

Introducción a JUnit

Objetivo: el objetivo de esta práctica es que el participante cree una clase Java con un conjunto de métodos y genere y ejecute un caso de prueba (*TestCase*) para probar los métodos de dicha clase usando JUnit desde el IDE Netbeans.

La clase que se va a construir es una calculadora sencilla, en un principio con las cuatro operaciones básicas: suma, resta, multiplicación y división usando dos argumentos.

Pasos a seguir:

1. Crear la clase Calculadora con las funcionalidades de suma, resta, multiplicación y división usando sólo dos argumentos.

File Edit View Navigate Source Refactor Build Build Noversioning Tools Window Help Projects Projects Projects Projects Files Services Start Page × Velcome to NetBeans Velcome to NetBeans IDE 6.0.1 My NetBeans Welcome to NetBeans IDE 6.0.1 My NetBeans Velcome to NetBeans IDE 6.0.1 My NetBeans News & Ocnecting to a This document demon Connecting to a This document demon Connecting to a No setting Up NetBeand How to set up the Soy JavaK No lintel Mass v I.0.4 and O Nominate NetBeand Developer.com is now nominations for its 20 Awards. Vote to O Using Page Fragments: the ade All NeWS >> A Econtrod Document demon The ade All NeWS >> A Econtrod Document demon Output to Page fragments: the ade All NeWS or reate a web in the ade All NeWS >> A Econtrod Document demon Econtrod Document demon Developer.com is now nominations for its 20 Awards. Vote to Econtrod Document demon Developer.com is now nominations for its 20 Awards. Vote to Econtrod Document demon Developer.com is now nominations for its 20 Awards. Vote to Econtrod Document demon		NetBeans IDE 6.0.1	
Projects Image: Ima	<u>F</u> ile <u>E</u> dit <u>V</u> iew <u>N</u> avigate <u>S</u> ource R	ef <u>a</u> ctor <u>B</u> uild <u>R</u> un Versioning <u>T</u> ools <u>W</u> indow <u>H</u> elp	
Projects Image * Image * Image * <td< th=""><th>1 2 2 5 × 6 0</th><th>9 C T W > B.</th><th></th></td<>	1 2 2 5 × 6 0	9 C T W > B.	
Main.java - Navigator 4LL NEWS >> A	Projects 10 × Files Services	Start Page ×	
Welcome to NetBeans IDE 6.0.1 My NetBeans Recent Projects News & • Connecting to a M This document demor configure the MySQL c on your co • Setting Up NetBea How to set up the Soy JavaFX on Intel Macs v 10,4 and Main.java - Navigator • X <no project="" recent=""> • Nominate NetBea Developer.com is now nominations for its 20 Awards. Vote to <no available="" view=""> • Using Page Fragm How to create a web it two page fragments: the ade</no></no>		The second s	
Main.java - Navigator • X <no project="" recent=""> • Setting Up NetBea How to set up the Soy JavaFX on Intel Mass v 10.4 and <no project="" recent=""> • Nominate NetBea Developer, com is now nominations for its 20 Awards. Vote to <no project="" recent=""> • Using Page Fragn How to create a web it two page fragments: it heade ALL NEWS >> A</no></no></no>		Welcome to NetBeans IDE 6.0.1	My NetBeans
Main.java - Navigator III × <no project="" recent=""> Setting Up NetBea How to set up the Soy JavaEX on Intel Macs v 10.4 and <no available="" view=""> IIII > <no available="" view=""> ALL NEWS >> A</no></no></no>		Recent Projects	s News &
ALL NEWS >> A	Main.java - Navigator 🐠 🗙	<no project="" recent=""></no>	Connecting to a M This document demor configure the MySQL c on your co Setting Up NetBea How to set up the Soy JavaFX on Intel Macs v 10.4 and Nominate NetBear Developer.com is now nominations for its 20 Awards. Vote to Using Page Fragn How to create a web i two page fragments: t
		Eastured Demo	ALL NEWS >> A

a. Abrir Netbeans

- b. Crear un nuevo proyecto de tipo *Java Application*. Para esto se debe seleccionar en el menú File | New Project....
 - En la ventana emergente *New Project* se debe seleccionar la categoría *Java* y dentro de esta categoría seleccionar el tipo de proyecto "*Java Application*".
 - Pulsar el botón "Next>".
 - En la ventana emergente **New Java Application** colocar en el campo **Project Name** el valor *Calculadora* y asegurarse de que estén seleccionadas las dos opciones de la ventana.
 - Pulsar el botón "*Finish*". Entonces aparecerá el proyecto Calculadora con sus carpetas asociadas, el cual puede verse en la ventana Projects
- c. Crear la clase Calculator.
 - En el proyecto *calculadora*, en la carpeta *Source Packages* seleccionar el paquete *calculadora*.
 - Con el botón derecho del ratón y seleccionar: New | Java Class....

		Calculadora - NetBeans IDE 6.0.1	_ O X
<u>F</u> ile <u>E</u> dit ⊻iew	<u>N</u> avigate <u>S</u> ource	Ref <u>a</u> ctor <u>B</u> uild <u>R</u> un Vers <u>i</u> oning <u>T</u> ools <u>W</u> indow <u>H</u> elp	
* * *	1 🔀 🔁 🖻	崎 🎑 🔄 🔄 🖓 👘	
Projects 🛯 🗙 Files	Services	Start Page × 🔊 Main.java ×	
✓ <a>Source Pace	kages		
	New	• 🛃 java Class ate, choose Tools T	emplates 🔸
D 📑 Test Pack	<u>F</u> ind (trl+F 🔲 Java Package ce in the editor.	
▷ G Libraries▷ G Test Libra	Cu <u>t</u> (<u>C</u> opy (Paste	trl+X	
	<u>D</u> elete : Refactor	uprimir POther	
Main.java - Naviga	Compile Package - I Local History	* @param args the command line argument	5
▼ 🚷 Main	Tools Jtj args/	<pre></pre>	
🏰 🗆 🐼 🖨 🌄	◆ <u>≡</u>	1:1 INS	

- En la ventana emergente *New Java Class*, escribir Calculator en el campo *Class Name* el valor y pulsar el botón "*Finish*".

0	P	New Java Class	×
Steps	Name and I	Location	
 Choose File Type Name and Location 	Class <u>N</u> ame:	Calculator	
	<u>P</u> roject:	Calculadora	
A	Location:	Source Packages	•
	Pac <u>k</u> age:	calculadora	
	<u>C</u> reated File:	ebas/NetbeansJUnit/Calculadora/src/calculadora/Calculat	»r
		< Back Next > Finish Cancel Hel;	2

- En la ventana emergente New Java Class, escribir Calculator en el campo Class Name el valor y pulsar el botón "Finish".
- Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase Calculator.
- d. Escribir los métodos de la clase Calculator.
 - En la pestaña del código de la clase Calculator colocar entre los paréntesis delimitadores del código de la clase los métodos de la calculadora. Una vez escritos el código de la clase Calculator debe verse de la siguiente forma:

```
public class Calculator {
   public double add(double number1, double number2){
      return number1 + number2;
   }
   public double subtract(double number1, double number2){
      return number1 - number2;
   }
   public double multiply(double number1, double number2) {
      return number1 * number2;
   }
   public double divide(double number1, double number2){
      return number1 / number2;
   }
}
```

PRÁCTICAS DEL CURSO PRUEBA DE SOFTWARE



- Pulsar <Crtl+S> para salvar el trabajo realizado hasta ahora.
- Construir el proyecto para verificar que no tiene errores seleccionando en el menú: Build | Build Main Project.
- En la ventana Output se podrá ver el resultado de la acción Build, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana Output, corregirlos y volver a construir el proyecto.

2. Crear un caso de prueba usando JUnit para la clase Calculator que pruebe cada uno de sus métodos

- a. En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba, que en este caso es la clase Calculator.java.
 - En la barra de manú: Tools | Create JUnit Test.
 - Aparece una ventana emergente para seleccionar la versión de JUnit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x y pulsar el botón Select.

- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de pPrueba. Por ahora se van a desmarcar las opciones *Test Initializer* y *Test Finalizer*. Luego se debe pulsar el botón *OK*.

	Cre	ate Tests	X)
<u>C</u> lass to Test:	calculadora.	Calculator	
Class <u>N</u> ame:	calculadora	.CalculatorTest)
Location:	Test Packaç	ges 🗸 🗸	
Code Gene	ation		
Method Ac	cess Levels	Generated Code	
🗹 <u>P</u> ubli	с	🔲 Te <u>s</u> t Initializer	
 ✓ Pr <u>o</u> te	ected	🔲 Test Finalizer	
🗹 Pac <u>k</u>	age Private	🗹 Default <u>M</u> ethod Bodies	
		Generated Comments	
		🗹 Javadoc Comments	
		☑ Source Code H <u>i</u> nts	
L		ок Cancel <u>H</u> elp	

b. Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase del *framework* JUnit y que tiene cuatro métodos de prueba, testAdd(), testSubtract(), testMultiply() y testDivide(), para probar cada uno de los métodos de la clase Calculator usando el método assertEquals().

PRÁCTICAS DEL CURSO PRUEBA DE SOFTWARE



- c. Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase del framework JUnit y que tiene cuatro métodos de prueba, testAdd(), testSubtract(), testMultiply() y testDivide(), para probar cada uno de los métodos de la clase Calculator usando el método assertEquals().
 - Para que cada uno de los métodos de prueba compile correctamente, se debe borrar de cada uno las dos últimas líneas de código, es decir, se debe borrar de cada método de prueba el siguiente código:

```
// TODO review the generated test code and remove the
default call to fail.
fail("The test case is a prototype.");
```

- Pulsar <Crtl+S> para salvar el trabajo realizado hasta ahora.
- Se compila la clase de prueba generada seleccionando en el menú Build | Compile "CalculatorTest.java".

- En la ventana Output se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.
- El código del caso de prueba CalculatorTest debería ser el siguiente:

```
package calculadora;
import junit.framework.TestCase;
/**
 * @author Administrador
 * /
public class CalculatorTest extends TestCase {
    public CalculatorTest(String testName) {
       super(testName);
    }
    /**
     * Test of add method, of class Calculator.
     * /
    public void testAdd() {
        System.out.println("add");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculator instance = new Calculator();
        double expResult = 0.0;
        double result = instance.add(number1, number2);
       assertEquals(expResult, result);
    }
    /**
     * Test of subtract method, of class Calculator.
     */
    public void testSubtract() {
        System.out.println("subtract");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculator instance = new Calculator();
        double expResult = 0.0;
        double result = instance.subtract(number1, number2);
        assertEquals(expResult, result);
    }
    /**
     * Test of multiply method, of class Calculator.
     */
    public void testMultiply() {
        System.out.println("multiply");
        double number1 = 0.0;
        double number2 = 0.0;
        Calculator instance = new Calculator();
        double expResult = 0.0;
        double result = instance.multiply(number1, number2);
```

```
assertEquals(expResult, result);
}
/**
 * Test of divide method, of class Calculator.
 */
public void testDivide() {
   System.out.println("divide");
   double number1 = 0.0;
   double number2 = 0.0;
   Calculator instance = new Calculator();
   double expResult = 0.0;
   double result = instance.divide(number1, number2);
   assertEquals(expResult, result);
}
```

- 3. Ejecutar el Caso de Prueba para comprobar las funcionalidades de la clase Calculadora.
 - a. Para ejecutar el caso de prueba primero se deben escribir los valores de los parámetros de cada uno de los métodos de prueba, es decir, definir cuales son los valores de los argumentos que se van a probar en cada una de las operaciones de la calculadora y cual es el valor del resultado esperado. Esto se puede hacer directamente en el código de cada método de prueba colocando los valores a operar en los parámetros number1 y number2 y el valor esperado de la operación a probar con esos argumentos en la variable expResult. Se guarda el trabajo con <Ctrl+S>.
 - b. Se ejecuta el caso de prueba desde el menú seleccionando: Run | Test "Calcualdora".

PRÁCTICAS DEL CURSO PRUEBA DE SOFTWARE



Práctica

Creación de un caso de prueba usando los métodos setUp() y tearDown() de la clase TestCase de JUnit

Objetivo: crear una clase Java y definir y ejecutar un caso de prueba para las funcionalidades de dicha clase usando los métodos setUp() y tearDown() de la clase TestCase de JUnit desde *Netbeans*.

La clase que se va a construir es una clase para representar y operar números complejos. Un número complejo es un par ordenado de números reales (a, b), en donde a se conoce como parte real y b como parte imaginaria del número complejo. Los números complejos tienen las siguientes operaciones:

- Suma: (a,b)+(c,d)=(a+c)+(b+d)i
- Multiplicación: $(a,b) \cdot (c,d) = (ac-bd) + (ad+cb)i$
- Igualdad: $(a,b) = (c,d) \Rightarrow a = c \land b = d$

La clase para representar los números complejos y sus operaciones se muestra a continuación:

6	ComplexNumber
٥	double Real
0	double Imaginary
۲	getReal ()
	getImaginary ()
	add ()
0	multiply ()
0	equals ()

- 1. Crear la clase ComplexNumber con sus atributos y las funcionalidades:
 - Obtener la parte real del número complejo, getReal()
 - Obtener la parte imaginaria del número complejo, getImaginary()
 - Sumar números complejos, add()
 - Multiplicar números complejos, multiply()
 - Verificar igualdad de números complejos, equals()
 - a. Abrir Netbeans.
 - b. Crear un nuevo proyecto de tipo *Java Application*. Para esto se debe seleccionar en el menú: File | New Project ...
 - En la ventana Emergente New Project se debe seleccionar la categoría Java y dentro de esta categoría seleccionar el tipo de proyecto "Java Application". Pulsar el botón "Next>".
 - En la ventana emergente New Java Application colocar en el campo Project Name el valor ComplexNumber y asegurar que estén seleccionadas las dos opciones de la ventana. Pulsar el botón "Finish".
 - Entonces aparecerá el proyecto **ComplexNumber** con sus carpetas asociadas, el cual puede verse en la ventana *Projects*.
 - c. Crear la clase ComplexNumber
 - En el Proyecto *ComplexNumber*, en la carpeta *Source Packages* seleccionar el paquete **complexnumber**.
 - Pulsar el botón derecho del ratón y seleccionar: New | Java Class...
 - En la ventana emergente *New Java Class*, colocar en el campo *Class Name* el valor ComplexNumber y pulsar "*Finish*".
 - Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase ComplexNumber, como se muestra a continuación:



- d. Escribir los métodos de la clase ComplexNumber
 - En la pestaña del código de la clase ComplexNumber escribir entre los paréntesis delimitadores del código de la clase los atributos y los métodos de la ComplexNumber. Una vez escritos el código de la clase debe verse de la siguiente forma:

```
package complexnumber;
public class ComplexNumber {
  private double Real;
 private double Imaginary;
 public ComplexNumber(double re, double imag) {
   Real = re;
    Imaginary = imag;
  }
 public double getReal() {
    return Real;
 public double getImaginary() {
    return Imaginary;
  }
  public ComplexNumber add(ComplexNumber c) {
    return new ComplexNumber(getReal() + c.getReal(),
                              getImaginary() + c.getImaginary());
  }
  public ComplexNumber multiply(ComplexNumber c) {
    double re = getReal()*c.getReal() - getImaginary()*c.getImaginary();
    double imag = getImaginary()*c.getReal() +
    getReal()*c.getImaginary();
    return new ComplexNumber(re, imag);
  }
 @Override
 public boolean equals(Object anObject) {
    if (anObject instanceof ComplexNumber) {
        ComplexNumber c = (ComplexNumber) anObject;
     return ((c.getReal() == getReal()) && (c.getImaginary() ==
getImaginary()));
   } else
     return false;
  }
```

- Pulsar <Ctrl+S> para salvar el trabajo realizado hasta ahora.
- Construir el proyecto para verificar que no tiene errores. Esto se hace seleccionando en el menú: Build |Build Main Project
- En la ventana *Output* se podrá ver el resultado de la acción *Build*, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana *Output*, corregirlos y volver a construir el proyecto.

2. Crear un caso de prueba usando JUnit para la clase ComplexNumber que pruebe cada uno de sus métodos, usando el método setUp() para crear los objetos y estructuras de datos necesarias para establecer el contexto de los métodos del caso de prueba y el método tearDown() para liberar los recursos reservados para realizar la ejecución del caso de prueba.

