

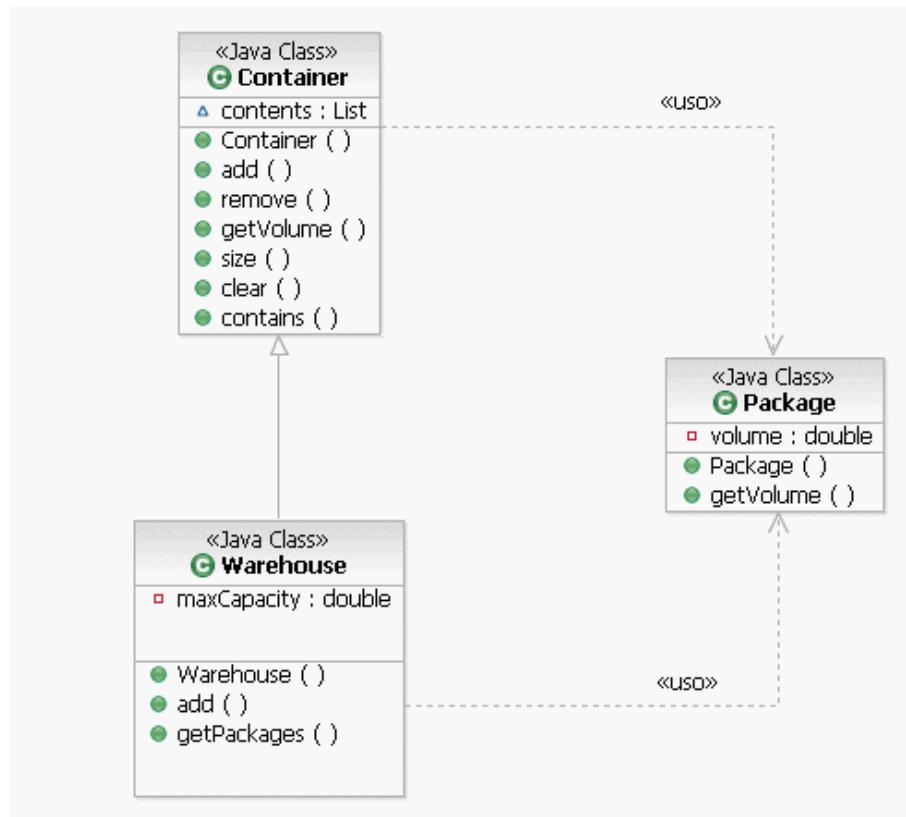
Práctica

4

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 los 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:
 - o Para la realización de las pruebas de las clases `Container` y `Warehouse` se va a trabajar con un *array* de 5 *packages* donde el

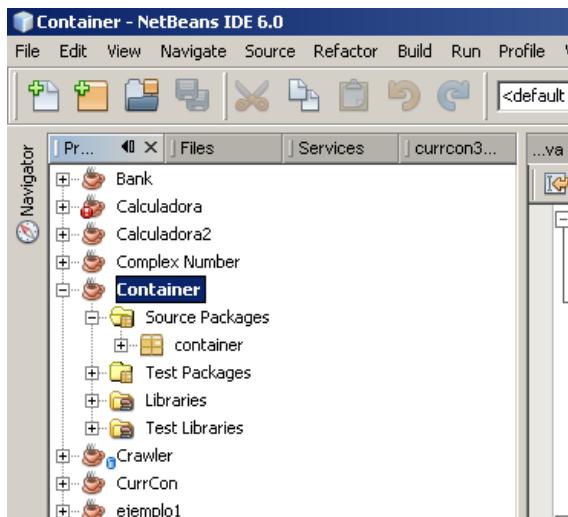
volumen de cada uno será el resultado de multiplicar 10 por el valor de la posición que ocupa en el array más 1 ($10 * (i + 1)$), para generar *packages* con volúmenes diferentes.

