INGENIERÍA DEL SOFTWARE II

Práctica 1

Pruebas con Junit y Eclemma

Univ. Cantabria – Fac. de Ciencias

Carlos Blanco, Juan Hernández

Objetivos

- Realizar pruebas de caja negra de forma automática
  - Familiarizarse con el framework JUnit
  - Ejecutar pruebas implementadas con JUnit
  - Implementar casos de prueba
- Realizar pruebas de caja blanca usando un plugin complemento de Junit (Eclemma)
  - Familiarizarse con el plugin Eclemma
  - Realizar automáticamente pruebas de caja blanca con distintos criterios de cobertura

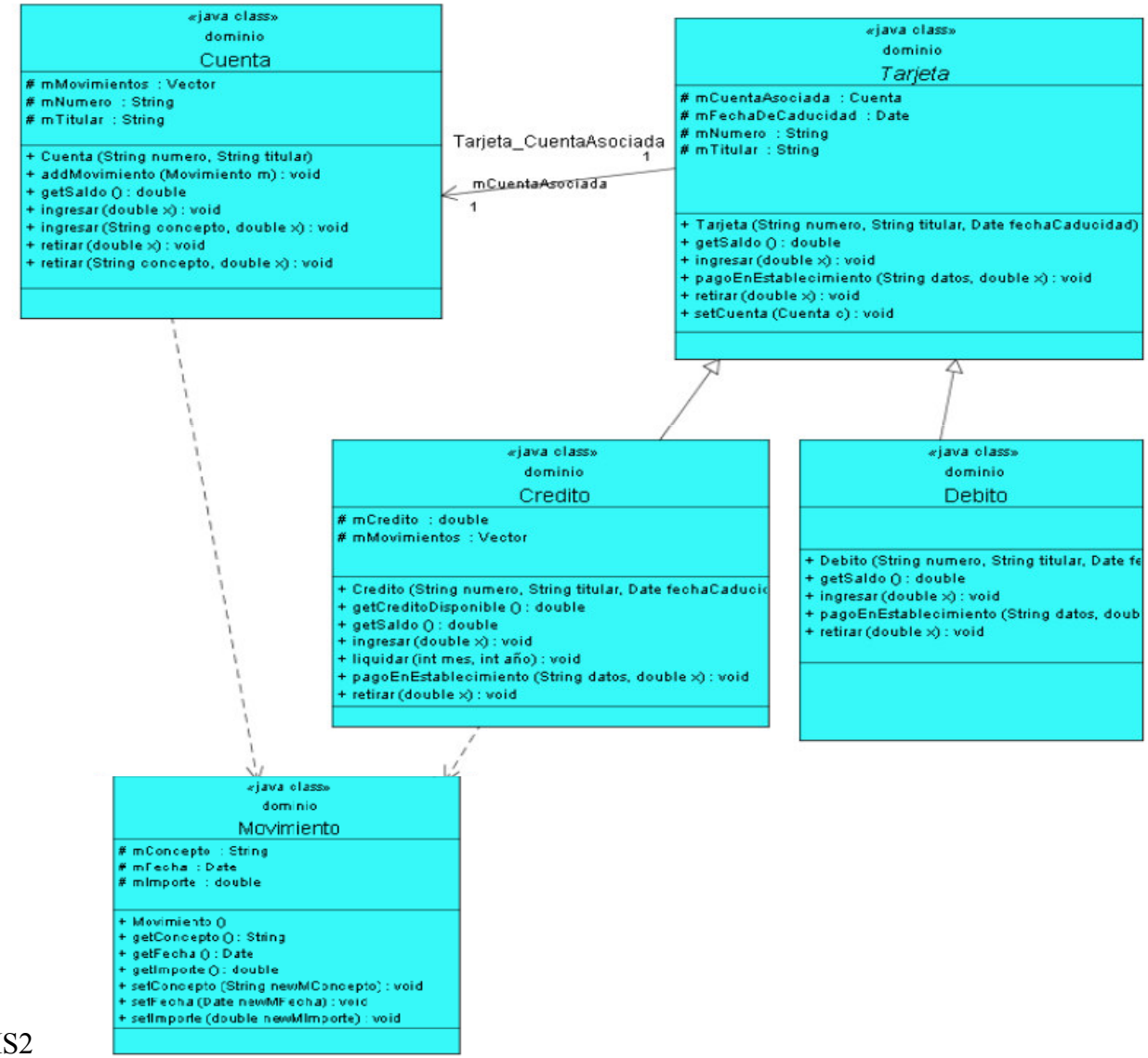
- **Junit** http://www.junit.org
  - Para hacer **pruebas unitarias**
  - Es *open source* y está integrado en la plataforma eclipse