Para esto hay que realizar los siguientes pasos:

- a. En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba, que en este caso es la clase ComplexNumber.java.
 - En la barra de menú seleccionar: Tools | Create JUnit Tests.
 - Aparece una ventana emergente para seleccionar la versión de JUnit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x y pulsar el botón Select.
 - Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba.
 Verificar que estén marcadas todas las opciones pulsar el botón *OK*.
- b. Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase de JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase TestCase de JUnit, y el esqueleto de código para los métodos de prueba de la clase ComplexNumber: testgetReal, testgetImaginary(), testAdd(), testMultiply() y testEquals(), para probar cada uno de los métodos de la clase ComplexNumber usando el método AssertEquals().
 - Se deben declarar como atributos privados los elementos que se van a usar en el método setUp() y hacer las inicializaciones correspondientes en el método. En este caso se van a utilizar tres instancias de números complejos que se declaran como atributos privados de la clase ComplexNumberTest y que se crearán e inicializarán en el cuerpo del método setUp(). El código asociado a los atributos privados y al método setUp() es el siguiente:

```
public class ComplexNumberTest extends TestCase {
    private ComplexNumber cOneZero;
    private ComplexNumber cZeroOne;
    private ComplexNumber cOneOne;

    protected void setUp() throws Exception {
        cOneZero = new ComplexNumber (1,0);
        cZeroOne = new ComplexNumber (0,1);
    }
}
```

	cOneOne =	new	ComplexNumber	(1,1);
}				

- En los métodos de prueba testGetReal() y testGetImaginary() se elimina la instrucción:

```
ComplexNumber instance = null;
```

- En los métodos de prueba testAdd(), testMultiply() se eliminan las instrucciones:

```
ComplexNumber c = null;
ComplexNumber instance = null;
ComplexNumber expResult = null;
```

- En el método de prueba testEquals() se eliminan las instrucciones:

```
Object anObject = null;
ComplexNumber instance = null;
```

- Y se sustituyen los números complejos instance, c, expResult y anObject por los números complejo del conjunto números complejos creados en el método setUp() que se vaya a usar en cada una de las pruebas.
- Adicionalmente, para que cada uno de los métodos de prueba compile correctamente, se debe borrar de cada uno las dos últimas líneas de código, es decir, se debe borrar de cada método de prueba el siguiente código:

```
// TODO review the generated test code and remove the
default call to fail.
fail("The test case is a prototype.");
```

 El método tearDown(), que se encarga de liberar los recursos reservados en el métodos setUp(), colocará en null a los apuntadores de los objetos ComplexNumber creados en el método setUp(). Para esto se sustituirá en el esqueleto del método tearDown() la instrucción

```
super.tearDown();
```

por las instrucciones:

```
cOneZero = null;
cZeroOne = null;
cOneOne = null;
```

- Pulsar <Ctrl+S> para salvar el trabajo.

- c. Se compila la clase de prueba generada seleccionando: Build | Compile "ComplexNumberTest.java".
 - En la ventana Output se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo, se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente. El código del caso de prueba ComplexNumberTest debería ser el siguiente:

```
package complexnumber;
import junit.framework.TestCase;
public class ComplexNumberTest extends TestCase {
    private ComplexNumber cOneZero;
    private ComplexNumber cZeroOne;
   private ComplexNumber cOneOne;
    public ComplexNumberTest(String testName) {
        super(testName);
    }
    @Override
    protected void setUp() throws Exception {
       cOneZero = new ComplexNumber(1, 0);
        cZeroOne = new ComplexNumber(0, 1);
        cOneOne = new ComplexNumber(1, 1);
    }
    @Override
    protected void tearDown() throws Exception {
       cOneZero = null;
       cZeroOne = null;
       cOneOne = null;
    }
    /**
     * Test of getReal method, of class ComplexNumber.
     * /
    public void testGetReal() {
        System.out.println("getReal");
        double expResult = 0.0;
       double result = cZeroOne.getReal();
       assertEquals(expResult, result);
    }
    /**
    * Test of getImaginary method, of class ComplexNumber.
     */
    public void testGetImaginary() {
        System.out.println("getImaginary");
        double expResult = 0.0;
        double result = cOneZero.getImaginary();
        assertEquals(expResult, result);
    }
    /**
     * Test of add method, of class ComplexNumber.
     */
    public void testAdd() {
        System.out.println("multiply");
        ComplexNumber result = cZeroOne.add(cOneZero);
        assertEquals(cOneOne, result);
```

```
}
/**
* Test of multiply method, of class ComplexNumber.
*/
public void testMultiply() {
   System.out.println("multiply");
   ComplexNumber result = cZeroOne.multiply(cOneZero);
   assertEquals(cZeroOne, result);
}
/**
 * Test of equals method, of class ComplexNumber.
 */
public void testEquals() {
    System.out.println("equals");
   boolean expResult = false;
   boolean result = cZeroOne.equals(cOneZero);
   assertEquals(expResult, result);
}
```

- 3. Ejecutar el caso de prueba para comprobar la funcionalidad de la clase ComplexNumber
 - Se ejecuta el caso de prueba seleccionando: Run | Test "ComplexNumber".
 - En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación

JUnit 1	est Results	• x							
💎 AI	P All 5 tests passed. get Real								
	- 🏠 complexnumber.ComplexNumberTest passed	getImaginary							
	••• • testGetReal passed (0,016 s)	multiply							
with the second seco									
	- • testAdd passed (0,0 s) equals								
	••• • testMultiply passed (0,0 s)								
	• testEquals passed (0,0 s)								
Out	put - Complex_Number (test)								
N	indt -								
1º									
101	Compliant L course file to C.\Doguments and Sattings\}ddministrader\Wis degumentse\We	tRoonsDrojosts)Corpley Number)build)slosses							
1-	compiling i source file co c. (pocuments and becomys(auministrator(his documentos(ke	cheansflojeccs(complex Number(bulld(classes							
	Compile.	- Deservation - Constant Markey had to be and all and							
	Compling I source file to U:\Documents and Settings\Administrador\Mis documentos\NetBeansProjects\Complex Number\build\test\classes								
	Comprise-Cest.								
	Testsuite. Complexitumber.complexitumberlest								
	getkel								
	get magnary								
	multiply								
	multiply								
	equals								
	Tests run: 5, Failures: 0, Errors: 0, Time elapsed: 0,062 sec								
	an and a second								
	getkeal								
	getImaginary								
	multiply								
	muttibià								
	equals								
	test-report:								
	test:								
	BUILD SUCCESSFUL (total time: 0 seconds)								

Referencias Bibliográficas

- Fontoura, M., Pree W., Rumpe B. *The UML Profile for Framework Architectures*. Capítulo 6. Addison-Wesley. 2002

Práctica 3

Creación de casos de prueba y suites de pruebas usando JUnit

Objetivo: crear y ejecutar casos y suites de pruebas usando JUnit desde *Netbeans*, para probar las funcionalidades de las clases de un carrito de compras usado en las aplicaciones de comercio electrónico

La funcionalidad del carrito de compras a trabajar es un ejemplo sencillo y va a estar constituido por dos clases: Product y ShoppingCart, cuya estructura se muestra a continuación:



La clase Product representa los productos que se pueden comprar añadiendo al carrito de compras y la clase ShoppingCart representa al carrito de compras donde se pueden introducir productos a comprar como una lista de ítems.

Después de crear las clases a trabajar en el ambiente Netbeans 6.0 se debe:

- Crear y ejecutar los casos de prueba para las funcionalidades de la clase Product y de la Clase ShoppingCart, utilizando los métodos setUp() y tearDown() para evitar la duplicación de código y poder reusar los fixtures en cada caso de prueba
- Crear y ejecutar una suite de pruebas que permita ejecutar secuencialmente las pruebas de todas las funcionalidades de la aplicación del carrito de compras

1. Crear un nuevo proyecto de tipo Java Application llamado Shopping

Para esto se debe:

- Seleccionar en el menú: File | New Project ...
- En la ventana Emergente *New Project* se debe seleccionar la categoría Java y dentro de esta categoría seleccionar el tipo de proyecto "Java Application". Pulsar el botón "*Next>*"
- En la ventana emergente *New Java Application* escribir en el campo Project Name el valor Shopping y asegurar que estén seleccionadas las dos opciones de la ventana y pulsar el botón "*Finish*".
- Entonces aparecerá el proyecto con sus carpetas asociadas, el cual puede verse en la ventana *Projects*, que se muestra a continuación:



2. Crear la clase Product con sus atributos y servicios:

Atributos:

- code: String
- title: String
- description: String
- price: double

Servicios:

- Constructor de productos con la siguiente signatura:
 - Product(String code, String title, String description, double price)
- Obtener el código de un producto, getCode()
- Obtener el nombre de un producto, getTitle()
- Obtener la descripción de un producto, getDescription()
- Obtener el precio de un producto, getPrice()

Para esto hay que realizar los siguientes pasos:

b. Crear la clase Product

- En el Proyecto Shopping en la carpeta Source Packages seleccionar el paquete shopping. Pulsar el botón derecho del ratón y seleccionar New | Java Class...
- En la ventana emergente New Java Class, escribir en el campo Class Name el valor Product y pulsar el botón "Finish".
- Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase Product.
- c. Definir los atributos y métodos de la clase Product.
 - En la pestaña del código de la clase Product colocar entre los paréntesis delimitadores del código de la clase los atributos y los métodos de la clase Product. Una vez escritos el código de la clase debe verse de la siguiente forma:

```
this.code = code;
       this.title = title;
        this.description = description;
     this.price = price;
    }
    public String getCode() {
     return code;
    }
    public String getTitle() {
     return title;
    }
   public String getDescription() {
     return description;
    }
   public double getPrice() {
     return price;
    }
}
```

- Pulsar <Crtl+S> para salvar el trabajo realizado hasta ahora.
- Construir el proyecto para verificar que no tiene errores. Esto se hace seleccionando en el menú: Build | Build Main Project
- En la ventana *Output* se podrá ver el resultado de la acción *Build*, la cual debería mostrar que se ha compilado exitosamente. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana *Output*, corregirlos y volver a construir el proyecto.

- 3. Crear la clase ShoopingCart con sus atributos y servicios: Atributos:
 - items: ArrayList

Servicios:

- Constructor de ShoppingCart donde se crea un ArrayList de items
- Obtener el monto total de la compra, getBalance()
- Añadir un item de producto al carrito, addItem()
- Quitar un item de producto del carrito, removeItem()
- Obtener la cantidad de ítems de productos que hay en el carrito, getItemCount()
- Vaciar el carrito, empty()

Para ello hay que realizar los siguientes pasos:

- a. Crear la clase ShoppingCart.
 - En el proyecto *shopping* en la carpeta *Source Packages* seleccionar el paquete shopping. Pulsar el botón derecho del ratón y seleccionar en el menú de contexto: New | Java Class....
 - En la ventana emergente "*New Java Class*", colocar en el campo "*Class Name*" el valor ShoppingCart y pulsar "*Finish*".
 - Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase ShoppingCart

b. Definir los atributos y métodos de la clase ShoppingCart

- En la pestaña del código de la clase ShoppingCart colocar entre los paréntesis delimitadores del código de la clase los atributos y los métodos de la ShoppingCart. Una vez escritos el código de la clase debe verse de la siguiente forma:

```
package shopping;
import java.util.*;
public class ShoopingCart {
    private ArrayList items;
    public ShoppingCart() {
        items = new ArrayList();
      }
    public double getBalance() {
        double balance = 0.00;
        for (Iterator i = items.iterator(); i.hasNext();) {
            Product item = (Product)i.next();
            balance += item.getPrice();
        }
    }
}
```

```
return balance;
}
public void addItem(Product item) {
    items.add(item);
}
public void removeItem(Product item) {
    items.remove(item);
}
public int getItemCount() {
    return items.size();
}
public void empty() {
    items.clear();
}
```

- Pulsar <Crtl+S> para salvar el trabajo realizado hasta ahora
- Construir el proyecto para verificar que no tiene errores. Esto se hace seleccionando Build | Build Main Project.
- En la ventana *Output* se podrá ver el resultado de la acción *Build*, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana Output, corregirlos y volver a construir el proyecto.
- 4. Crear un caso de prueba usando JUnit para la clase Product que pruebe cada uno de sus métodos, usando el método setUp() para crear los objetos y estructuras de datos necesarias para establecer el contexto de los métodos del caso de prueba y el método tearDown() para liberar los recursos reservados para realizar la ejecución del caso de prueba.

Para ello hay que realizar los siguientes pasos:

- En la ventana Projects se selecciona la clase para la que se quiere construir el caso de prueba o TestCase, que en este caso es la clase Product.
- En la barra del menú seleccionar Tools | Create Junit Tests. Aparece una ventana emergente para seleccionar la versión de JUnit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x y el botón *Select*.
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba. Verificar que estén marcadas todas las opciones y pulsar el botón "*OK*".

- Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase de JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase testCase del framework JUnit, y el esqueleto de código para los métodos de prueba de la clase Product, para probar cada uno de los métodos de la clase usando el método assertEquals().
- Se deben declarar como atributos privados los elementos que se van a usar en el método setUp() y hacer las inicializaciones correspondientes en el método. En este caso se va a utilizar una instancia de la clase Product que se declara como atributo privados de la clase testProduct y que se creará e inicializará en el cuerpo del método setUp(). El código asociado a los atributos privados y al método setUp() es el siguiente:

```
public class ProductTest extends TestCase {
    private Product prod1;

protected void setUp() throws Exception {
    prod1 = new Product("P001", "Plasma TV LG 32",
        "Plasma TV with TDT Decod. and high resolution Screen",
        699.99);
}
```

- En los métodos de prueba se eliminan la instrucción:

```
Product instance = null;
```

Y en el código de cada método de prueba se sustituye instance por el producto prod1 creado en el método setUp() y se escriben los valores esperados de acuerdo a los valores de los atributos del producto prod1.

 Adicionalmente, para que cada uno de los métodos de prueba compile correctamente, se debe borrar de cada uno las dos últimas líneas de código, es decir, se debe borrar de cada método de prueba el siguiente código:

```
// TODO review the generated test code and remove
    the default call to fail.
fail("The test case is a prototype.");
```

 El método tearDown(), que se encarga de liberar los recursos reservados en el métodos setUp(), colocará en null a los apuntadores de los objetos creados en el método SetUp(). Para esto se sustituirá en el esqueleto del método tearDown() la instrucción:

```
super.tearDown();
```

por las instrucciones:

prod1 = null;

- Grabar el trabajo con < Crtl+S>
- Se compila la clase de prueba con Build | "ProductTest.java".
- En la ventana Output se ve el resultado de la compilación, el cual debería mostrar que se ha compilado exitosamente. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente. El código del caso de prueba ProductTest debería ser el siguiente:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package shopping;
import junit.framework.TestCase;
/**
 * @author Administrador
 */
public class ProductTest extends TestCase {
   private Product prod1;
    public ProductTest(String testName) {
        super(testName);
    }
    @Override
    protected void setUp() throws Exception {
     prod1 = new Product("P001", "Plasma TV LG 32", "Plasma TV with
TDT Decod. and high resolution Screen", 699.99);
    }
    @Override
    protected void tearDown() throws Exception {
       prod1 = null;
    }
    /**
     * Test of getCode method, of class Product.
     */
    public void testGetCode() {
        System.out.println("getCode");
        String expResult = "P001";
        String result = prod1.getCode();
        assertEquals(expResult, result);
    }
    / * *
     * Test of getTitle method, of class Product.
     * /
    public void testGetTitle() {
```