- 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 manera 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) {
            return super.add(b);
        }
        return false;
    }

    public Iterator<Package> getPackages() {
        List<Package> copyContents = new ArrayList<Package>(contents);
        return copyContents.iterator();
    }
}

```

```

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;
        }
    }
});

return copyContents.iterator();
}
}

```

- 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++) {
            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++) {
            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++) {
            container.clear();
            for (int j=0; j<i; j++) {
                container.add(b[j]);
            }
            for (int j=0; j<i; j++) {
                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++) {
                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++) {
            expResult = expResult + b[i].getVolume();
        }

        container.clear();
        for (int i=0; i<num_Packages_To_Test; i++) {
            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++) {
            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();
    }

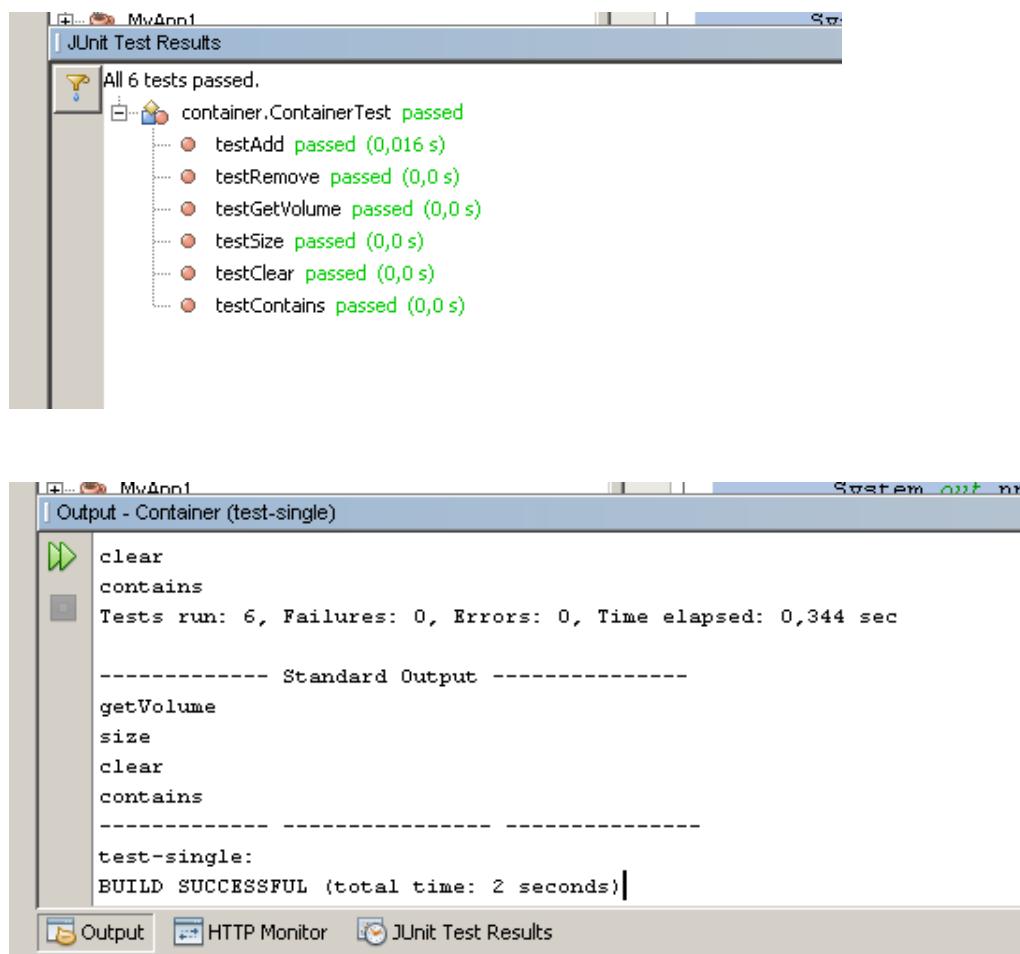
    /**
     * Test of contains method, of class Container.
     */
    public void testContains() {
        System.out.println("contains");
        container.clear();
        for (int i=0; i<num_Packages_To_Test; i++) {
            container.add(b[i]);
        }
        for (int i=0; i<num_Packages_To_Test-1;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