- **Eclemma** http://www.eclemma.org/
  - Actualizar Eclipse:
    - Help -> software update -> find and install -> search for new feature to install
    - Site **http://update.eclemma.org/**

- *Ejemplo de un Sistema Bancario*

- *En nuestro banco hay*
  - *Cuentas y*
  - *Tarjetas (asociadas a una cuenta)*
    - *Tarjetas de Crédito y*
    - *Tarjetas de Débito*
- *Las operaciones que se realizan sobre una Cuenta quedan registradas en un Vector de objetos de clase Movimiento (= con las tarjetas de Crédito)*
- *En Tarjeta, todas las operaciones son abstractas excepto el constructor y setCuenta(Cuenta)"*

Práctica 1 Pruebas
Ejemplo



Ejemplo – Definición de Casos

- 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

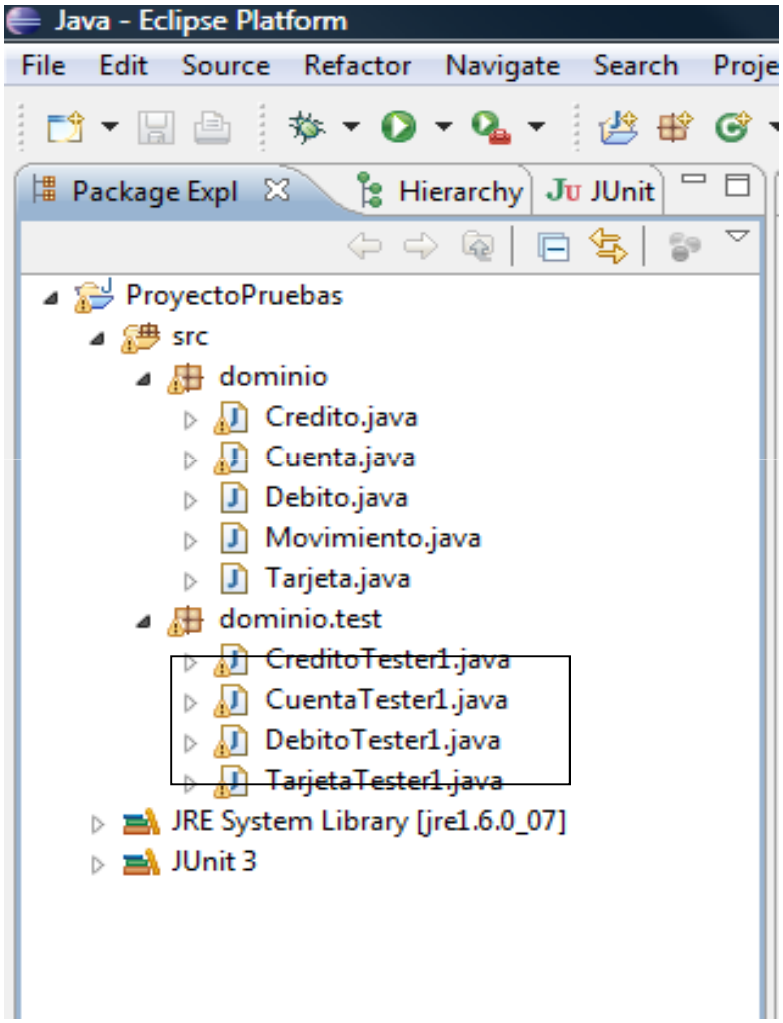
Práctica 1 Pruebas
Ejemplo – Definición de Casos