```
System.out.println("getTitle");
        String expResult = "Plasma TV LG 32";
        String result = prod1.getTitle();
        assertEquals(expResult, result);
    }
    /**
     * Test of getDescription method, of class Product.
     * /
    public void testGetDescription() {
        System.out.println("getDescription");
        String expResult = "Plasma TV with TDT Decod. and high
resolution Screen";
        String result = prod1.getDescription();
        assertEquals(expResult, result);
    }
    /**
     * Test of getPrice method, of class Product.
     * /
    public void testGetPrice() {
       System.out.println("getPrice");
        double expResult = 699.99;
       double result = prod1.getPrice();
       assertEquals(expResult, result);
    }
}
```

- 5. Ejecutar el Caso de Prueba para comprobar las funcionalidades de la clase Product
 - Se ejecuta el caso de prueba seleccionando: Run | Test "Product"..
 - En las ventanas *JUnit Test Result* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación

L	
JUnit Test Results	
All 4 tests passed. All 4 tests passed. • • testGetCode passed (0,0 s) • • • testGetTitle passed (0,0 s) • • • testGetDescription passed (0,0 s) • • • testGetPrice passed (0,0 s)	
🔁 Output 🛛 🖃 HTTP Monitor 🛛 🐼 JUnit Test Results	

L					
Out	put - shopping (test-single)				
IN	Testsuite: shopping.ProductTest				
IN .	getCode				
101	getTitle				
	getDescription				
	getPrice				
	Tests run: 4, Failures: 0, Errors: 0, Time elapsed: 0,109 sec				
	Standard Output				
	getCode				
	getTitle				
	getDescription				
	getPrice				
	test-single:				
	BUILD SUCCESSFUL (total time: 0 seconds)				
	Output 📰 HTTP Monitor 🐼 JUnit Test Results				

6. Crear un caso de prueba usando JUnit para la clase ShoppingCart que pruebe cada uno de sus métodos, usando el método setUp() para crear los objetos y estructuras de datos necesarias para establecer el contexto de los métodos del caso de prueba y el método tearDown() para liberar los recursos reservados para realizar la ejecución del caso de prueba.

Para esto hay que realizar los siguientes pasos:

- a. En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba o TestCase, que en este caso es la clase ShoppingCart.
 - En el menú seleccionar: Tools |Create Junit Tests.
 - Aparece una ventana emergente para seleccionar la versión de Junit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x.
 - Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba.
 Verificar que estén marcadas todas las opciones. Luego se debe seleccionar el botón "*OK*".
 - Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase del Framework JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase testCase del framework junit, y el esqueleto de código para los métodos de prueba de la clase Product, para probar cada uno de los métodos de la clase usando el método Assert.
 - Se deben declarar como atributos privados los elementos que se van a usar en el método setUp() y hacer las inicializaciones correspondientes en el método. En este caso se va a utilizar una instancia de la clase ShoppingCart y una instancia de la clase

Product denominadas respectivamente cart1 y prod1, que se declaran como atributo privados de la clase TestShoppingCart y que se crearán e inicializarán en el cuerpo del método setUp(). El código asociado a los atributos privados y al método setUp() de la clase TestShoppingCart es el siguiente:

```
public class ShoppingcartlTest extends TestCase {
    private ShoppingCart cartl;
    private Product prodl;

protected void setUp() {
    cart1 = new ShoppingCart();
    prod1 = new Product("P001", "Plasma TV LG 32", "Plasma TV
with TDT Decod. and high resolution Screen", 699.99);
    cart1.addItem(prod1);
}
```

- En los métodos de prueba se eliminan las instrucciones:

```
ShoppingCart instance = new ShoppingCart();
Product instance = null;
```

Y en el código de cada método de prueba se sustituye el ShoppingCart instance y Product instance por ShoppingCart cart1 y Product prod1 creado en el método setUp() y se colocan los valores esperados de acuerdo a los valores de los atributos de prod1.

- Se debe completar el esqueleto del método testAddItem comprobando que el número de ítems en el carrito sea correcto y que el valor de la compra se actualice correctamente una vez añadido un item.
- Se debe completar el esqueleto de código del método testEmpty() indicando que para hacer la prueba el número de elementos de la lista debe ser igual a 0.
- Adicionalmente, para que cada uno de los métodos de prueba compile correctamente, se debe borrar de cada uno las dos últimas líneas de código, es decir, se debe borrar de cada método de prueba el siguiente código:

```
// TODO review the generated test code and remove the
default call to fail.
fail("The test case is a prototype.");
```

 El método tearDown(), que se encarga de liberar los recursos reservados en el métodos setUp(), colocará en null a los apuntadores de los objetos creados en el método SetUp().
 Para esto se sustituirá en el esqueleto del método tearDown() la instrucción:

```
super.tearDown();
```

por las instrucciones:

cart1 = null;

prod1 = null;

- Se pulsa < Crtl+S> para salvar el trabajo
- Se compila la clase de prueba con Build | Compile "ProductTest.java".
- En la ventana Output se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.
- El código del caso de prueba ShoppingCartTest debería ser el siguiente:

```
package shopping;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class ShoppingCartTest extends TestCase {
    private ShoppingCart cart1;
    private Product prod1;
    public static Test suite() {
        TestSuite suite = new TestSuite(ShoppingCartTest.class);
        return suite;
    }
    @Override
    protected void setUp() {
        cart1 = new ShoppingCart();
        prod1 = new Product("P001", "Plasma TV LG 32", "Plasma TV
with TDT Decod. and high resolution Screen", 699.99);
        cart1.addItem(prod1);
    }
```

```
/**
    * Tears down the test fixture.
     * Called after every test case method.
    */
   @Override
   protected void tearDown() {
       cart1 = null;
       prod1 = null;
    }
     /**
     * Test of getBalance method, of class ShoppingCart.
    * /
   public void testGetBalance() {
       System.out.println("getBalance");
       double expResult = 699.99;
       double result = cart1.getBalance();
       assertEquals(expResult, result);
    }
     /**
     * Test of addItem method, of class ShoppingCart.
     * /
   public void testAddItem() {
     System.out.println("addItem");
       Product prod2 = new Product("P002", "DVD player Sanmsung",
"DVD player with remote control and programming features", 39.99);
       cart1.addItem(prod2);
       assertEquals(2, cart1.getItemCount());
       double expectedBalance = prod1.getPrice() + prod2.getPrice();
       assertEquals(expectedBalance, cart1.getBalance());
    }
     /**
    * Test of getItemCount method, of class ShoppingCart.
    */
   public void testGetItemCount() {
       System.out.println("getItemCount");
       int expResult = 1;
       int result = cart1.getItemCount();
       assertEquals(expResult, result);
    }
     /**
     * Test of removeItem method, of class ShoppingCart.
    */
   public void testRemoveItem() {
     System.out.println("removeItem");
       cart1.removeItem(prod1);
       assertEquals(0, cart1.getItemCount());
    }
    /**
    * Test of empty method, of class ShoppingCart.
    */
   public void testEmpty() {
       System.out.println("empty");
       cart1.empty();
       assertEquals(0, cart1.getItemCount());
    }
```

7. Ejecutar el caso de prueba para de la clase ShoppingCart

- Ejecuta el caso de prueba seleccionando Run | Test "ShoppingCart".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación.



8. Crear una *suite de pruebas* usando JUnit para el paquete shopping que pruebe todas sus funcionalidades

Para esto hay que realizar los siguientes pasos:

- En la ventana *Projects* se selecciona el paquete para la que se quiere construir la suite de pruebas, que en este caso es el paquete Shopping
- En la barra de menú seleccionar: Tools | Create Junit Tests.
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto de la suite de pruebas. Verificar que estén marcadas todas las opciones y seleccionar *OK*.
- Entonces aparecerá la ventana asociada al esqueleto de código de la suite de prueba generada para el paquete shopping. Como en el paquete shopping se encuentra la clase Main, cuando se genera la *suite* de prueba para el paquete se genera automáticamente un caso de prueba

para la clase Main. Para que el caso de prueba de la clase Main compile se debe eliminar del esqueleto de la clase MainTest, específicamente en el método testMain las instrucciones

```
// TODO review the generated test code and remove the default call
to fail.
fail("The test case is a prototype.");
```

- Se pulsa <*Crtl*+*S*> para salvar el trabajo.
- Se compila la suite de prueba generada seleccionando: Build | Compile "ShoppingSuite.java".
- En la ventana Output se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.
- El código de la suite de prueba ShoppingSuite debería ser el siguiente:

```
package shopping;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class ShoppingSuite extends TestCase {
    public ShoppingSuite(String testName) {
        super(testName);
    }
    public static Test suite() {
        TestSuite suite = new TestSuite("ShoppingSuite");
        suite.addTest(shopping.MainTest.suite());
        suite.addTest(shopping.ProductTest.suite());
        suite.addTest(shopping.ShoppingCartTest.suite());
        return suite;
    }
    protected void setUp() throws Exception {
        super.setUp();
    }
    protected void tearDown() throws Exception {
        super.tearDown();
    }
}
```

9. Ejecutar la suite de pruebas del paquete shopping que prueba todas las funcionalidades de sus clases constituyentes

- Se ejecuta la suite de prueba de prueba seleccionando: Run | Run File | Run "ShoppingSuite.java".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución de la suite de pruebas, como se muestra a continuación

	1.1	i i Product Lest Java				
		All 10 bests passed				
	All To cests passed.					
		Inopping.bioppingbace passed				
		testMain passed (U,U s)				
		 testGetCode passed (0,0 s) 				
		🛶 🕘 testGetTitle passed (0,0 s)				
		testGetDescription passed (0,156 s)				
		testGetPrice passed (0,0 s)				
		testGetBalance passed (0,0 s)				
		• • testAddItem passed (0.0 s)				
		testGetTtemCount_passed (0.0 s)				
		testBernoveltern passed (0,0,s)				
		U,UI6 s				
Ì	Out	out - shopping (test-single)				
ſ		init:				
l		deps-jar:				
l		compile:				
L		compile-test-single: Testsuite: cherning SherningSuite				
L		main				
L		getCode				
L		getTitle				
L		getDescription				
L		getPrice				
l		getBalance				
L		getItemCount				
l		removeItem				
l		empty				
l		Tests run: 10, Failures: 0, Errors: 0, Time elapsed: 0,391 sec				
l		Standard Output				
l		main				
l		getCode				
l		getTitle				
l		getDescription getDrice				
l		getBalance				
l		addItem				
1		getItemCount				
		removeItem				
		empty				
1		test-single:				
		BUILD SUCCESSFUL (total time: 1 second)				
ſ	100	utput 🔄 HTTP Monitor 🛛 🐼 JUnit Test Results				
2	_					

Referencias Bibliográficas

- Clark, M. JunitPrimer. Disponible en: <u>http://clarkware.com/articles/JUnitPrimer.html</u>

Práctica

Creación de casos de prueba y suites de pruebas usando el JUnit

Objetivo: crear y ejecutar casos de prueba con *suites de pruebas* usando JUnit y Netbeans para probar las funcionalidades de una aplicación que permite registrar el almacenamiento de paquetes en un almacén.

Esta aplicación está formada por las clases Container, Warehouse y Package, cuya estructura se muestra a continuación:



La clase Container representa un contenedor que puede almacenar paquetes, representados como una lista de paquetes y a partir del cual se puede definir distintos tipos de contenedores, como es el caso de la clase Warehouse, que es una subclase de Container con la característica particular de tener un volumen limitado para almacenar paquetes, representado por el atributo maxCapacity. La clase Package es la clase que representa los paquetes a almacenar y cada paquete ocupa un volumen, representado por el atributo volume.

Los métodos de la clase Container son los siguientes:

- Container(): es el método constructor que crea un nuevo contenedor como una lista enlazada vacía de paquetes
- add(Package p): añade un Paquete al Contenedor. Retorna un valor booleano cuyo valor es true si el paquete fue añadido con éxito y false en caso contrario.
- remove(Package p): elimina un paquete del contenedor. Retorna un booleano cuyo valor es *true* si el objeto Package fue eliminado con éxito al Container o *false* en caso contrario
- getVolume(): retorna el volumen total ocupado por todos los paquetes que están en el contenedor.
- size(): retorna la cantidad de paquetes que hay en el contenedor.
- clear(): vacía el contenedor eliminando todos los paquetes que están almacenados en este.
- contains(Package p): retorna *true* si el Paquete p está almacenado en el Contenedor, de lo contrario retorna *false*.

La clase Warehouse hereda los métodos de la clase Container, sobrescribe el método add(Package p) de la clase Container ya que debe comprobar si hay espacio suficiente en el almacén para almacenar el paquete p. Además, posee lo siguientes métodos adicionales:

- Warehouse(): es constructor que crea un nuevo almacén definiendo su capacidad máxima, además de ser una lista enlazada vacía de paquetes, al ser una subclase de la clase Container.
- getPackages(): este método retorna un *iterador* que contiene todos los paquetes almacenados en el almacén en orden ascendente según su volumen.

La clase Package posee los siguientes métodos:

- Package(): es el método constructor que crea un nuevo paquete definiendo su volumen en el valor del atributo volumen.
- getVolume(): retorna el volumen que ocupa el paquete.

Después de crear las clases en Netbeans se debe:

- Crear y ejecutar los casos de prueba para las funcionalidades propias de cada clase, utilizando los métodos setUp() y tearDown() para evitar la duplicación de código y poder reusar los *fixtures* en cada caso de prueba. A la hora de hacer las pruebas se debe tener en cuenta lo siguiente:
 - Para la realización de las pruebas de las clases Container y
 Warehouse se va a trabajar con un array de 5 packages donde el

volumen de cada uno será el resultado de multiplicar 10 por el valor de la posición que ocupa en el *array* más 1 (10 * (i + 1)), para generar *packages* con volúmenes diferentes.

- En el método SetUp() de las clases de prueba de Container y
 Warehouse se debe comprobar, usando un método assert, que se está trabajando con al menos un Package.
- En la prueba del método add(Package p) además de comprobar que el paquete se ha añadido de manera exitosa, se debe comprobar que no se puede agregar un paquete que ya ha sido añadido previamente.
- En la prueba del método remove(Package p) además de comprobar que se remueve el paquete de manea exitosa, se debe comprobar que no se pueden remover un paquete si no está almacenado.
- En la prueba del método getPackages() se debe comprobar que los paquetes son devueltos ordenados de menor a mayor según su volumen.
- Crear y ejecutar una suite de pruebas que permita ejecutar secuencialmente las pruebas de todas las funcionalidades de la aplicación

Pasos a seguir:

1. Crear un nuevo proyecto de tipo Java Application llamado container

- Para esto se debe seleccionar en el menú File | New Project...
- En la ventana Emergente *New Project* se debe seleccionar la categoría Java y dentro de esta categoría seleccionar el tipo de proyecto "Java Application".
- Pulsa el botón "Next>".
- En la ventana emergente "*New Java Application*" colocar en el campo "Project Name" el valor **Container** y asegurar que estén seleccionadas las dos opciones de la ventana y pulsar el botón "*Finish*".
- Entonces aparecerá el proyecto **Container** con sus carpetas asociadas, el cual puede verse en la ventana *Projects*, que se muestra a continuación:



2. Crear la clase Container con sus atributos y servicios

Para esto hay que realizar los siguientes pasos:

a. Crear la clase Container

- En el proyecto *Container* en la carpeta *Source Packages* seleccionar el paquete *container*.
- Pulsar el botón derecho del ratón y seleccionar la opción New | Java Class...
- En la ventana emergente *New Java Class*, escribir en el campo *Class Name* el valor Container y pulsar el botón "*Finish*"
- Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase Container.

b. Definir los atributos y métodos de la clase Container.

- En la pestaña del código de la clase Container escribir el código de la clase como se muestra a continuación:

```
package container;
import java.util.LinkedList;
import java.util.List;
public class Container {
    List<Package> contents;
    public Container() {
        contents = new LinkedList<Package>();
    }
    public boolean add(Package b) {
        return !contents.contains(b) && contents.add(b);
    }
}
```
```
public boolean remove(Package b) {
      return contents.remove(b);
}
public double getVolume() {
      int sum = 0;
      for (Package b : contents) {
           sum += b.getVolume();
      }
      return sum;
}
public int size() {
      return contents.size();
}
public void clear() {
      contents.clear();
}
public boolean contains(Package b) {
      return contents.contains(b);
}
```

 Pulsar <Ctrl+s> para salvar el trabajo realizado hasta ahora. Nótese que todavía no compila ya que tiene dependencias con otras clases no creadas.

3. Crear la clase Warehouse (subclase de la clase Container) con sus atributos y servicios

- Para esto hay que realizar los mismos pasos que para la creación de la clase Container y una vez escritos el código de la clase Warehouse debe verse de la siguiente forma:

```
package container;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Iterator;
import java.util.List;
public class Warehouse extends Container {
      private final double maxCapacity;
      public Warehouse(double capacity) {
             maxCapacity = capacity;
      }
    @Override
      public boolean add(Package b) {
             if (b.getVolume() + getVolume() <= maxCapacity) {</pre>
                   return super.add(b);
             }
             return false;
      }
      public Iterator<Package> getPackages() {
             List<Package> copyContents = new ArrayList<Package>(contents);
```

```
Collections.sort(copyContents, new Comparator<Package>() {
    public int compare(Package b1, Package b2) {
        //return (int)b1.getVolume() - (int)b2.getVolume();
        //return (int)(b1.getVolume() - b2.getVolume());
        if (b1.getVolume() < b2.getVolume()) {
            return -1;
        } else if (b1.getVolume() == b2.getVolume()) {
            return 0;
        } else {
            return 1;
        }
    }
}
</pre>
```

 Pulsar <Ctrl+s> para salvar el trabajo realizado hasta ahora. Nótese que todavía no compila ya que tiene dependencias con otras clases no creadas.

4. Crear la clase Package con sus atributos y servicios

- Para esto hay que realizar los mismos pasos que para la creación de la clase Container y una vez escritos el código de la clase Package debe verse de la siguiente forma:

```
package container;
public class Package {
```

```
private double volume = 0;
public Package(double volume) {
    this.volume = volume;
}
public double getVolume() {
    return volume;
}
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado hasta ahora.
- Construir el proyecto para verificar que no tiene errores. Esto se hace seleccionando en el menú: Build | Build Main Project.
- En la ventana Output se podrá ver el resultado de la acción Build, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana Output, corregirlos y volver a construir el proyecto.
- 5. Crear un caso de prueba usando JUnit para la clase Container que pruebe cada uno de sus métodos, usando el método setUp() para crear los objetos y estructuras de datos necesarias para establecer el contexto de

los métodos del caso de prueba y las condiciones a considerar establecidas en el enunciado de la práctica.