```

- Pulsar <**Ctrl+S**> 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:



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++) {
            if (i<Warehouse_Volume) {
                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++) {
            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
            added", !Warehouse.contains(b[i]));
        }
    }
}
```

```

        supposed to have been added!", Warehouse.contains(b[i]));

    }
    for (int i=Warehouse_Volume; i<num_Packages_To_Test; i++) {

        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++) {
        list.add(b[i]);
    }

    // First we add the Packages to the Warehouse in some random
order.
    for (int i=0; i<Warehouse_Volume; i++) {
        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) {
        Package Package = it.next();

        assertTrue("Packages are not returned by Warehouse.getPackages()
iterator in the correct order", b[count] == Package);

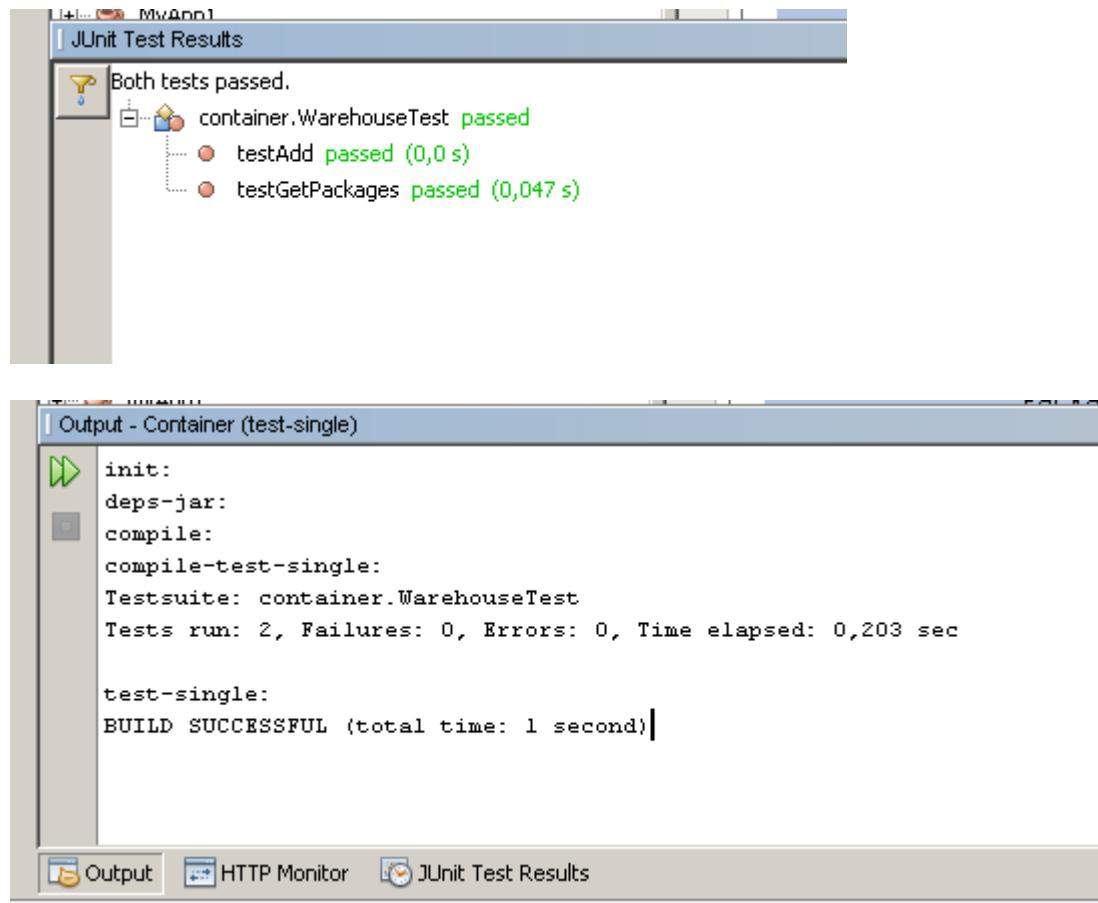
        if (b[count] != Package) {
            break;
        }
        count++;
    }
    assertEquals("Warehouse.getPackages() did not return all the
Packages", Warehouse_Volume, count);
}
}

```

- 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;

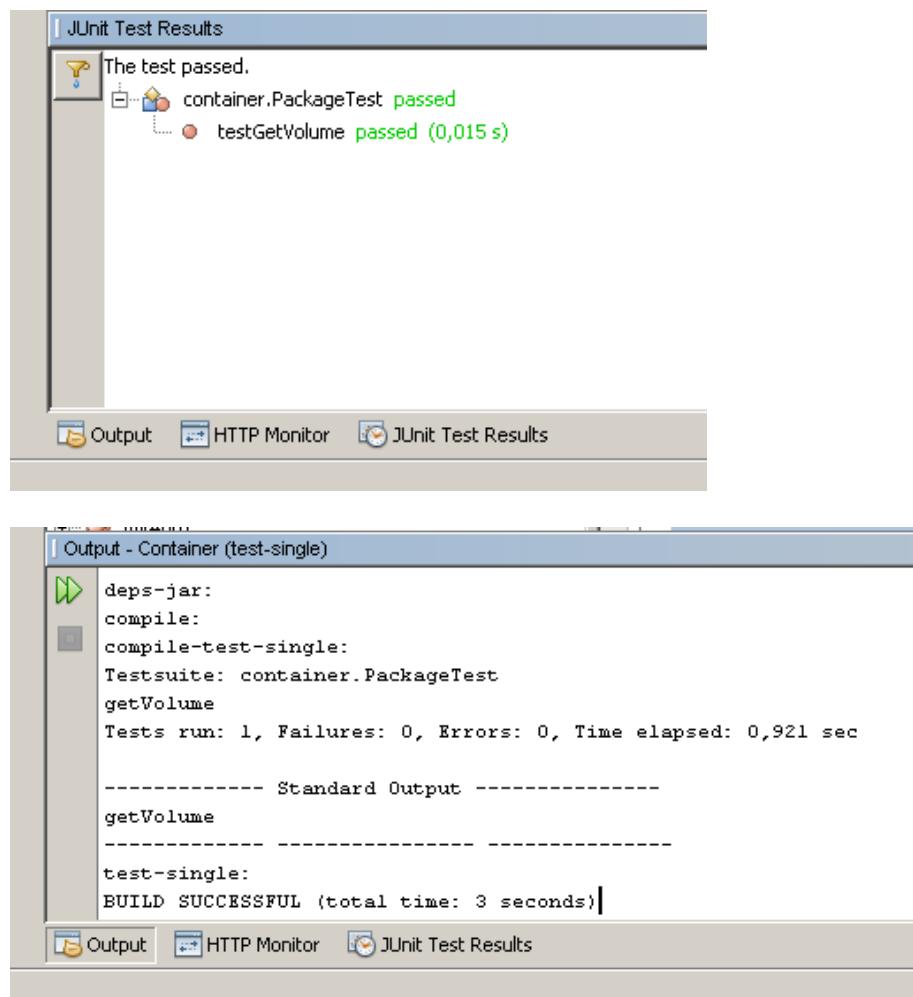
public class PackageTest extends TestCase {

    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



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);
    }

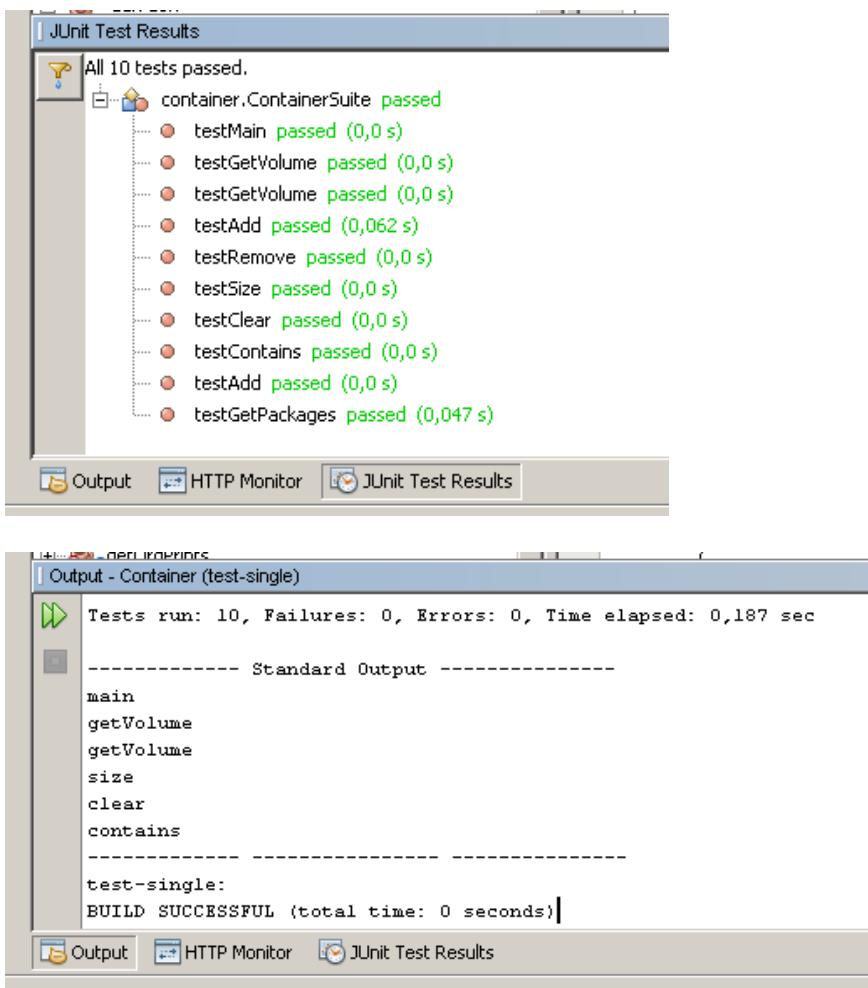
    public static Test suite() {
        TestSuite suite = new TestSuite("ContainerSuite");
        suite.addTest(container.MainTest.suite());