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
import dominio.Cuenta;

public class CuentaTester1 extends TestCase
{
        Cuenta cuenta;

        public CuentaTester1(String sTestName)
        {
                super(sTestName);
        }

        public void setUp() throws Exception
        {
                cuenta = new Cuenta("0001.0002.12.1234567890", "Fulano de Tal");
        }

         public void tearDown() throws Exception
         {
         }
```

```
public void testIngresar1000()
{
        try {
                cuenta.ingresar(1000);
                assertTrue(cuenta.getSaldo()==1000.0);
        }
        catch (Exception e)
        {
                fail("No debería haber fallado");
        }
}
 public void testRetirar1000()
 {
         try
         {
                 cuenta.retirar(1000);
         }
         catch (Exception e)
         {
         }
         assertTrue(cuenta.getSaldo()==0.0);
```

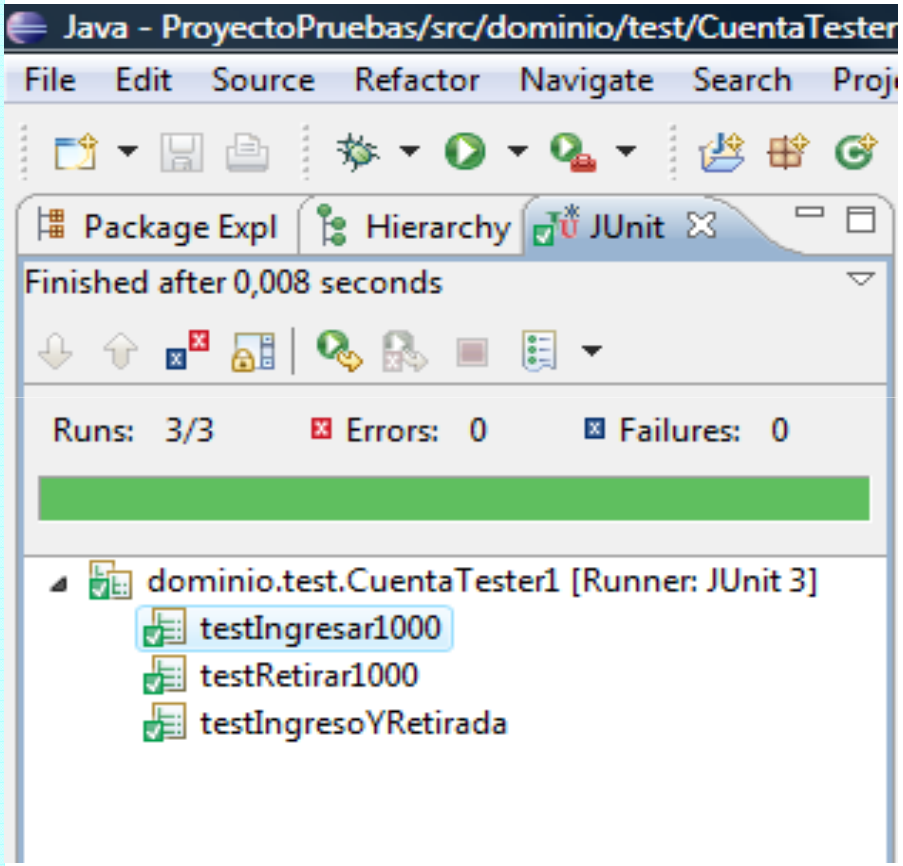
```
        public static void main(String args[])
        {
                junit.swingui.TestRunner.run(CuentaTester1.class);
        }
}
```

Práctica 1 Pruebas
Ejemplo – Ejecución JUnit



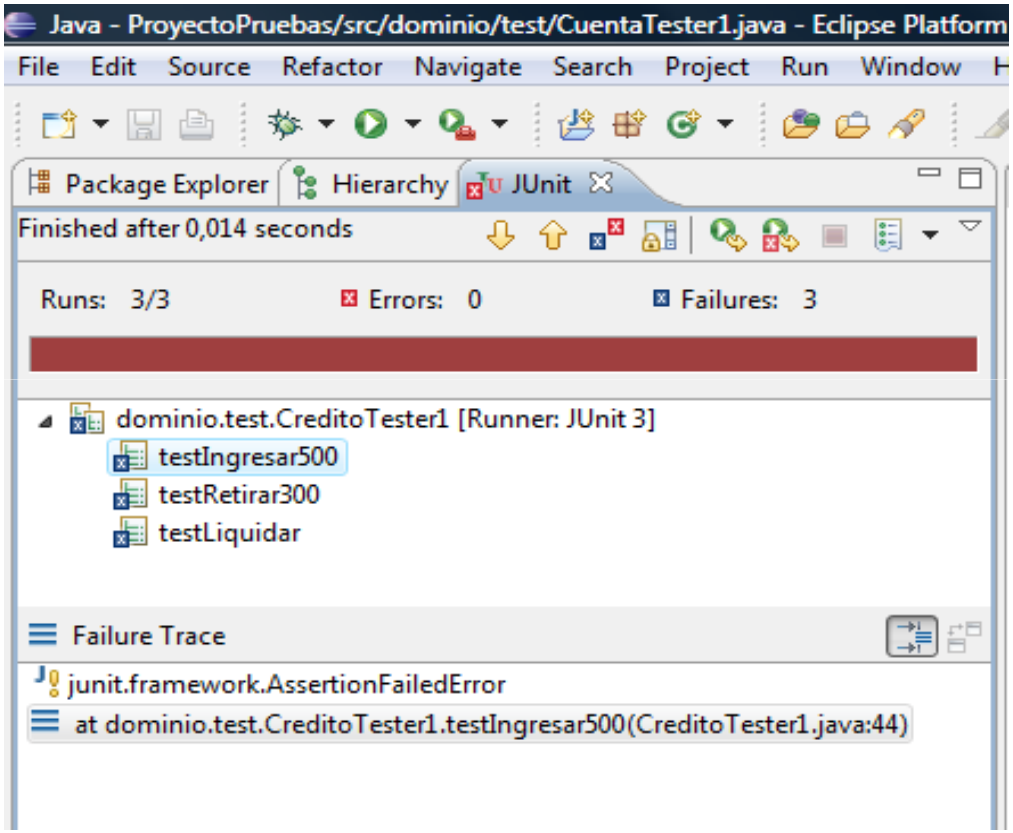
- Hay que ejecutar las clases de prueba:

  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

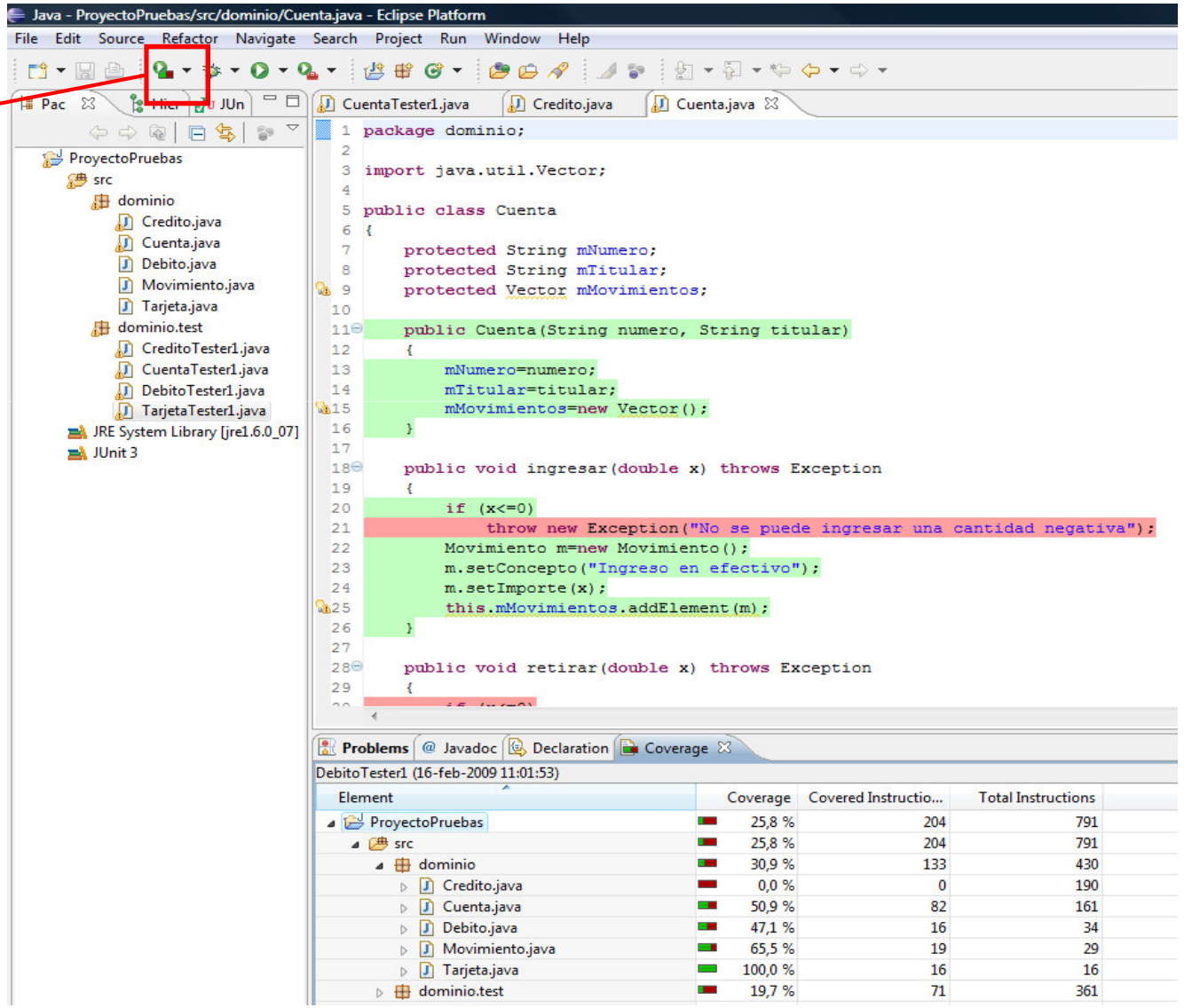


Probando la clase crédito "creditoTester1.java"

- Se ejecutan los 3 casos de prueba y se detectan varios fallos

Práctica 1 Pruebas
Ejemplo – Ejecución Eclemma

- 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



1. Instalación
   - En caso de no haberlo hecho antes, descargar el entorno ECLIPSE (Junit estará incluido) e instalar Eclemma
2. Familiarizarse con las herramientas
   - Utilizar los archivos del ejemplo del sistema bancario
     - Analizar los casos de prueba diseñados viendo cómo se utilizan los setUp, tearDown, assertTrue…
     - Probar cómo se ejecutan las pruebas lanzando Junit y Eclemma
     - Detectar el error encontrado en la clase Crédito y intentar resolverlo
     - Crear un nuevo caso de prueba para las clases del sistema bancario Cuenta y Crédito
     - Crear casos que aumenten la cobertura
3. Implementar una clase "Lista" y realizar las pruebas de caja negra y caja blanca
   - Entregar esta parte por moodle

- Implementar una clase **Lista**
  - La lista incluirá varios elementos ordenados
  - Cada uno ha de proponer su propia clase lista…
  - Ejemplos:
    - Una lista que acepte números de 3 cifras mayores que cero
    - Una lista que incluya cadenas de caracteres y esté ordenada por la longitud de la cadena
    - La lista incluye objetos de tipo *Persona* y está ordenada por la edad
    - La lista sólo incluye números múltiplos de 5, etc…

- Realizar pruebas de **caja negra** y **caja blanca**
  - Diseñar casos de prueba con los valores interesantes elegidos
  - Ejecutarlos con JUnit
  - Si todas las pruebas son correctas, ejecutar Eclemma para comprobar cobertura de sentencias y alcanzar el mayor porcentaje de cobertura
  - En este punto se puede utilizar lo que vimos en teoría:
    - Clases de equivalencia
    - Coberturas 1-wise, 2-wise
    - …

- Para el ejemplo de la lista que incluye números de 3 cifras > 0:
  - Posibles **valores interesantes** {-1,0,999,1000}
  - Para 1-wise, incluir al menos una vez cada valor interesante

```
Public void testAñadir1(){
    lista.añadir(-1);
    assertTrue(lista.getLenght()==0);
    }
```

  - Para 2-wise, incluir un par de valores interesantes en cada caso de prueba

```
Public void testAñadir2(){
    lista.añadir(-1);
    Lista.añadir(999);
    assertTrue(lista.getLenght()==1);
    assertTrue(lista.getElement(0)==999);
    }
```