Para esto hay que realizar los siguientes pasos:

- En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba o *TestCase*, que en este caso es la clase Container.
- En la barra de menú seleccionar Tools | Create Junit Tests.
- Aparece una ventana emergente para seleccionar la versión de JUnit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x y pulsar el botón Select.
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba. Verificar que estén seleccionadas todas las opciones y pulsar el botón *OK*.
- Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase de JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase TestCase de JUnit, y el esqueleto de código para los métodos de prueba de la clase Container, para probar cada uno de los métodos de la clase usando el método assert.
- Una vez creado el caso de prueba para la clase Container este debería de ser:

```
package container;
```

```
import junit.framework.TestCase;
public class ContainerTest extends TestCase {
    private Container container = null;
   private int num_Packages_To_Test = 5;
   private Package[] b = null;
    private double package_Unit_Volume = 10.0;
    public ContainerTest(String testName) {
       super(testName);
    }
    @Override
    protected void setUp() throws Exception {
      assertTrue("Test case error, you must test at least 1 Package",
num_Packages_To_Test > 0);
      // We create the Packages that we need.
       b = new Package[num_Packages_To_Test];
      for (int i=0; i<num_Packages_To_Test; i++) {</pre>
          b[i] = new Package((i+1)*package_Unit_Volume);
      }
        // Now, we create the Container
```

```
container = new Container();
    }
    @Override
    protected void tearDown() throws Exception {
        super.tearDown();
    }
    /**
     * Test of add method, of class Container.
     */
    public void testAdd() {
      container.clear();
      for (int i=0; i<num_Packages_To_Test; i++) {</pre>
           assertTrue("container.add(Package) failed to add a new Package",
container.add(b[i]));
          assertFalse("container.add(Package) seems to allow the same
Package to be added twice", container.add(b[i]));
          assertTrue("container does not contain a Package after it is
supposed to have been added", container.contains(b[i]));
      }
    }
    /**
     * Test of remove method, of class Container.
     * /
    public void testRemove() {
       container.clear();
       assertFalse("container.remove(Package) should fail because Container
is empty, but it didn't", container.remove(b[0]));
      for (int i=0; i<num_Packages_To_Test; i++) {</pre>
           container.clear();
           for (int j=0; j<i; j++) {
             container.add(b[j]);
           for (int j=0; j<i; j++) {</pre>
             assertTrue("container.remove(Package) failed to remove a Package
that is supposed to be inside", container.remove(b[j]));
             assertFalse("container still contains a Package after it is
supposed to have been removed!", container.contains(b[j]));
           for (int j=i; j<num_Packages_To_Test; j++) {</pre>
             assertFalse("container.remove(Package) did not fail for a
Package that is not inside", container.remove(b[j]));
           }
      }
    }
    /**
     * Test of getVolume method, of class Container.
     * /
    public void testGetVolume() {
        System.out.println("getVolume");
        double expResult = 0.0;
        double result = 0.0;
        for (int i=0; i<num_Packages_To_Test; i++) {</pre>
          expResult = expResult + b[i].getVolume();
      }
         container.clear();
        for (int i=0; i<num_Packages_To_Test; i++) {</pre>
          container.add(b[i]);
        result = container.getVolume();
        assertEquals(expResult, result);
```

```
/**
     * Test of size method, of class Container.
    */
    public void testSize() {
        System.out.println("size");
        container.clear();
        for (int i=0; i<num_Packages_To_Test; i++) {</pre>
          container.add(b[i]);
      }
        int result = container.size();
        assertEquals(num_Packages_To_Test, result);
    }
    /**
    * Test of clear method, of class Container.
     * /
    public void testClear() {
       System.out.println("clear");
       container.clear();
    }
    /**
     * Test of contains method, of class Container.
     * /
    public void testContains() {
        System.out.println("contains");
            container.clear();
            for (int i=0; i<num_Packages_To_Test; i++) {</pre>
          container.add(b[i]);
      }
        for (int i=0; i<num_Packages_To_Test-1; i++) {</pre>
                assertTrue("container contains a
Package",container.contains(b[i]));
          }
      }
```

- Pulsar <Ctrl+s> para para salvar el trabajo.
- Se compila la clase de prueba generada seleccionando en el menú Build | Compile "ContainerTest.java".
- En la ventana *Output* se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.

6. Ejecutar el caso de prueba de las funcionalidades de la clase Container

- Se ejecuta el caso de prueba seleccionando en el menú: Run | Test "Container".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación:

🗊 🗠 MyAnn1		Str
JUnit Test Results		
P All 6 tests passed.		
📥 🗄 🏤 container.ContainerTest passed		
• testAdd passed (0,016 s)		
• testRemove passed (0,0 s)		
• testGetVolume passed (0,0 s)		
🧼 🧶 testSize passed (0,0 s)		
🔍 testClear passed (0,0 s)		
testContains passed (0,0 s)		
ο		System out
ē <u>™ MvAnn1</u> Output - Container (test-single)		System out
G		System out
Output - Container (test-single)		Sustem out
Cutput - Container (test-single) Clear contains Tests run: 6. Failures: 0. Error	s: 0. Time elaps	Swatem out
MyAnol Output - Container (test-single) Clear contains Tests run: 6, Failures: 0, Error	s: O, Time elapse	Svatem out
Output - Container (test-single) Clear contains Tests run: 6, Failures: 0, Error Standard Output	s: O, Time elapse	Svatem out ed: 0,344 sec
MwAnn1 Output - Container (test-single) clear contains Tests run: 6, Failures: 0, Error Standard Output getVolume	s: O, Time elapso	Svatem out ed: 0,344 sec
MwAnn1 Output - Container (test-single) clear contains Tests run: 6, Failures: 0, Error 	s: 0, Time elaps	Sustem out ed: 0,344 sec
MwAnn1 Output - Container (test-single) clear contains Tests run: 6, Failures: 0, Error Standard Output getVolume size clear	s: 0, Time elaps	Svatem out ed: 0,344 sec
MyAnn1 Output - Container (test-single) clear contains Tests run: 6, Failures: 0, Error Standard Output getVolume size clear contains	s: 0, Time elaps	Svatem out ad: 0,344 sec
MyAnn1 Output - Container (test-single) Clear contains Tests run: 6, Failures: 0, Error 	s: 0, Time elaps	Svatem out
MvAnn1 Output - Container (test-single) Clear contains Tests run: 6, Failures: 0, Error	s: 0, Time elaps	Svatem out

7. Crear un caso de prueba usando JUnit para la clase Warehouse que pruebe cada uno de sus métodos, usando el método setUp() para crear los objetos y estructuras de datos necesarias para establecer el contexto de los métodos del caso de prueba y las condiciones a considerar establecidas en el enunciado de la práctica.

Para esto hay que realizar los siguientes pasos:

- En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba, que en este caso es la clase Warehouse.java
- En la barra de menú seleccionar Tools | Create Junit Tests.
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba. Verificar que estén marcadas todas las opciones y pulsar el botón *OK*. Nótese que no pregunta por la versión de JUnit, ya que seleccionamos la versión 3.x en para ContainerTest.
- Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase del

framework JUnit y que tiene el esqueleto de código para los métodos
setUp() y tearDown() heredados de la clase TestCase del JUnit, y
el esqueleto de código para los métodos de prueba de la clase
Warehouse, para probar cada uno de los métodos de la clase usando el
método assert.

- Una vez creado el caso de prueba para la clase Warehouse este debe verse de la siguiente manera:

```
package container;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Random;
import junit.framework.TestCase;
public class WarehouseTest extends TestCase {
    private Warehouse Warehouse = null;
    private int num_Packages_To_Test = 5;
    private int Warehouse_Volume = num_Packages_To_Test - 1;
    private Package[ ] b = null;
    private double package_Unit_Volume = 10.0;
    @Override
    protected void setUp() throws Exception {
      assertTrue("Test case error, you must test at least 1 Package",
num_Packages_To_Test > 0);
      assertTrue("This test case is set up assuming that the Warehouse
cannot contain all the Packages, please check and change parameters",
num_Packages_To_Test > Warehouse_Volume);
      double Warehouse_capacity = 0;
      // We create the Packages that we need.
        b = new Package[num_Packages_To_Test];
      for (int i=0; i<num_Packages_To_Test; i++) {</pre>
           if (i<Warehouse_Volume) {</pre>
             Warehouse_capacity += (i+1)*package_Unit_Volume;
          b[i] = new Package((i+1)*package_Unit_Volume);
      }
      // Now, we create the Warehouse once we figure out how big a Warehouse
we
      // need.
        Warehouse = new Warehouse(Warehouse_capacity);
    }
    /** Test to check that Warehouse.add(Package) is implemented correctly */
    public void testAdd() {
      Warehouse.clear();
      for (int i=0; i<Warehouse_Volume; i++) {</pre>
          assertTrue("Warehouse.add(Package) failed to add a new Package!",
Warehouse.add(b[i]));
          assertFalse("Warehouse.add(Package) seems to allow the same
Package to be added twice!", Warehouse.add(b[i]));
```

assertTrue("Warehouse does not contain a Package after it is

PRÁCTICAS DEL CURSO PRUEBA DE SOFTWARE

```
supposed to have been added!", Warehouse.contains(b[i]));
      for (int i=Warehouse_Volume; i<num_Packages_To_Test; i++) {</pre>
          assertFalse("Warehouse.add(Package) allows a Package to be added
even though it is already full!", Warehouse.add(b[i]));
      }
    }
    /** Test to check that Warehouse.getPackages() is implemented correctly
* /
    public void testGetPackages() {
      Random rnd = new Random();
          Warehouse.clear();
           // We put all the Packages into a list.
           LinkedList<Package> list = new LinkedList<Package>();
           for (int i=0; i<Warehouse_Volume; i++) {</pre>
             list.add(b[i]);
           }
          // First we add the Packages to the Warehouse in some random
order.
          for (int i=0; i<Warehouse_Volume; i++) {</pre>
             Warehouse.add(list.remove(rnd.nextInt(list.size())));
           }
          // Next we call the iterator and check that the Packages come out
in the
       correct order.
          Iterator<Package> it = Warehouse.getPackages();
          int count = 0;
          while (it.hasNext() && count < Warehouse_Volume) {</pre>
             Package Package = it.next();
             assertTrue("Packages are not returned by Warehouse.getPackages()
iterator in the correct order", b[count] == Package);
             if (b[count] != Package) {
                 break;
             }
             count++;
           }
          assertEquals("Warehouse.getPackages() did not return all the
Packages", Warehouse_Volume, count);
      }
```

- Pulsar <Ctrl+s> en el teclado para salvar el trabajo realizado hasta ahora.
- Se compila la clase de prueba generada seleccionando en el menú Build | Compile "WarehouseTest.java".
- En la ventana *Output* se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.

8. Ejecutar el caso de prueba para comprobar las funcionalidades de la clase Warehouse

- Se ejecuta el caso de prueba seleccionando en el menú Run | Test "Warehouse".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación



9. Crear un caso de prueba usando JUnit para la clase Package que pruebe cada uno de sus métodos

- En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba, que en este caso es la clase Package.
- En la barra de menú seleccionar en el menú Tools | Create Junit Tests.
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba. Verificar que estén marcadas todas las opciones y pulsar *OK*.
- Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase de JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase TestCase de JUnit, y el esqueleto

de código para los métodos de prueba de la clase Package, para probar cada uno de los métodos de la clase usando el método assert.

- Una vez creado el caso de prueba para la clase Package este debe verse de la siguiente manera:

```
package container;
import junit.framework.TestCase;
public class PackageTest extends TestCase {
    public PackageTest(String testName) {
       super(testName);
    }
    /**
     * Test of getVolume method, of class Package.
    * /
    public void testGetVolume() {
        System.out.println("getVolume");
        Package instance = new Package(1.0);
        double expResult = 1.0;
        double result = instance.getVolume();
        assertEquals(expResult, result);
    }
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado hasta ahora.
- Se compila la clase de prueba generada seleccionando desde el menú Build | Compile "WarehouseTest.java".
- En la ventana *Output* se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.

10. Ejecutar el caso de prueba para comprobar las funcionalidades de la clase Package

- Se ejecuta el caso de prueba seleccionando Run | Test "Package".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación

The test passed. Container.PackageTest passed Set testGetVolume passed (0,015 s) Coutput TestGetVolume passed (0,015 s) Coutput TestGetVolume passed (0,015 s) Coutput TestGetVolume TestGetVolume TestSingle:	JUnit Test Results	
<pre>container.PackageTest passed testGetVolume passed (0,015 s) lotut important for important imported important important</pre>	The test passed.	
<pre> • testGetVolume passed (0,015 s) • TTP Monitor</pre>	📥 📩 📩 container.PackageTest passed	
Output Image: HTTP Monitor Output - Container (test-single) Output - Container (test-single) Compile:	🔲 🔍 🕒 testGetVolume passed (0,015 s)	
Output Image: HTTP Monitor Output Container (test-single) Output Container (test-single) Compile-test-single: compile-test-single: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
Output Image: HTTP Monitor Output Compile: compile: <t< td=""><td></td><td></td></t<>		
Image: State of the state		
Output Image: Image		
Output Image: HTTP Monitor Output - Container (test-single) Output - Container (test-single) deps-jar: compile: compile: <td></td> <td></td>		
Output HTTP Monitor Output - Container (test-single) deps-jar: compile: compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
Output HTTP Monitor Output - Container (test-single) Output - Container (test-single) compile-test-single: compile-test-single: Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
Output ITTP Monitor Itest Results Output - Container (test-single) Itest compile: compile: compile-test-single: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
Output - Container (test-single) Image: Compile - C	🔁 Output 🛛 📰 HTTP Monitor 🛛 🐼 JUnit Test Results	
Output - Container (test-single) Image: compile: compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
Output - Container (test-single) Image: Compile: compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec		
<pre>deps-jar: compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec </pre>	Output - Container (test single)	
<pre>deps-jar: compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec </pre>		
Compile: compile-test-single: Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec 	deps-jar:	
Testsuite: container.PackageTest getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec 	compile:	
getVolume Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec 	Testsuite: container.PackageTest	
Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec 	getVolume	
	Tests run: 1, Failures: 0, Errors: 0, Time elapsed: 0,921 sec	
getVolume test-single: BUILD SUCCESSFUL (total time: 3 seconds) Coutput HTTP Monitor Couton JUnit Test Results	Standard Output	
test-single: BUILD SUCCESSFUL (total time: 3 seconds)	getVolume	
BUILD SUCCESSFUL (total time: 3 seconds)	test-single:	
Cutput 📰 HTTP Monitor 🐼 JUnit Test Results	BUILD SUCCESSFUL (total time: 3 seconds)	
	Cutout	

11. Crear una *suite* de pruebas usando JUnit para el paquete container que pruebe todas sus funcionalidades

- En la ventana *Projects* se selecciona el paquete para la que se quiere construir la suite de pruebas, que en este caso es el paquete container.
- En la barra de menú seleccionar Tools | Create Junit Tests.
- Aparece la ventana emergente Create Tests donde se define los parámetros para generar el esqueleto de la suite de pruebas. Verificar que estén marcadas todas las opciones y pulsar OK. (Puede que se regeneren métodos de prueba añadiéndoles un "1" al nombre del método de prueba, estos métodos se pueden borrar o y si no se borran, los únicos que deberían de generar un error al ejecutar en JUnit).
- Entonces aparecerá la ventana asociada al esqueleto de código de la suite de prueba generada para el paquete container.
- Como en el paquete container se encuentra la clase Main, cuando se genera la *suite* de prueba para el paquete se genera automáticamente un caso de prueba para la clase Main. Para que el caso de prueba de la

clase Main compile se debe eliminar del esqueleto de la clase MainTest, específicamente en el método testMain las instrucciones:

```
// TODO review the generated test code and remove the default call to
fail.
fail("The test case is a prototype.");
```

- Pulsar <Ctrl+s> para salvar el trabajo.
- Seleccionar ContainerSuite.java en el proyecto y compilar la suite de prueba generada desde el menú con Build | Compile "ContainerSuite.java".
- En la ventana *Output* se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente. El código de la suite de prueba ContainerSuite debería ser el siguiente:

```
package container;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;
public class ContainerSuite extends TestCase {
    public ContainerSuite(String testName) {
       super(testName);
    }
    public static Test suite() {
        TestSuite suite = new TestSuite("ContainerSuite");
        suite.addTest(container.MainTest.suite());
       suite.addTest(container.PackageTest.suite());
       suite.addTest(container.ContainerTest.suite());
       suite.addTest(container.WarehouseTest.suite());
       return suite;
    }
    protected void setUp() throws Exception {
        super.setUp();
    }
    protected void tearDown() throws Exception {
       super.tearDown();
    }
```

12. Ejecutar la *suite* de pruebas del paquete container que prueba todas las funcionalidades de sus clases constituyentes

- Se ejecuta la suite de prueba de prueba seleccionando: Run | Run File | Run "ContainerSuite.java".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución de la suite de pruebas, como se muestra a continuación



Práctica

Pruebas unitarias con Mocks Objects usando JUnit y EasyMock

Objetivo: crear y ejecutar casos de pruebas usando Mocks Objects para probar una clase de manera aislada de sus clases colaboradoras usando Mocks Objects con la herramienta *EasyMocks* y el framework *JUnit* y *Netbeans*.

La aplicación con la que se va a trabajar en esta práctica es un Videoclub en Internet. La aplicación está compuesta por un lado el cliente (clase ClientVideoClubService) y por otro lado el servidor (interface IVideoClubService) que entrega los videos (interface IVideo). El lado cliente usa al servidor para obtener los videos y mostrarlos a los usuarios. Esta aplicación está formada por las interfaces IVideoClubServices, IVideo y la clase ClientVideoClubService cuya estructura se muestra a continuación:



Después de crear las interfaces y clases a trabajar en el entorno Netbeans 6.0 se mostrará como crear y ejecutar un caso de prueba para la clase ClientVideoClubService usando *Mocks Objects* para realizar la prueba simulando con estos los objetos de las interfaces que colaboran con la clase a probar. Todo se hará con Netbeans usando JUnit y la herramienta EasyMock para manejar los *Mocks Objects*.

1. Crear un nuevo proyecto de tipo Java Application llamado VideoClub

- Para esto se debe seleccionar en el menú: File | New Project ...
- En la ventana emergente *New Project* se debe seleccionar la categoría Java y dentro de esta categoría seleccionar el tipo de proyecto "*Java Application*"
- Pulsar el botón "Next>"
- En la ventana emergente *New Java Application* colocar en el campo *Project Name* el valor *VideoClub* y asegurar que estén seleccionadas las dos opciones de la ventana.
- Pulsar el botón "Finish"

Entonces aparecerá el proyecto VideoClub con sus carpetas asociadas.

2. Crear la clase ClientVideoClubService con sus atributos y servicios

Para esto hay que realizar los siguientes pasos:

- a) Crear la clase ClientVideoClubService
 - En el Proyecto *VideoClub* en la carpeta *Source Packages* seleccionar el paquete *videoclub*.
 - Pulsar el botón derecho del ratón y seleccionar New | Java Class....
 - En la ventana emergente New Java Class, escribir en el campo Class Name el valor ClientVideoClubService y pulsar el botón "Finish".
 - Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase ClientVideoClubService.
- b) Definir los atributos y métodos de la clase ClientVideoClubService
 - En la pestaña del código de la clase ClientVideoClubService escribir entre los paréntesis delimitadores del código de la clase los atributos y los métodos de la clase como se indica a continuación:

package videoclub;

```
this.remoteVideoClubService = remoteVideoClubService;
}
public IVideo getVideo(int VideoNumber) {
    return remoteVideoClubService.getVideo(VideoNumber);
}
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado.
- 3. Crear la interfaz IVideoClubService con su servicio getVideo()

a) Crear la interfaz IVideoClubService

- En el Proyecto VideoClub en la carpeta Source Packages seleccionar el paquete videoclub.
- Pulsar el botón derecho del ratón y seleccionar: New | Java Interface....
- En la ventana emergente *New Java Interface*, colocar en el campo *Interface Name* el valor IVideoClubService y pulsar el botón *"Finish"*.
- Entonces se abrirá la pestaña correspondiente al esqueleto de código de la interfaz IVideoClubService.

b) Definir los métodos de la interfaz IVideoClubService.

 En la pestaña del código de la interfaz colocar entre los paréntesis delimitadores del código el método de la interfaz. Una vez escritos el código de la interfaz debe verse de la siguiente forma:

```
package videoclub;
public interface IVideoClubService {
    IVideo getVideo(int number);
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado.

4. Crear la interfaz IVideo clase con sus servicios

- Para esto hay que realizar los mismos pasos que para la creación de la interfaz IVideoClubService y una vez escrito el código debe verse de la siguiente forma:

```
package videoclub;
import java.io.InputStream;
```

```
public interface IVideo {
    int getNumber();
    String getTitle();
    InputStream getDataAsStream();
}
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado.
- Construir el proyecto para verificar que no tiene errores seleccionando en el menú: Build | Build Main Project.
- En la ventana Output se podrá ver el resultado de la acción Build, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana Output, corregirlos y volver a construir el proyecto
- 5. Crear y ejecutar un caso de prueba para la clase ClientVideoClubService usando Mocks Objects para realizar la prueba simulando con estos los objetos de las interfaces que colaboran con la clase a probar. Todo se hará en el ambiente de desarrollo Netbeans usando JUnit y la herramienta EasyMock para manejar los Mocks Objects.

Para esto hay que realizar los siguientes pasos:

- a) Añadir a la carpeta *TestLibraries* del proyecto *VideoClub* la biblioteca de EasyMock¹.
 - Seleccionar la carpeta *Test Libraries* y presionar el botón derecho del ratón.
 - Del menú emergente seleccionar la opción Add Library...
 - Si EasyMock no se encuentra en la lista, añadirlo pulsando *Create*. En la ventana emergente, en el campo *Library Name* añadir EasyMock y a su vez, en la de carpetas, seleccionar el fichero easymock.jar (donde lo hayamos descomprimido el fichero easymock2.4.zip.

Add Library	
Available Libraries	Create New Library X Library Name: EasyMock2.4 Library Type: Class Libraries
Add Library Cancel	OK Cancel

¹ EasyMock2.4 debe haberse bajado del sitio http://www.easymock.org/Downloads.html y descromprimirse, por ejemplo, en c:\

- Seleccionar la librería de clases **EasyMock2.4** y presionar el botón "Add Library".



b) Construir el caso de prueba para la clase ClientVideoClubService.

- En la ventana *Projects* se selecciona la clase para la que se quiere construir el caso de prueba, que en este caso es la clase ClientVideoClubService.java.
- En la barra de menú seleccionar Tools | Create Junit Tests.
- Aparece una ventana emergente para seleccionar la versión de JUnit que se va a usar para crear el caso de prueba. Seleccionar la opción JUnit 3.x y pulsar el botón "Select".
- Aparece la ventana emergente *Create Tests* donde se define los parámetros para generar el esqueleto del caso de prueba. Verificar que estén marcadas todas las opciones excepto la opción *Default Methods Bodies* y pulsar *OK*.
- Entonces aparecerá la ventana asociada al esqueleto de código del caso de prueba generado, que es una subclase de la clase TestCase de JUnit y que tiene el esqueleto de código para los métodos setUp() y tearDown() heredados de la clase TestCase del framework JUnit.
- Definir dos Mock Objects para cada una de las interfaces colaboradoras de la clase que se está probando llamados remoteVideoClubServiceMock y Video28Mock e instanciarlos en el método setUp() del caso de prueba usando el método createMock() de la librería EasyMock. El código asociado a este paso se muestra a continuación:

```
import org.easymock.EasyMock;
import junit.framework.TestCase;
public class ClientVideoClubServiceTest extends TestCase {
    private IVideoClubService remoteVideoClubServiceMock;
```

```
private IVideo Video28Mock;
protected void setUp() throws Exception {
    super.setUp();
    Video28Mock = EasyMock.createMock(IVideo.class);
    remoteVideoClubServiceMock =
        EasyMock.createMock(IVideoClubService.class);
}
```

 Crear un método para probar el método constructor de la clase considerando como una excepción el que no se coloque como parámetro una instancia de la interfaz IVideoClubService. El código asociado a este método es el siguiente:

```
public void testClientVideoClubService() {
    try {
        new ClientVideoClubService(null);
        fail("Expected IllegalArgumentException");
    } catch (IllegalArgumentException e) {
        // expected
    }
        new ClientVideoClubService(remoteVideoClubServiceMock);
    }
}
```

- Crear un método para probar el método getVideo() usando los *Mocks Objects* creados en el método SetUp(). Para esto se debe:
 - Establecer las expectativas de los Moks Objects usando el método EasyMock.expect(), definiendo que el número del video a obtener es 28 y que el video a retornar es el Video28Mock.
 - Colocar en modo Replay el mock object remoteVideoClubServiceMock usando el método EasyMock.replay().
 - Ejecutar el método que se vaya a probar realizando la prueba correspondiente
 - Comprobar al finalizar la prueba que el mock object remoteVideoClubServiceMock cumplió las expectativas, usando el método EasyMock.verify().
- El código de este método de prueba es el siguiente:

```
public void testGetVideo() throws Exception {
        EasyMock.expect(remoteVideoClubServiceMock.getVideo(28))
        .andReturn(Video28Mock);
      EasyMock.replay(remoteVideoClubServiceMock);
        IVideoClubService clientVideoClubService =
            new clientVideoClubService(remoteVideoClubServiceMock);
        IVideo result = clientVideoClubService.getVideo(28);
        assertEquals(Video28Mock, result);
        EasyMock.verify(remoteVideoClubServiceMock);
    }
}
```

- Una vez creado el caso de prueba para la clase ClientVideoClubService este debe verse de la siguiente manera:

```
package videoclub;
import org.easymock.EasyMock;
import junit.framework.TestCase;
public class ClientVideoClubServiceTest extends TestCase {
   private IVideoClubService remoteVideoClubServiceMock;
    private IVideo Video28Mock;
   protected void setUp() throws Exception {
        super.setUp();
        Video28Mock = EasyMock.createMock(IVideo.class);
        remoteVideoClubServiceMock =
            EasyMock.createMock(IVideoClubService.class);
    }
    public void testClientVideoClubService() {
        try {
            new ClientVideoClubService(null);
            fail("Expected IllegalArgumentException");
        } catch (IllegalArgumentException e) {
            // expected
        new ClientVideoClubService(remoteVideoClubServiceMock);
    }
    /**
     * Test of getVideo of class ClientVideoClubService.
     * /
    public void testGetVideo() throws Exception {
        EasyMock.expect(remoteVideoClubServiceMock.getVideo(28))
            .andReturn(Video28Mock);
        EasyMock.replay(remoteVideoClubServiceMock);
        IVideoClubService clientVideoClubService =
           new ClientVideoClubService(remoteVideoClubServiceMock);
        IVideo result = clientVideoClubService.getVideo(28);
        assertEquals(Video28Mock, result);
        EasyMock.verify(remoteVideoClubServiceMock);
    }
```

- Pulsar <Ctrl+s> para salvar el trabajo realizado.
- Se compila la clase de prueba generada seleccionando Build Compile "ClientVideoClubServiceTest.java".

En la ventana *Output* se ve el resultado de la compilación, el cual debería ser exitoso. De no serlo se revisan los errores indicados, se arreglan y se vuelve a compilar la clase hasta que compile exitosamente.

6. Ejecutar el Caso de Prueba para comprobar las funcionalidades de la clase Container.

- Se ejecuta el caso de prueba seleccionando Run | Test "ClientVideoClubService".
- En las ventanas *TestResult* y *Output* pueden verse los resultados de la ejecución del caso de prueba, como se muestra a continuación.



_	deps-jar:
	compile:
	compile-test-single:
	Testsuite: videoclub.ClientVideoClubServiceTest
	Tests run: 2, Failures: 0, Errors: 0, Time elapsed: 0,469 sec
	test-single:
	BUILD SUCCESSFUL (total time: 1 second)
	· · · · ·

Práctica

Pruebas unitarias de programas en C#.NET usando NUnit

Objetivo: crear y ejecutar casos de pruebas de clase escritas en C# (desde el entorno Visual C# 2008 Express y el framework .NET 2.0) usando NUnit 2.4.8

La aplicación con la que se va a trabajar en esta práctica es una aplicación bancaria con una clase Account (Cuenta) que representa las cuentas de una entidad bancaria. Sobre las cuentas se puede consultar su saldo, hacer depósitos de dinero, hacer retiros de dinero y hacer transferencias de dinero a otras cuentas. Esto se hace respectivamente usando los métodos Balance(), Deposit (amount), Withdraw(amount) y TransferFunds(account destination, amount). El diagrama de la clase Account se muestra a continuación.



Después de crear la clase en el ambiente Visual C# 2008 Express se debe:

Crear y ejecutar un caso de prueba para la clase Account usando el framework NUnit 2.4.8 y luego ejecutar el caso de prueba desde la interfaz gráfica de NUnit.

Pasos a seguir:

1. Crear un nuevo proyecto de tipo *Biblioteca de Clases* llamado Bank

- Para esto se debe seleccionar en el menú: Archivo | Nuevo Proyecto.
- En la ventana Emergente *Nuevo Proyecto* se debe seleccionar la categoría *Biblioteca de clases* y en el campo *Nombre* escribir Bank. Pulsar el botón "*Aceptar*".

😅 CSharp - Microsoft Visual C# 2008 Ex	press	_ <u> </u>
Archivo Edición Ver Proyecto Gen	erar Depurar Datos Herramientas Ventana Ayuda	
i 🛅 🖽 • 📂 🔛 🥔 X 🖻 🛍 1) - (박 - 루 - 특) 🕨 🔹 🕐	• 💀 🕿 🐋 🎌 🖬 🗉 • 🖕
Cuado de herrannertes	Nevo proyecto Plantillas: Plantillas instaladas de Visual Studio Image: Plantillas: Aplicación de dases Aplicación de dases Image: Plantillas Image: Plantillas Discorr Buscar Proyecto para crear una biblioteca de dases de C# (.dl). (.NET Framework 3.5) Nombre: Bank	2 X E X E X E X E X E X E X E X E

 Entonces aparecerá el proyecto Bank con sus carpetas asociadas, como se muestra en la siguiente Figura.



2. Agregar el Framework NUnit en la carpeta References

Para poder crear las clases de pruebas sebe añadir el framework NUnit a las librerías que se van a usar en el proyecto. Para esto ha que ejecutar los siguientes pasos:

- En el Explorador de soluciones seleccionar la carpeta References



- Presionar el botón derecho del ratón y seleccionar la opción Agregar referencia... Entonces aparecerá la ventana emergente Agregar referencia. En esta ventana seleccionar el componente *nunit.framework*¹ y pulsar el botón Aceptar.

Nombre de componente 🔺	Versión	Motor en	Ruta de accesc 4
mock-assembly	2.4.8.0	v2.0.50727	C:\Archivos de
mscorlib	2.0.0.0	v2.0.50727	c:\WINDOWS\/
nonamespace-assembly	2.4.8.0	v2.0.50727	C:\Archivos de
notestfixtures-assembly	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.core	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.core.extensions	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.core.interfaces	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.core.tests	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.extensions.tests	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.fixtures	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.fixtures.tests	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.framework	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.framework.extensions	2.4.8.0	v2.0.50727	C:\Archivos de
nunit.framework.tests	2.4.8.0	v2.0.50727	C:\Archivos de
pupit.mocks	2.4.8.0	v2.0.50727	C:\Archivos de

- Entonces en la carpeta *References* aparecerá el *framework* Nunit, como se muestra en la siguiente figura, el cual ya se puede usar para definir las clases de prueba en este proyecto.

