

A continuación se proporciona documentación que puede servir de ayuda para el desarrollo de esta práctica.

Creación de casos de prueba

El programador utiliza un conjunto clases donde se construyen los casos de prueba y se ejecutan automáticamente.

Se utilizan clases que extienden de TestCase:
- Tienen una parte setUp() que se ejecuta a lo primero y sirve inicializar el objeto que se está probando, hacer conexiones,…
- Una parte TearDown() que se ejecuta después de cada método test, para liberar recursos, memoria, conexiones…
- Una serie de tests que realizan varias operaciones y comprueban el resultado obtenido mediante métodos assert

Método assertxxx() de JUnit	**Qué comprueba**
assertTrue(expresión)	comprueba que expresión evalúe a true
assertFalse(expresión)	comprueba que expresión evalúe a false
assertEquals(esperado,real)	comprueba que esperado sea igual a real
assertNull(objeto)	comprueba que objeto sea null
assertNotNull(objeto)	comprueba que objeto no sea null
assertSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado y objeto_real sean el mismo objeto
assertNotSame(objeto_esperado,objeto_real)	comprueba que objeto_esperado no sea el mismo objeto que objeto_real
fail()	hace que el test termine con fallo

Ejemplo

- Librerías a utilizar, creación de la clase de pruebas y métodos setUp y tearDown

```
 1 package test;
 2 import junit.framework.Test;
 3 import junit.framework.TestCase;
 4 import junit.framework.TestSuite;
 5 import dominio.Cuenta;
 6
 7 public class CuentaTester1 extends TestCase
 8 {
 9     Cuenta cuenta;
10
11     public CuentaTester1(String sTestName)
12     {
13         super(sTestName);
14     }
15
16     public void setUp() throws Exception
17     {
18         cuenta = new Cuenta("0001.0002.12.1234567890", "Fulano de Tal");
19     }
20
21     public void tearDown() throws Exception
22     {
23     }
24
```

Práctica 7. Pruebas

- **Test de una situación válida**, en el que ingresamos una cantidad en la cuenta y probamos que el nuevo saldo es el que debería de ser.
- Para asegurarnos de que no lanza ninguna excepción, ponemos un fail en la parte de código que no se debería de ejecutar y si se ejecutara nos lanzaría un failure al pasar los tests de prueba.

```
25 // Este test ingresa 1000 en la cuenta recien creada ( en setUp() )
26     public void testIngresar1000()
27     {
28         try
29         {
30             cuenta.ingresar(1000);
31         }
32         catch (Exception e)
33         {
34             fail("No deberia lanzar excepcion");
35             // Este caso no deberia lanzar una excepcion asi que ponemos un fail
36             // para estar seguros ya que en caso de lanzarla nos avisaria dando un Failure
37         }
38         assertTrue(cuenta.getSaldo()==1000.0);
39         // Con el assert comprobamos que se ingresa correctamente
40         // y ahora la cuenta tiene un saldo de 1000
41     }
```

- **Test de una situación no válida** en la que intentamos retirar más dinero del que hay en la cuenta.
- El estado de la cuenta no es el estado en el que la deja el test anterior (con saldo de 1000) sino que es el que se definió en el setUp(), por lo que tenemos una cuenta recién creada.
- En este caso, antes de salir del test comprobamos que no se ha hecho nada y la cuenta sigue teniendo saldo 0.
- Además, para asegurarnos de que la operación retirar lanza una excepción, ponemos un fail detrás de la operación retirar que sabemos que no debería de ejecutarse, y si lo hace nos saldría un Failure al pasar los tests.

```
43 // Este test retira 1000 de la cuenta y deberia lanzar una excepcion
44 // En el test anterior ingresamos 1000 pero cada test es independiente
45 // y ahora la cuenta esta recien creada ( en setUp() ) y no tiene saldo
46     public void testRetirar1000()
47     {
48         try
49         {
50             cuenta.retirar(1000);
51             fail("Deberia lanzar excepcion y no llegar aqui");
52             // Como deberia lanzar una excepcion comprobamos que no ejecute esta linea
53             // poniendo un fail que nos avise si lo hace
54         }
55         catch (Exception e)
56         {
57         }
58         assertTrue(cuenta.getSaldo()==0.0);
59         // Antes de salir ponemos un assert para comprobar que la cuenta
60         // se queda con saldo 0, y no -1000 ni cosas asi
61     }
```

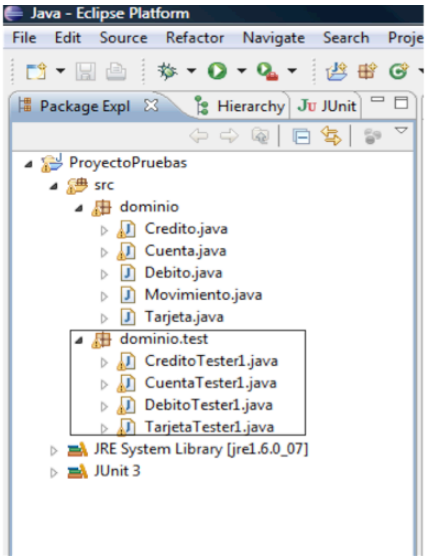
Práctica 7. Pruebas

Ejecución de Pruebas

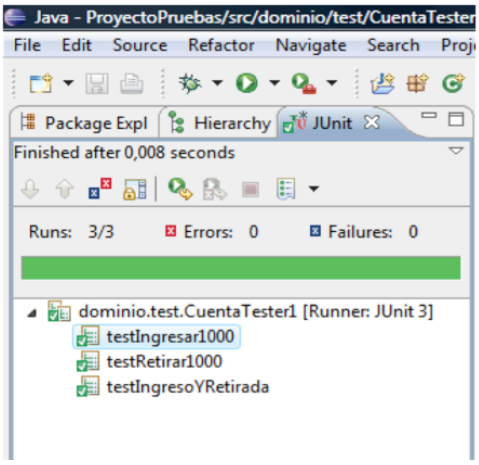
Para ejecutar los casos de prueba realizados hay que hacerlo como JUnit Test.

Al hacerlo se muestra el resultado obtenido en cada caso de prueba incluido en la clase, y este resultado puede ser:

- **Sin errores**: cuando el test (los assert) corresponden a lo que hemos indicado.
- **Error**: cuando el test no corresponde a lo indicado.
- **Fallo**: indica que se ha lanzado un fail. En el ejemplo los hemos usado para comprobar que se lanzan las excepciones adecuadas.
-



- Hay que ejecutar las clases de prueba:
  run as -> JUnit Test



Probando la clase cuenta "cuentaTester1.java"

- Se ejecutan los 3 casos de prueba que se han diseñado, y los 3 han finalizado sin errores

Práctica 7. Pruebas

Para ejecutar las pruebas de caja blanca con Eclemma, hay que pulsar sobre el icono indicado en la figura.

Tras la ejecución podemos observar la cobertura alcanzada y resaltados con distintos colores las partes del código que han sido ejecutadas y las que no. De este modo, para aumentar la cobertura podemos escribir nuevos casos de prueba que fuercen la ejecución de esas partes de código que aún no se han probado.

- Para ejecutar las pruebas de caja blanca
- Las marcas **verdes** indican sentencias ejecutadas
- Las **rojas** las no ejecutadas
- Las **amarillas** las parcialmente ejecutadas
- En la ventana inferior "coverage" se incluye el % de cobertura de sentencias