        suite.addTest(container.PackageTest.suite());
        suite.addTest(container.ContainerTest.suite());
        suite.addTest(container.WarehouseTest.suite());
        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



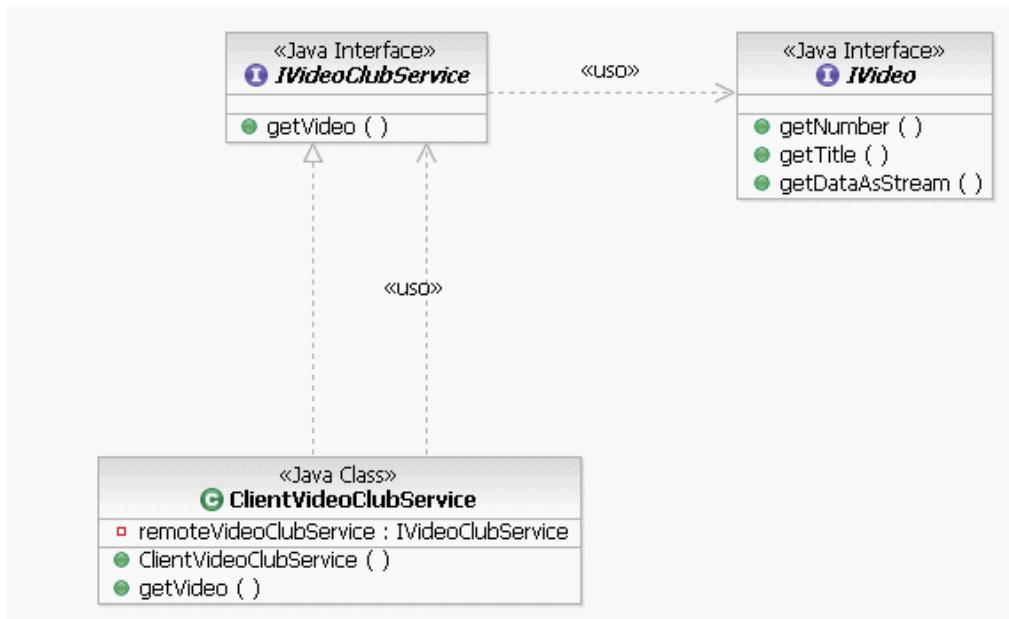
Práctica

5

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;

public class ClientVideoClubService implements IVideoClubService {
    private IVideoClubService remoteVideoClubService;
    public ClientVideoClubService(IVideoClubService
                                remoteVideoClubService) {
        if (remoteVideoClubService == null) {
            throw new IllegalArgumentException(
                "'remoteVideoClubService' must not be null");
```

```

        }
        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

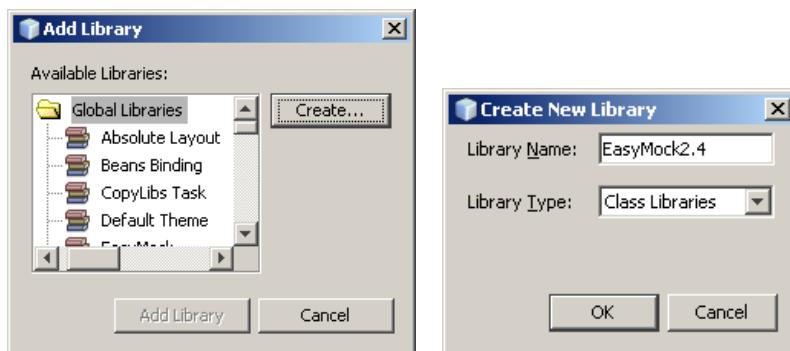