¹ Se debe haber instalado previamente el Framework NUnit 2.4.8 para .NET 2.0, el cual se obtiene de http://www.nunit.org/, el archivo a descargar e instalar es NUnit-2.4.8-net-2.0.msi



- 3. Crear la clase Account con sus atributos y servicios
 - Seleccionar el proyecto Bank
 - Presionar el botón derecho del ratón y en menú de contexto que aparece seleccionar la opción Agregar | Nuevo elemento... Entonces aparecerá la ventana emergente "Agregar nuevo elemento –Bank", como se muestra a continuación:

A	gregar nuevo e	lemento - Bai	nk						<u>?×</u>	
i	Plantillas:								0 0 0-0- 0 0 0 0-0-	
	Plantillas instaladas de Visual Studio									
		c #			¢#	-				
	ADO.NET Entity D	Archivo de código	Archivo de configuración	Archivo de configuraci	Archivo de información	Archivo de manifiesto	Archivo de recursos	Archivo de texto		
				C#		E3 E 3	200 200 200 200 200 200 200 200 200 200	•		
	Archivo XML	Base de datos basada en	Base de datos local	Clase	Clases de LINQ to SQL	Conjunto de datos	Control de usuario	Control de usuario (WPF)		
			S⊂#	Ct#						
	Cuadro Acerca de	Formulario primario MDI	Interfaz	Visualizador del depurador	Windows Forms				-	
	Definición de clas	e vacía								
	Nombre:	Class2.cs								
							Ag	regar Ca	incelar	

- En la ventana emergente seleccionar la opción Clase C#, escribir en el campo Nombre Account.cs como nombre de la clase y pulsar "Agregar".
- Entonces aparecerá la clase Account en la estructura del proyecto y se desplegará la pestaña asociada para escribir su código, como se muestra a continuación:

- Escribir el código de la clase Account, como aparece a continuación:



- Guardar los cambios realizados seleccionando la opción Archivo | Guardar Account.cs o la opción Archivo | Guardar todo. También se puede hacer pulsando los iconos asociados a estas opciones
- Compilar el proyecto seleccionando el menú Generar | Generar Solución.
- 4. Crear la clase de prueba AccountTest usando NUnit

- Seleccionar el proyecto Bank.
- Pulsar el botón derecho del ratón y en el menú de contexto seleccionar la opción Agregar | Nuevo elemento....
- En la ventana emergente seleccionar la opción Clase C#, escribir en el campo Nombre AccountTest.cs como nombre de la clase y pulsar el botón "Agregar".
- Entonces aparecerá la clase AccountTest en la estructura del proyecto y se desplegará la pestaña asociada para escribir su código, como se muestra a continuación:



- Escribir el código de los métodos de la clase de prueba:
 - Se debe indicar, antes del nombre de la clase, que la clase AccountTest es una clase de prueba y que se va a definir usando el *framewrok* NUnit. Para esto se debe:
 - 1. Decir que se va a usar el Framework NUNit con la instrucción:

using NUnit.Framework;

2. Usar el atributo TextFixture para indicar que la clase es una clase de prueba, esto se hace con la instrucción:

[TestFixture]

- Escribir los métodos de prueba de la clase. Para cada método a escribir se debe:
 - 1. Usar el atributo Test para indicar que es un método de prueba. Esto se hace con la instrucción:

[Test]

2. Escribir el código del método de prueba, usando los métodos Asserts que sean necesarios. Por ejemplo, para probar el método TransferFunds(), el código del método

Práctica 6

de prueba va a crear una cuenta source que va a inicializar con un balance de 150 y una cuenta destination que va a inicializar con un balance de 100. Luego se invoca al método TransferFunds() desde la cuenta source para transferir una cantidad de 100 a la cuenta destination. Luego, para realizar la prueba se invoca al método Assert.AreEqual() para la cuenta source y para la cuenta destination, para verificar si después de ejecutar el método TransferFunds(), sus valores de balance reflejan los valores esperados, que son respectivamente 250 para la cuenta destination y 100 para la cuenta source.

 Una vez escrito el código de los métodos de la clase de prueba, este debe verse se la siguiente forma:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Bank
ł
   using NUnit.Framework;
   [TestFixture]
    public class AccountTest
        [Test]
        public void DepositTest()
        {
            Account source = new Account();
           source.Deposit(200.00F);
           Assert.AreEqual(200.00F, source.Balance);
        }
        [Test]
        public void WithdrawTest()
        {
            Account source = new Account();
           source.Deposit(200.00F);
           source.Withdraw(50.00F);
           Assert.AreEqual(150.00F, source.Balance);
        }
        [Test]
        public void TransferFundsTest()
            Account source = new Account();
            source.Deposit(200.00F);
            Account destination = new Account();
            destination.Deposit(150.00F);
            source.TransferFunds(destination, 100.00F);
            Assert.AreEqual(250.00F, destination.Balance);
            Assert.AreEqual(100.00F, source.Balance);
        }
```

- Guardar los cambios realizados seleccionando la opción Archivo | Guardar Account.cs o bien Archivo | Guardar todo. También se puede hacer presionando los iconos asociados a estas opciones 🗔 🗊.
- Compilar el proyecto seleccionando el menú Generar | Generar Solución. Si el proyecto no compila debido a algún error arreglarlo y volver a ejecutar este paso hasta que la compilación y la generación de la solución sea exitosa.

5. Ejecutar los métodos de la clase de prueba AccountTest usando NUnit

 Iniciar la aplicación NUnit. Para esto se debe ir al menú de *Inicio* de Windows, luego en "*Todos los programas*" seleccionar el submenú *NUnit* 2.4.8 y allí hacer clic sobre el programa *NUnit GUI (.NET 2.0)*. Entonces aparecerá la interfaz gráfica de NUnit, como se muestra a continuación:

_						lasion. El el biereste ne combila acistas a algar	
NU	Unit						
File	View	Test	Tools	Help			
Tests						Run Stop	
gories							
Cate							
						Even and Exhura Laste Met Pure Consele Out Consele Even Lase	
Unlo	aded				Te		

- Cargar el archivo .dll del proyecto en NUnit seleccionando File | Open Project... y aparecerá una ventana de diálogo donde se debe indicae el archivo .dll resultante de la generación del proyecto, en este caso el archivo Bank.dll², y se pulsar el botón *Abrir*, como se muestra a continuación:

² Por defecto, la ruta completa del archivo Bank.dll sería C:\Documents and Settings\Administrador\Mis documentos\Visual Studio 2008\Projects\Bank\Bank\bin\Release\Bank.dll

Open Project		<u> ? ×</u>
Buscar en:	🔁 Release 💽 🔶 🖻 🗥	
) Recent	S Bank1.dll Bank.dll nunit.framework.dll	
C Escritorio		
Dis documentos		
Mi PC		
S		
Mis sitios de red	Nombre: Abr	ir
	Tipo: Projects & Assemblies(*.nunit,*.dll,*.exe) Cano	elar

- Entonces en la parte izquierda de NUnit aparecerá la estructura de la clase de prueba, que en este caso es la clase AccountTest, que contiene los métodos de prueba DepositTest, TransferFundsTest y WithdrawTest. En la parte derecha se encuentran los botones para ejecutar y parar las pruebas y un área para desplegar los resultados, formada por un conjunto de pestañas para mostrar los resultados de la ejecución de las pruebas, como se muestra a continuación:

Ny Bank1.dll - NUnit		
File View Project Test Tools Hel	p	
C:\Documents and Settings\Adminis Bank - AccountTest - DepositTest - TransferFundsTest - WithdrawTest	Tra	
Ready	Test Cases : 3	

- Ejecutar el caso de prueba. Para esto se pulsa el botón *Run* de NUnit y entonces se desplegará información textual y visual de la ejecución de las pruebas, como se muestra a continuación:



Ejercicio adicional

Definir el *Fixture* de la clase de prueba AccountTest para crear e inicializar para todos los métodos de prueba que contiene dos instancias de la clase Account, source y destination, e inicializar la cuenta source con un depósito de 200 y la cuenta destination con un depósito de 150.

Pasos a seguir:

 Como se ve en el código de la clase de prueba construida anteriormente, cada uno de los métodos de prueba crea e inicializa estructuras de manera redundante. Para evitar esto se debe definir e inicializar el *Fixture* del caso de prueba, a través de la utilización del atributo [SetUp] y el uso del un método Init() para ejecutar los métodos necesarios. Así el código optimizado de la clase de prueba quedaría de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Bank
ł
using NUnit.Framework;
[TestFixture]
public class AccountTest
ł
Account source;
Account destination;
[SetUp]
   public void Init()
    {
        source = new Account();
        source.Deposit(200.00F);
        destination = new Account();
        destination.Deposit(150.00F);
    }
    [Test]
    public void DepositTest()
    {
        Assert.AreEqual(200.00F, source.Balance);
    }
    [Test]
    public void WithdrawTest()
        source.Withdraw(50.00F);
        Assert.AreEqual(150.00F, source.Balance);
    }
    [Test]
    public void TransferFundsTest()
    {
        source.TransferFunds(destination, 100.00F);
        Assert.AreEqual(250.00F, destination.Balance);
```

```
Assert.AreEqual(100.00F, source.Balance);
}
}
```

- 2. Después de modificar el código de la clase de prueba, compilar y generar de nuevo el proyecto.
- 3. Guardar las modificaciones realizadas.
- 4. Ejecutar el caso de prueba. Para ello, en NUnit seleccionar en el menú File | Reload Project y ejecutar la pruebas pulsando el botón *Run*.

Referencias Bibliográficas

- Nunit.Org. *Nunit Quick Start*. Disponible en: http://www.nunit.org/index.php?p=quickStart&r=2.2

Práctica

7

Practicas opcionales:

- JUnit 4
- Pruebas de cobertura (Emma)
- Análisis estático (PMD)
- Pruebas en entornos Web (JMeter)
- Automatización de pruebas similares (JTestCase)

Objetivo: Según el tiempo restante y preferencias, se sugieren unas prácticas para cubrir otros conceptos de pruebas.

1. Probar ComplexNumber, con JUnit 4.

- Partiendo de la práctica 2 ya creada, desde el panel de proyectos:
 - Indicar a Netbeans que **ComplexNumber** es el proyecto principal
 - Borrar el archivo ComplexNumberTest.java. Seleccionar dicho archivo en el panel de proyecto y con el botón derecho seleccionar Delete.
- Regenerar el archivo ComplexNumberTest.java otra vez, seleccionando ComplexNumber.java y desde el menú seleccionar Tools | Create JUnit Test. En esta ocasión seleccionando JUnit 4, en las opciones de la ventana Create Test, comprobar que estén todas las opciones seleccionadas salvo Defult Method Bodies.

🗊 Create Tes	its			X		
<u>⊂</u> lass to Test:	complexnumberj4	I.Compl	e×Number			
Class Name: complexnumberj4.ComplexNumberTest						
Location:	Test Packages			-		
Code Gener	ation					
Method Ac	cess Levels	Genera	ated Code			
⊡ Put	blic	~	Te <u>s</u> t Initialize	r		
🔽 Pr <u>c</u>	ptected	◄	Test Finali <u>z</u> er			
🔽 Pac	c <u>k</u> age Private		Default <u>M</u> etho	od Bodies		
		Genera	ated Comment:	s		
		◄	Javadoc Com	ments		
		◄	Source Code	Hints		
	OK		Cancel	Help		

 Completar los en los métodos de prueba los cuerpos de las funciones tal y como vienen a continuación (o como se describió en la practica 2)

package complexnumberj4;
<pre>import org.junit.After;</pre>
<pre>import org.junit.AfterClass;</pre>
<pre>import org.junit.Before;</pre>
<pre>import org.junit.BeforeClass;</pre>
<pre>import org.junit.Test;</pre>
<pre>import static org.junit.Assert.*;</pre>
<pre>public class ComplexNumberTest {</pre>
private ComplexNumber cOneZero; private ComplexNumber cZeroOne; private ComplexNumber cOneOne;

```
public ComplexNumberTest() {
@BeforeClass
public static void setUpClass() throws Exception {
}
@AfterClass
public static void tearDownClass() throws Exception {
@Before
public void setUp() {
   cOneZero = new ComplexNumber(1, 0);
    cZeroOne = new ComplexNumber(0, 1);
    cOneOne = new ComplexNumber(1, 1);
}
@After
public void tearDown() {
ł
/**
 * Test of getReal method, of class ComplexNumber.
*/
@Test
public void testGetReal() {
    System.out.println("getReal");
    double expResult = 0.0;
   double result = cZeroOne.getReal();
   assertEquals(expResult, result);
}
/**
 * Test of getImaginary method, of class ComplexNumber.
 */
@Test
public void testGetImaginary() {
   System.out.println("getImaginary");
    double expResult = 0.0;
   double result = cOneZero.getImaginary();
   assertEquals(expResult, result);
}
/**
 * Test of add method, of class ComplexNumber.
 */
@Test
public void testAdd() {
    System.out.println("multiply");
    ComplexNumber result = cZeroOne.add(cOneZero);
    assertEquals(cOneOne, result);
}
/**
 * Test of multiply method, of class ComplexNumber.
*/
@Test
public void testMultiply() {
    System.out.println("multiply");
    ComplexNumber result = cZeroOne.multiply(cOneZero);
```
```
assertEquals(cZeroOne, result);
}
/**
 * Test of equals method, of class ComplexNumber.
 */
@Test
public void testEquals() {
 System.out.println("equals");
 boolean expResult = false;
 boolean result = cZeroOne.equals(cOneZero);
 assertEquals(expResult, result);
}
```

- Ejecutar los casos de prueba, seleccionando ComplexNumberTest.java en el panel de proyectos y en el menú Run | Test "ComplexNumber".

2. Pruebas de cobertura

- a. Instalar los *plug-ins* de cobertura ¹ en Netbeans.
 - Seleccionar Tools | Plugins.

NetBeans IDE 6.1			
File Edit View Navigate Source Refacto	or Build Run Profile Versioning 1	Fools Window Help	
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	🍯 🥙 🍸 🦉 🕨	Add to Palette	JMB
J 40 × J Files J Services St	tart Page 🗙 🙆 CalculatorTest.java	Internationalization	e Coverage - Project "Calculadora" × 🕢 🖌 💌 🗖
		NetBeaps Platforms	
E- Carce Packages		Libraries	S
Calculator, java		Component Libraries	2
Main.java	Welcome to NetBear	Servers	My NetBeans
🕀 🧊 Test Packages		Templates	
🗈 📻 calculadora		DTDs and XML Schemas	NetBeans IDE 6.5 Release 21/10/08
🗄 🔂 Load Generator Scripts 💌		Palette 🕨 🕨	availability of NetBeans IDE 6.5 Release
		Plugins	Candidate! Download Ne
∫ Calculator.java - Navigator			O Screencast: NetBeans ID 20/10/08
Members View		Options	This video shows how to debug server pages
E-🕎 Calculator			using the JavaScript Debugger in NetBeans
add(double number1, dou			IDE 0.5.
	<no project<="" recent="" td=""><td>></td><td>• Take Part in the NetBean 20/10/08</td></no>	>	• Take Part in the NetBean 20/10/08
			The NetBeans Project turns 10 and the celebration kicks off today! Be a part of the
Dutwit	Lillinë Toot Poouëo		X LBMD Output: found 8 violations
Na kest passed 4 kests failed	Journe rest Nesures	-	Prind Output. Iound o violations
aculadora CalculatorTest FAIL	IFD	add	
testAdd FAILED (0.078 s)		subtract	runcime coverage data
e testSubtract FAILED (0.0 s	5)	multiply	
testMultiply FAILED (0.016	is)	divide	
testDivide FAILED (0.0 s)		1	
		1	*1
		4	
/		,	

Seleccionar Code Coverage y pulsar Install.

Reiva				<u>D</u> earch: [test
Install	Name	Category ₹	Source	
	Use System Desktop	Base IDE	କ୍ରିଭି	Code Coverage Plugin
	SipAgent	SIP	ß	Nutrie pluste
	Jellytools	Testing Tools	ß	p NetBeans Beta Plugin
	XTest Samples	Testing Tools	ß	Version: 2.2.1
	Accessibility Checker	Testing Tools	ß	Author: Mikhail Kondratvey (mikk@netheans.org)
	soapUI Web Service Testing	Testing Tools		Date: 25/03/08
	JemmySupport	Testing Tools	ß	Source: NetBeans Beta
	Interactive UI Gestures Collector	Testing Tools		Homepage: http://codecoverage.netbeans.org/
	Jemmy Module	Testing Tools	ß	
	NB JUnit IDE Integration	Testing Tools	ß	Blugin Description
	Java Text Copy Paste Module	Tools	କିଳି	right beschpabli
	RegExPlugin	Tools	କିଳି	The Code Coverage Plugin is a Netbeans 6.0 plugin that enhances the existing
	Code Coverage Plugin	Tools	β	Netbeans functionality with new code coverage features. The plugin works as a transparent additional service that colors all year files according to the unit tests coverage information. With code Coverage plugin enabled user continues to work with his/her project in the usual way built can easily view the test coverage of the project classes. The code coverage plugin will update the code coverage data and refresh editors markup every time a unit test is executed for the project. Currently the Java Application, Java Uibrary, Java Project with Existing Sources and Netbeans module projects are supported.

- Seleccionar el proyecto de la **Práctica 3** de **Container**, como proyecto principal. Para ello en el menú seleccionar en el panel de proyectos el proyecto *Container* y pulsando el botón derecho, en el menú de contexto seleccionar "*Set as MAin Project*".

¹ http://emma.sourceforge.net/



 Abrir los distintos ficheros que componen el proyecto y ver como el código cubierto por los casos de prueba sombreado en un verde suave.



- Ver el informe de cobertura generado seleccionando el menú de contexto (botón derecho del ratón) desde el panel de proyecto Container.

🇊 Container - NetBeans IDE 6.	1			_ 🗆 🗡
<u>File Edit View Navigate Sou</u>	rce Ref <u>a</u> ctor <u>B</u> uild <u>R</u> un <u>P</u> rofile Versionir	ng <u>T</u> ools <u>W</u> indow <u>H</u> elp		
1 🔁 🔁 🖶 📈	👆 📋 🦻 🦿 🔀	- T 🍞 🕨 🗔 - (186.1/305.8MB	
J ≪ JFiles JSer	va 🕃 test-data.xml 🗙 🙆 Container.jav	a × Code Coverage - Project "Containe	r" × 🖄 Warehousejava 🗙	
🖅 🍉 Calculadora 📃	Project: Container			
🕀 🍥 ComplexNumber	Project is covered			
E ComplexNumber J4	Total classes covered: 75% (3 (4)			
Container	Total lines covered: 85% (29 / 34)			
E-F container	Total packages covered: 100% (1 / 1)			
🔤 🚮 Containe				
- 🚳 Main, jav,	Package coverage			
- 🙆 Package.	Show only not covered packages			
- 🙆 Warehou	Fully-qualified Package Name	Classes	Lines	
	container	75% (3 / 4)	85% (29 / 34)	<u> </u>
E Containo				
🕀 🔂 Libraries				
🕴 🔁 Tack Libyavian 🗡				
LiA/arehouse iava - N 4 X				
Members View				
Warebouse Container				<u> </u>
→ ♦ Warehouse(double (Class coverage			
🧿 add(Package b) : bo	Show only not covered classes			
— 🥥 getPackages() : Iter	Fully-gualified Class Name		Lines	
	container.Container	100% (13 / 13)		<u> </u>
	container.Main	0% (0 / 2)		
	container.Warehouse	79% (11 / 14)		
▲				_
🗛 🔲 🐻 🗑 🏷 🤘				<u> </u>
Output	JUnit Test Results	PMD Output] HTTP Monitor	
All 10 tests passed.	rTech packed	getVolume		<u> </u>
E 🏠 container.MainTest	passed	clear		
🗄 💑 container.PackageT	'est passed	contains		
🗄 🏤 container.Warehou	seTest passed			
		getVolume		_1
		J		_

3. Análisis estático con PMD

- Instalar el *plug-in* de PDM si no se encuentra disponible. Igual que en el ejercicio anterior, en el menú seleccionar Tools | Plugins y comprobar que está instalado PMD.

🗊 Plugins						>
Updates	Available Plugins (88) Download	ed Installed (48)	Settings			
<u>R</u> eloa	d Catalog				Search:	l
Uninstall	Name	Category 🔻	Active			1
	Mercurial	Base IDE	0		PMD Deactivate	l
	IDE Platform	Base IDE	O		Version: 2.2.1	1
	CVS	Base IDE	O		Source: ond.nbm	1
	Java	Java	2			1
	Java Persistence	Java	<u> </u>		Rhumin Description	1
	Java Profiler	Java	<u> </u>		Plugin Description	1
	PMD	Java	<u> </u>		Runs the PMD source code scanner tool. Finds various problems in your code and	1
	Spring Beans	Java	e		reports them to you in a tabular output window.	1
	Ant	Java				1
	NetBeans Plugin Development	Java	O			1
	GUI Builder	Java				1
	Java Debugger	Java				1
	XML Tools Java Ext	Java				1
	Platform	Platform		•		1
Uninst	tall					
					<u>C</u> lose <u>H</u> elp	

- En el proyecto anterior (Container), ir seleccionando las clases y comprobar que Warehouse.java infringe una de las reglas de PMD. Para ello, seleccionar el panel de proyectos la clase y con el botón derecho en el menú de contexto seleccionar Tools | Run PMD.

🇊 Container - NetBeans IDB	6.1				
File Edit View Navigate S	Source Refactor Build	Run Profile Ver	rsioning Tools Window	Help	
1 🔁 🔁 🖶 🔰	5 🔓 📋 🍤	C <default co<="" th=""><th>nfig> 🗾 👕</th><th>1 🛛 👔</th><th>ҧ т 🕕 т 🛛 явлить нивы 🔕 🧔 🐔</th></default>	nfig> 🗾 👕	1 🛛 👔	ҧ т 🕕 т 🛛 явлить нивы 🔕 🧔 🐔
P 40 × ∫Files	Servicesva 🛛	🖉 🖉 Warehouse.java	a 🗙 🚳 ComplexNumber	.java 🗙 📑 .	Anagramsjava 🗙 🚺 🖬 🗖
- 🔐 🙆 Warehous	e java	n - I mail	72 - JI - 17 - 17	r 🖪 🖉	
🕀 💼 Test Packages	Open		CONTRIBUTORS B	E LTABLE	FOR ANY DIRECT. INDIRECT. INCIDEN
庄 🗋 Load Generator :	Out	Chrl+X	EXEMPLARY. OR	CONSEQUEN	ITTAL DAMAGES (INCLUDING, BUT NOT
🕀 💼 Libraries	Copy	Chita	UREMENT OF SUBS	TITUTE GO	OODS OR SERVICES: LOSS OF USE. DAT
🗄 💼 Test Libraries	Rocke	CELV	OR BUSINESS IN	TERRUPTIO	N) HOWEVER CAUSED AND ON ANY THEO
🗄 🌐 helloworld	Casada Ella	50	V. WHETHER IN C	ONTRACT.	STRICT LIABILITY, OR TORT (INCLUD
🗄 🍥 🎒 JFlap_NB	Complie File	F9	CE OR OTHERMISE	ARTSING	TN ANY WAY OUT OF THE USE OF THI
🗄 🍉 JTestCase	Run File	Shift+F6	EVEN IF ADVIS	ED OF THE	POSSIBILITY OF SUCH DAMAGE.
🗄 🍉 VideoClub	Debug "Warehouse.jav	a" Ctrl+Shift+F5			CONTRACTOR OF SOME DIMINUM
🗄 📆 Source Packages	Profile File				
🗄 📑 videoclub -			Game Annlicatio	n #/	
🙆 ClientVide	Add		pound inppirouoro.	** (
	Delete	Delete	tow anagrame u	· ·	
	Save As Template		.coy.anagramo.a	-,	
Navigator -	Circl Lineare	Alt I CZ	ow energy li	h NordLib	reru.
Members View	Find Osages	AIL+F7	aut Dimension:	o.worabib	stary,
🛛 🖃 😪 😪 🖓 🗠 🕞 🖓 🗠 🗠	Refactor	•	awt Point.		
🔶 Warehouse(dout	BeanInfo Editor		awt Toolkit.		
🥥 add(Package b) : "			awt.event.Acti	onListene	er:
🦳 🥥 getPackages() : 🗌	Reverse Engineer		awt.event.Wind	owListene	
	File Members	Ctrl+F12	k.swing.JFrame:		
	File Hierarchy	Alt+F12			
Output	Local History	•	tes	I PMD Out	tout: found 7 vi 🛛 🖶 🗙 🛛 HTTP Monitor
loca	Tools	Þ	Add to Favorites		Decommondation
com toy, apagrams ui, Apagra	Properties		Country Black To 1	and a chiffe and	Avoid unused imports such as 'iava, awt.event.Action
com.toy.anagrams.ui.Anagram	s [39]	UnusedImports	Create JUnit Tests	Ltri+Snift+U	Avoid unused imports such as 'java.awt.event.Windo
com.toy.anagrams.ui.Anagram	s [216]	UnusedFormalPara	Add to Palette		woid unused method parameters such as 'evt'.
com.toy.anagrams.ui.Anagram	s [220]	UnusedFormalPara	Rún PMD (ctrl+Alt+P	Avoid unused method parameters such as 'evt'.
com.toy.anagrams.ui.Anagram	s [231]	UnusedFormalPara	Apply Diff Patch		Avoid unused method parameters such as 'evt'.
com.toy.anagrams.ui.Anagram	s [235]	UnusedFormalPara	Internationalization		void unused method parameters such as 'evt'.
com.toy.anagrams.ui.Anagram	s [247]	UnusedFormalPara	and mademailed dom		Avoid unused method parameters such as 'evt'.
P					

- Se abrirá un ventana de PMD en la parte inferior se pueden analizar los errores reportados.



 Para habilitar/deshabilitar reglas o incorporar nuevos conjuntos de reglas, desde el menú seleccionar Tools | Options seleccionar Miscellaneous.
 Aparecerán tabuladores con las distintas herramientas. Seleccionando PMD podremos seleccionar las reglas que queremos que nos saque la herramienta y las

🇊 Opl	tions						×
000	S.		{ @ }		a W A S		
Ger	neral	Editor	Java Code	Fonts & Colors	Keymap	UML	Miscellaneous
Ant I	Appearance Enable <u>s</u> can Jse this butt Jse this butt Jse this butt and customiz	 Diff GU On to config on to specifier e their prop 	I Builder JavaS ure set of rules y additional rules erties.	sets	iler ToDo T.	asks Versioning Manage rule Manage rulesets	Visual Web
A	dvanced Opt	tions				ОК	Cancel

4. Pruebas en entornos Web (JMeter)

- Diseñado para pruebas funcionales de carga de recursos estáticos y dinámicos (ficheros, scripts, bases de datos, etc). en entornos Web, FTP, etc.
- Puede funcionar en modo programa (*stand-alone*) o integrado en Netbeans. Para test de carga se puede iniciar desde el directorio donde se haya descomprimido ejecutando el fichero: jmeter.bat.
- Con el botón derecho se pueden crear un plan de pruebas, simulando multitud de opciones número de usuarios, y tiempo que se quiere

Apache JMeter (2.3.1)		
<u>File Edit Run Options H</u> elp		0/0 🗆
Test Plan Thread Group WorkBench	Thread Group Name: Armando Camorra Comments: Action to be taken after a Sampler error © Continue O Stop Thread O Stop Test Thread Properties Number of Threads (users): 5 Ramp-Up Period (in seconds): 0 Loop Count: Forever 2 Scheduler	

- Una guía de ejercicios de prueba de carga completa se puede ejecutar la descrita en *Javapassion* por Sang Shin:

http://www.javapassion.com/javaperformance/#Load_Testing_with_JMeter

5. Automatización de pruebas similares (JTestCase)

- Descargarse el JTestCase y ejecutar el caso de prueba de la calculadora como ejemplo (en entorno línea de comandos o Netbeans).
- Mirar e intentar entender el fichero CalculatorTest.java y el fichero XML de pruebas (test-data.xml) con el conjunto de casos de prueba en XML.



Práctica

Pruebas unitarias de programas con acceso a Bases de Datos usando DBUnit

Objetivo: Crear y ejecutar casos de pruebas de clase escritas en Java que acceden a bases de datos en MySQL usando el marco de pruebas DBUnit.

La aplicación con la que se va a trabajar en esta práctica es una supuesta aplicación simplificada de Recursos Humanos. Específicamente se va a trabajar con la clase **Employee** (Empleado), cuyas instancias van a ser los empleados de la empresa con los siguientes atributos:

- private String employeeUid: contiene el número de identificación del empleado en la empresa
- private String firstName: contiene el nombre del empleado
- private String lastName: contiene el apellido del empleado
- private String startDate: contiene la fecha en la que comenzó trabajar el empleado en la empresa
- private String ssn: contiene el número de seguridad social de empleado

Los servicios que provee la clase Employee son los siguientes:

- public Employee(String number, String stDate, String Name, String securityNumber, String lastName): es el método creador de las instancias de empleado
- public String getEmployeeUid(): método que permite obtener el identificador del empleado
- public String getStartDate(): método que permite
- public String getFirstName(): método que permite obtener el nombre del empleado
- public String getSsn(): método que permite obtener el número de seguridad social del empleado

- public String getLastName (): método que permite obtener el apellido del empleado
- insertEmployee(): método que inserta al empleado en la tabla employees de la Base de Datos HR

Después de crear la clase en java en Netbeans se deben realizar las pruebas de acceso a la base de datos usando la herramienta DBUnit.

1. Instalar MySQL

Como se va a trabajar con una Base de Datos (BD) creada usando el Sistema Gestor de Bases de Datos MySQL, debe estar instalado en el ordenador. Si no está instalado, se deben seguir los siguientes pasos para su instalación:

- a) Descargar el MySQL Community Server de la dirección: http://dev.mysql.com/downloads/
- b) Ejecutar el archivo descargado y seguir las instrucciones de instalación y dejando todas las opciones por defecto.

2. Ejecutar MySQL

Para ejecutar el MySQL en el Menú de Inicio de Windows, en la opción "*Todos los programas*" seleccionar MYSQL, luego seleccionar MySQLServer 5.0 y finalmente seleccionar *MySQL Command Line Client*.





_		6	Mozilla Thunderbird	►					SQL. EI LIAS
dro	,	G	MySQL	Þ	Ē	MySQL Server 5.0 🔹 🕨	Hys.	MySQL (Command Line Client
ury		6	Nero	►	0	MySQL Administrator	0	MySQL N	Manual
	Internet	6	NetBeans	►	٢) MySQL Migration Toolkit	8	MySQL S	Server Instance Config Wizard
	Mozilla Firefox	6	Nokia PC Suite	►	\swarrow	MySQL Query Browser			
6	E-mail	6	Notepad++	►	0	MySQL System Tray Monitor			
9	Mozilla Thunder	6	NUnit 2.4.8	×					
77	Microsoft Offi	6	NX Client for Windows	►					
	2003	6	Oxford SuperLex	►					
X	Microsoft Offi	6	Picasa2	►					
	2003		PuTTY	≁					
	Microsoft Office	(m)	R	≁					
0=	2003	6	RapidMiner	►					
6111		6	Real	►					
	Notepad		Skype	►					
1		6	Sophos	►					
\sim	Adobe Acrobat	Contract	Spyware Doctor	≁					
174		(m)	Startup	≁					l ¶
Σ	WinEdt	(m)	STATGRAPHICS Plus 5.1	►					
		6	TomTom	►					En·la·venta
	NetBeans IDE 6	6	VideoLAN	►					configuració
Ā		(iii)	WinEdt	►					oonngaraon
- 🚨	Set Video mode	Contract	WinMerge	≁					pueden∙eje
(The second se	Zattoo	≁					I IIII
	All Programs 🕨	Y	Acrobat Distiller 8						I
		1	Adobe Acrobat 8 Standard						
		Ø	Internet Explorer						
🛃 Stai	rt 🛛 🕑 🧐 🃢	0	JFLAP						ne 🗀 C:\Java\JD 🗀 C:\

Entonces aparecerá la siguiente ventana donde se pueden ejecutar los comandos SQL en las Bases de Datos creadas con MySQL.