5. Crear y ejecutar un caso de prueba para la clase

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:

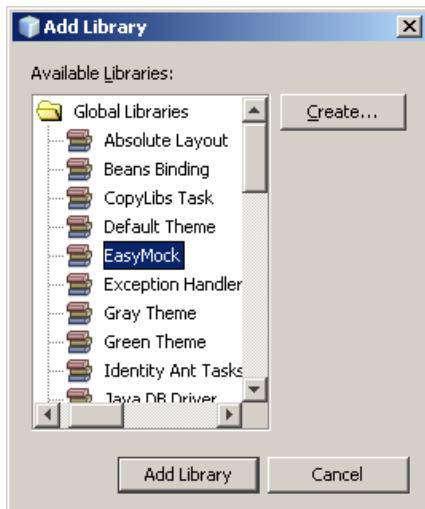
a) Añadir a la carpeta *TestLibraries* del proyecto *VideoClub* la biblioteca de EasyMock¹.

- 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).



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.

The screenshot shows two windows from an IDE:

- JUnit Test Results**: Displays the test results for the package `videoclub`. It shows "Both tests passed." and a tree view of the test suite:
 - `videoclub.ClientVideoClubServiceTest` passed
 - `testClientVideoClubService` passed (0,093 s)
 - `testGetVideo` passed (0,032 s)
- Output - VideoClub (test-single)**: Displays the build log:

```
init:  
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)
```