En la ventana se debe introducir el *password* del usuario *root*, colocado en la configuración de MySQL. Al introducirlo aparece la línea de comandos donde se pueden ejecutar las sentencias SQL para manipular las Bases de Datos en MySQL



3. Crear la Base de Datos HR

La Base de Datos HR contendrá las tablas de la aplicación de Recursos Humanos. Para crear la Base de Datos HR se debe ejecutar en le línea de comandos de MySQL la instrucción:

mysql>create database HR;

Una vez que la BD ha sido creada, para abrirla y poder crear y manipular sus tablas escribir la instrucción:

mysql>use HR;



4. Crear la tabla employees

La tabla employees almacenará toda la información de los empleados de la empresa. Para crearla, podemos hacer desde la línea de comandos de MySQL ejecutando la siguiente instrucción:

```
CREATE TABLE employees(
   employeeUid BIGINT,
   startDate VARCHAR(10),
   firstName VARCHAR(50),
   ssn VARCHAR(50),
   lastName VARCHAR(50),
   PRIMARY_KEY (employeeUid)
);
```

O bien con MySQL Query Browser con el editor de Tablas:

able Name: employee	s	Datah	ase: br	T	Commen	r -		_
olumns and Indices	Table Options Advanc	ed Option	ns			~ J		
Column Name	Datatype	NOTL	AUTO Flags		D	efault Value	Comment	
📍 employeeUid	義 BIGINT	1	🗸 🗹 UNSIGNED	🗹 ZEROI	FILL			
🗇 startDate	💫 VARCHAR(10)	<	BINARY					
🗇 firstName	💫 VARCHAR(50)	<	BINARY					
🗇 ssn	💫 VARCHAR(50)	<	BINARY					
🗇 lastName	💫 VARCHAR(50)	~	BINARY					
Indices Foreign Ke	ys Column Details							
	ys Column Details [-Index S	Settings-						_
Indices Foreign Ke	ys Column Details	Settings-	PRIMARY	I	ndex Columr employeeUic	ıs (Use D)rag'n'Drop)	
Indices Foreign Ke	ys Column Details Index S Index Index	Settings Name: Kind:	PRIMARY PRIMARY	×	ndex Columr employeeUic	ıs (Use D	Prag'n'Drop) + +	
Indices Foreign Ke	ys Column Details Index S Index Index Index	Settings Name: Kind: Type:	PRIMARY PRIMARY DEFAULT		ndex Columr employeeUid	ıs (Use D J	Prag'n'Drop) + -	
Indices Foreign Ke	ys Column Details Index S Index Index H –	Settings Name: Kind: Type:	PRIMARY PRIMARY DEFAULT		ndex Columr employeeUic	ış (Use D 1	Prag'n'Drop) + -	

Confirm Table	e Edit	×
?	Are you sure you want to execute the following SQL command to apply the changes to the table? CREATE TABLE 'hr'.'employees' ('employeeUid' BIGINT UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT, 'startDate' VARCHAR(10) NOT NULL, 'firstName' VARCHAR(50) NOT NULL, 'ssn' VARCHAR(50) NOT NULL, 'astName' VARCHAR(50) NOT NULL, 'lastName' Name' VARCHAR(50) NOT NULL,	
	Execute Cancel	

Una vez creada la tabla se puede ver su estructura ejecutando la instrucción:

mysql>desc employees;

Prácticas del curso Prueba de Software

Al ejecutarla se desplegará la estructura de la tabla employees, como se muestra a continuación:

🔜 MySQL Command	Line Client						
nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql> nysql>	employees;						
Field	Туре	Null	Кеу	Default	Extra	1	
employeeUid startDate firstName ssn lastName	varchar(10) varchar(10) varchar(50) varchar(50) varchar(50)	NO YES YES YES YES	PRI	NULL NULL NULL NULL			
5 rows in set (mysql)	(0.00 sec)						-

5. Instalar el controlador para acceder una base de datos en MySQL desde una aplicación en Java

Para poder acceder a la tabla employees desde una aplicación escrita en java se necesita el SDK de Java y el controlador MySQL. En Netbeans este controlador debe estar disponible como una biblioteca MYSQL JDBC del proyecto a crear en Netbeans. Si no está disponible los pasos para descargar e instalarlo en el ambiente Netbeans hay que ejecutar los siguientes pasos:

- a) Descargar el controlador de MySQL: http://dev.mysql.com/downloads/connector/j/5.1.html.
- b) Descomprimir el archivo en el directorio raíz
- c) Añadir la biblioteca (*library*) llamada MYSQL JDBC indicando el camino donde está el archivo .jar correspondiente a Netbeans
- d) Finalmente, añadir está biblioteca al proyecto en Netbeans

6. Crear un nuevo proyecto de tipo Java Application llamado PruebaDBUnit

Para ello, en Netbeans se debe seleccionar: File | New Project ...

- En la ventana emergente New Project se debe seleccionar la categoría Java y dentro de esta categoría seleccionar el tipo de proyecto "Java Application".
 Pulsar el botón "Next>".
- En la ventana emergente New Java Application escribir en el campo Project Name el valor PruebaDBUnit y asegurar que estén seleccionadas las dos opciones de la ventana.
- Pulsar el botón "Finish".

7. Crear la clase Employee con sus atributos y servicios

Para esto hay que realizar los siguientes pasos:

- a) Crear la clase Employee
 - En el proyecto PruebaDBUnit en la carpeta Source Packages seleccionar el paquete pruebadbuunit. Pulsar el botón derecho del ratón y seleccionar: New | Java Class....
 - En la ventana emergente *New Java Class*, escribir en el campo *Class Name* el valor *Employee* y pulsar el botón "*Finish*".
 - Entonces se abrirá la pestaña correspondiente al esqueleto de código de la clase Employee.
- b) Definir los atributos y métodos de la clase Employee
 - Completar la clase Employee con los atributos y los métodos descritos a continuación¹:

```
package pruebadbunit;
//~--- JDK imports ------
import java.sql.*;
public class Employee {
   private String employeeUid;
   private String firstName;
   private String lastName;
   private String ssn;
   private String startDate;
   public Employee (String number, String stDate, String Name, String securityNumber,
String lastName)
           throws Exception {
       if ((number == null) && (lastName == null)) {
            throw new IllegalArgumentException ("El número del empleado y el apellido no
pueden ser nulos");
        }
        this.employeeUid = number;
       this.startDate = stDate;
this.firstName = Name;
this.ssn = securityNumber;
this.lastName = lastName;
    }
    public String getEmployeeUid() {
       return this.employeeUid;
    1
    public String getStartDate() {
       return this.startDate;
    public String getFirstName() {
       return this.firstName;
    public String getSsn() {
```

¹ Nota: en el método InsertEmployee() se establece la conexión con la BD en MySQL y el procedimiento para hacer esto está documentado en el código correspondiente

```
return this.ssn;
    }
    public String getLastName() {
        return this.lastName;
    }
    public void InsertEmployee() throws Exception {
        // El método InsertEmployee() inserta el Empleado en la tabla employees de la BD
HR
        // Para esto se establece la conexión con la BD en MySQL, y se manda a ejecutar \,
la instrucción
        // INSERT correspondiente en la BD
        11
        // la variable con es del tipo java.sql.Connection y a través de ella se maneja
la conexión con la BD
        11
        Connection con = null;
        Statement stmt = null;
        String
                  strSentence;
        // En la variable strSentence se almacena la instrucción SQL INSERT que se va a
mandar a ejecutar
        //
        strSentence = "INSERT INTO employees (employeeUid, startDate, firstName, ssn,
lastName) VALUES (\""
                      + this.getEmployeeUid() + "\",\"" + this.getStartDate() + "\",\""
+ this.getFirstName() + "\",\""
                       + this.getSsn() + "\", \"" + this.getLastName() + "\")";
        try {
            // En este bloque try se establece una conexión con la BD
            // En el String url se define la variable url de JDBC, que es la dirección
para conectarse a la BD HR
            11
            String url = "jdbc:mysql:///HR";
            // En la siguiente insrucción se define el contrlador que debe cargarse para
manejar la conexión
            // con la BD, que enste caso es el org.gjt.mm.mysql.Driver
            11
            Class.forName("com.mysql.jdbc.Driver");
            // Class.forName("org.gjt.mm.mysql.Driver");
            // En la siguiente línea se intenta realizar una conexión con la BD,
asignándola a la variable con
            // y en esta se especifica la variable URL, y el login y la password para
conectarse a la BD
            11
            con = DriverManager.getConnection(url, "root", "password");
            // Una vez establecida la conexióncon la BD, se pueden ejecutar
instrucciones SQL en ella usando
            // la conexión // Para ejecutar una instrucción SQL en la BD primero se crea una variable
que se usará para ejecutar
            // la instrucción, que en nuestro caso va a ser la variable stmt
// Se ejecuta la instrucción con.createStatement() que crea y abre un canal
de comunicación medante
            // el que se pueden ejecutar las consultas
            11
            stmt = con.createStatement();
            // Una vez abierto el canal de comunicación entre la aplicación y la BD se
manda a ejecutar la
            // instrucción SQL, que en nuestro caso está almacenada en el String
strSentence
            // usando la instrucción stmt.executeUpdate(strSentence)
            //
            stmt.executeUpdate(strSentence);
            System.out.println("Los valores se añadieron a la BD");
          catch (SQLException e) {
```

e.pri	ntStackTrace();
} finally	7 {
if (c	con != null) {
t	ry {
	<pre>con.close();</pre>
} } } }	<pre>// Una vez ejecutada la consulta se cierra la conexión //abierta con la instrucción stmt.close() // stmt.close(); catch (Exception e) { System.out.println(e.getMessage());</pre>

- Pulsar *<Crtl+S>* para salvar el trabajo realizado hasta ahora
- Construir el proyecto para verificar que no tiene errores. Esto se hace seleccionando en el menú: Build | Build Main Project.
- En la ventana *Output* se podrá ver el resultado de la acción *Build*, la cual debería ser exitosa. De lo contrario se debe revisar el código en busca de los errores indicados en la ventana *Output*, corregirlos y volver a construir el proyecto.

8. Crear la clase de pruebas DBEmployeeTest para probar el código de la clase Employee que se conecta con la Base de Datos usando DBUnit

Primero se debe añadir DBUnit como una de las bibliotecas de pruebas a usar en las clases de prueba. Para ello se debe hacer lo siguiente:

- a) Descargar DBUnit de la dirección: http://www.dbunit.org/
- b) Añadir DBUnit como una biblioteca en Netbeans que se añadirá a las librerías de pruebas del proyecto que se está desarrollando

📦 Add JAR/Folder			×
Look in: 🛅 d	lbunit-2.3.0	💌 🤌 📂 📰	
dbunit-2.3	.0. jar	 Use <u>Relative Path:</u> Use <u>Absolute Path:</u> 	
File <u>n</u> ame:	dbunit-2.3.0.jar	Open	
Files of <u>t</u> ype:	Classpath Entry (folder, ZIP or JAR file)	▼ Cancel	

Para probar código que accede a Bases de Datos usando la herramienta DBUnit, el procedimiento general a seguir es el siguiente:

- a) Se define la clase de prueba como una subclase de la clase DBTestCase, que a su vez es una subclase de la clase TestCase de JUnit
- b) Para que la clase de prueba tenga acceso a la BD de pruebas se usa un objeto que implementa la interface IDataBaseTester. Usando este objeto se puede hacer la conexión con la Base de Datos, usando las propiedades del sistema, que están en la clase System. Todo esto se define en el método constructor de la clase de prueba a construir.
- c) Se debe construir un objeto que implemente la interface IDataSet, y que contiene un *dataset* que describe el contenido de la Base de Datos con la que va a ser inicializada al comienzo de la ejecución de las pruebas. Para hacer esto se usa la clase FlatXmlDataSet, que provee funcionalidades para construir un dataset a partir de un archivo que contenga un documento XML que describa la estructura y contenido de la BD. La inicialización del contenido de la Base de Datos se hace en el método setUp().

Ahora bien, como el método con acceso a la BD que se va a probar ejecuta una instrucción INSERT en la BD, implica que se va a alterar el contenido de la BD y que por lo tanto lo que se quiere ver en la prueba es que el estado de la Base de Datos haya cambiado de la manera esperada, verificando que se insertó el registro en la BD de manera correcta, es decir, que antes de ejecutar la instrucción INSERT el registro no estaba en la BD y después de ejecutarla se almacenó de manera correcta. Para esto, el método que va a hacer la prueba debe hacer lo siguiente:

- a) Inicializar la BD con información conocida
- Ejecutar el método a probar usando los datos definidos para el caso de prueba
- c) Se define la información que se espera encontrar en la BD después de ejecutar la instrucción (estado esperado de la BD) y se obtiene la información después de ejecutar la instrucción (estado obtenido de la BD)
- d) Se comparan los estados esperados y obtenidos de la BD a través de los métodos asserts de DBUNit.

Considerando todo lo anterior, el código de la clase de prueba debe ser:

```
package pruebadbunit;
//~--- non-JDK imports ------
import org.dbunit.Assertion;
import org.dbunit.DBTestCase;
import org.dbunit.IDatabaseTester;
import org.dbunit.PropertiesBasedJdbcDatabaseTester;
import org.dbunit.database.IDatabaseConnection;
import org.dbunit.dataset.IDataSet;
```

```
import org.dbunit.dataset.xml.FlatXmlDataSet;
import org.dbunit.operation.DatabaseOperation;
import org.junit.Test;
import org.junit.internal.runners.TestClassRunner;
import org.junit.runner.RunWith;
//~--- JDK imports ------
import java.io.File;
import java.sql.SQLException;
@RunWith(TestClassRunner.class)
public class DBEmployeeTest extends DBTestCase {
   public DBEmployeeTest() {
       //super();
        /*
        * Método constructor de la clase DBEmployeeTest
        * donde se usa el método setProperty de la clase System
         * para que la clase System conozca los datos de la conexión con la BD
        * como son:
        * - EL driver de acceso a la BD, que es "org.gjt.mm.mysql.Driver"
            (también se puede colocar como driver el valor "com.mysql.jdbc.Driver" si
se usó
             el driver jdbc que ya trea instalado Netbeans en ss librerias)
        \star - URL de la conexión con la BD HR, que es jdbc:mysql:///HR
        \star - Login y password para acceder la BD
        * /
        System.setProperty(PropertiesBasedJdbcDatabaseTester.DBUNIT DRIVER CLASS,
"com.mysql.jdbc.Driver");
       System.setProperty(PropertiesBasedJdbcDatabaseTester.DBUNIT CONNECTION URL,
"jdbc:mysql:///HR");
        System.setProperty(PropertiesBasedJdbcDatabaseTester.DBUNIT USERNAME, "root");
        System.setProperty(PropertiesBasedJdbcDatabaseTester.DBUNIT PASSWORD,
"pruebas");
   }
    @Override
   protected DatabaseOperation getSetUpOperation() throws Exception {
       return DatabaseOperation.CLEAN INSERT;
    @Override
    protected void setUp() throws Exception {
        // super.setUp();
        final IDatabaseTester databaseTester = getDatabaseTester();
        assertNotNull("DatabaseTester is not set", databaseTester);
        databaseTester.setSetUpOperation(getSetUpOperation());
        databaseTester.setDataSet(getDataSet());
        databaseTester.onSetup();
    }
    @Override
    protected IDataSet getDataSet() throws Exception {
       return null; // new FileInputStream("C:/employee-hr-seed-1-registro.xml"));
    }
    @Test
    public void testInsertEmployee() throws Exception {
        11
        // Inicialización del caso de prueba
        // Se crea un Objeto de la clase Employee, que será almacenado en la BD con el
método a probar
        11
                         = "8";
        String idEmp
        String startDate = "2005-04-04";
               firstname = "Peter";
ssn = "000-90-0001";
        String
        String
        String lastname = "Gabriel";
Employee instance = new Employee(idEmp, startDate, firstname, ssn, lastname);
        try {
            // (1) Inicialización del contenido de la Base de Datos
            // Se inicializa la base de datos
```

En este caso al usar la instrucción TRUNCATE TABLE de la clase de DBUnit 11 // DatabaseOperation, se borran todas las filas existentes en la BD, dejándola vacía. // La clase <code>DatabaseOperation</code> permite realizar operaciones sobre la <code>BD</code> // para definir el contenido de la BD antes y después de ejecutar // los métodos de prueba 11 IDatabaseConnection dbConnection = this.getConnection(); dsInicialization = new FlatXmlDataSet(new IDataSet File("C:/employee-hr-seed-1-registro.xml")); DatabaseOperation.TRUNCATE TABLE.execute(dbConnection, dsInicialization); // (2) Ejecución del método a probar instance.InsertEmployee(); // (3) Obtención del contenido de la tabla Empleado como está en la BD. // Para esto se usa el método getDataSet de la Interfaz IDatabaseConnection // que accede a la BD y retorna un dataset que refleja su contenido // que se asigna a dsObtained 11 IDataSet dsObtained = getConnection().createDataSet(); // (4) Obtención del contenido esperado // Para esto se construye un dataset en base al contenido experado de la BD // después de ejecutar la prueba y que se encuentra definido como // un documento xml que se encuentra almacenado en el archivo // C:/employee-hr-seed-1-registro.xml, su estrcutura es la siguiente:
// <?xml version='1.0' encoding='UTF-8'?> // <dataset> // <employees employeeUid='8' // startDate='2005-04-04' // firstName='Peter' ssn='000-90-0001' // lastName='Gabriel' /> // </dataset> 11 // El dataset construido se almacena en dsExpected 11 IDataSet dsExpected = new FlatXmlDataSet(new File("C:/employee-hr-seed-1registro.xml")); // (5) Verificación // Se verifica que los contenidos de los datasets esperados y obtenidos // son iguales, que es lo que se espera // usando los métodos Asserts que provee DBUnit 11 Assertion.assertEquals(dsObtained, dsExpected); //} catch (SQLException e) { 11 e.printStackTrace(); // e.printStackTrace();
// fail("Error al ejecutar un caso de prueba"); } catch (Exception e) { e.printStackTrace(); fail("Error al ejecutar un caso de prueba"); } } }

Crear el fichero: c:\employee-hr-seed-1-registro.xml

9. Ejecutar la clase de pruebas DBEmployeeTest para probar el código de la clase Employee que se conecta con la Base de Datos usando DBUnit

En el menú, seleccionar Run | RunFile | Run "DBEmployeeTest.java".