Pruebas Unitarias con el Patrón *Observer*

1 Introducción

Para realizar las pruebas unitarias del SCACV, hemos incluido en tres paquetes diferenciados las clases de prueba para los paquetes correspondientes del código producido: *ControlVelocidad*, *Monitorizacion* e *Interfaz*, según el diseño arquitectónico propuesto en el guión de esta práctica. Cada uno de los paquetes de pruebas contiene un *Test Suite* que inicia la ejecución de todos los casos de prueba (*Test Cases*) de ese paquete. En el paquete *Interfaz* se incluyeron los casos de prueba para las clases: `Interfaz`, `ListaObservadoresObservables`,`Observable`, `PanelBotones` y `Simulacion`, tal como se puede ver en la Figura 1.

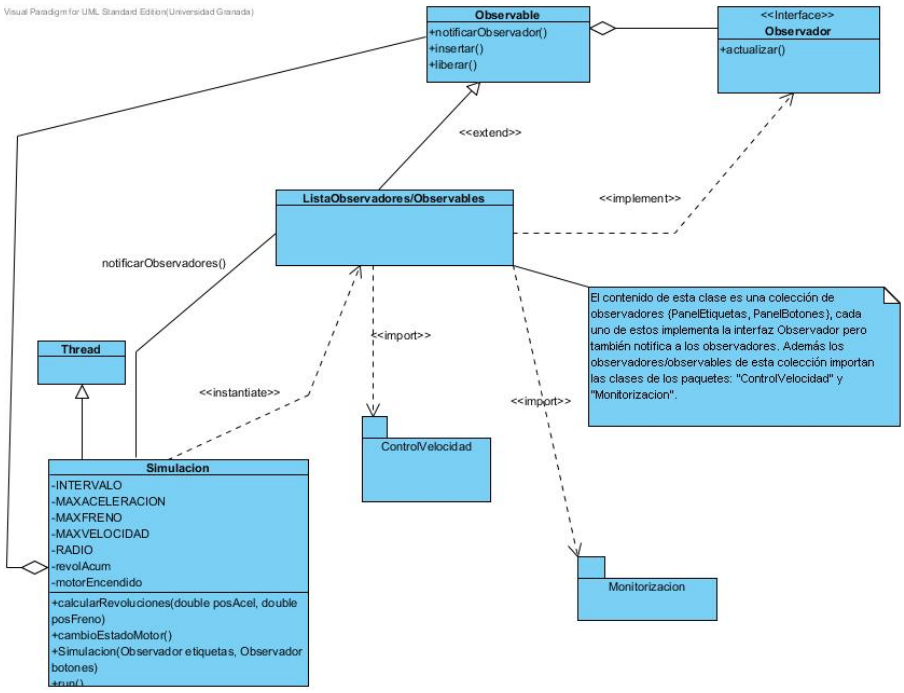

Figura 1: Arquitectura software del SCACV para realizar las pruebas unitarias.

No es necesario crear pruebas unitarias para la clase `PanelEtiquetas` ni para las clases que muestran indicadores en la interfaz de usuario, ya que su funcionamiento se basa en una modificación de la clase test para la propia `Interfaz`.

Lo anterior permite presentar los resultados obtenidos para los casos de prueba de las clases de código producido para el SCACV utilizando varios patrones de diseño de la siguiente manera:

1. Implementación de la prueba de un escuchador de eventos que implementa *TestListener* de `junit.framework.*` y sirve para desarrollar las pruebas unitarias del patrón Observador, utilizado en el paquete *ControlVelocidad*.

2. En el paquete *ControlVelocidad* se han de incluir los casos de prueba de las clases más relevantes, incluyendo la prueba unitaria de la clase *Pedal* que abstrae los detalles de clases (*acelerador*, *freno*) que contienen los métodos necesarios para interactuar con los dispositivos:

```
public interface Pedal{
  public double leerEstado();
  public void soltar();
  public void actualizar();
}
```

3. No se implementaron casos de prueba para clases del tipo *Almacen*, ya que sus métodos son de tipo *getter()* y *setter()* y sus pruebas son triviales.

4. Por último, en el paquete de *Monitorización* se ha de programar los casos de prueba de todas las clases que manejan los controles del vehículo.

1.1 Prueba del patrón *Observador*

El test unitario del patrón *Observador* se ha de realizar comprobando que cada vez que se dan las condiciones durante la ejecución de la aplicación, el *observable notifica* a todos los *observadores* de su lista y como consecuencia de esto cada uno de sus observadores ejecuta el método *actualizar()*.

Para programar tests con JUnit de determinados patrones de diseño cuya prueba consiste en verificar si durante su ejecución ocurrieron todos eventos de una lista, podemos utilizar el *patrón de pruebas* denominado *Escuchador de Eventos*. Para lo cual hemos de hacer lo siguiente:

- Crearnos una clase *ObservadorTestListener* que implemente las interfaces:
  1. *TestListener* de Java y
  2. la interfaz *Observador* del patrón de diseño.

La implementación de la interfaz (1) permite escuchar *eventos textuales*: *inicio de test*, *fin de test*, *error* o *fallo*, que ocurren como resultado de la ejecución de un *Test* en la aplicación. La implementación de la interfaz (2) convierte a la clase en un *Observador* y, por tanto, responde a la llamada al método *actualizar()* del patrón Observador (Figura 2).

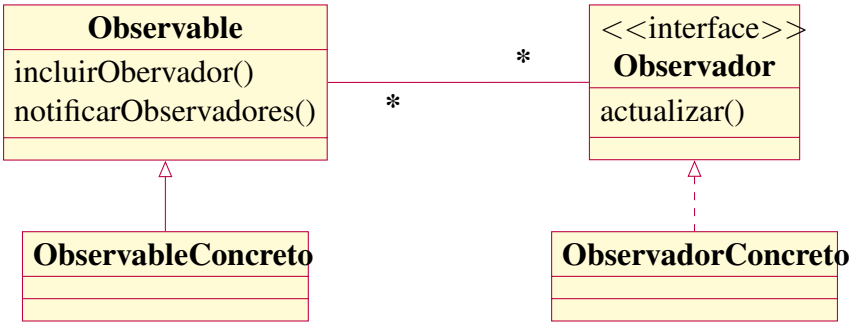

Figura 2: Diagrama de clases del patrón Observable–observador

El componente "Sujeto" de la aplicación que produce los *eventos textuales* (un objeto *TestResult*) llama al método `addTextListener()` para registrar en una lista interna a su *escuchador* ficticio asociado (*ObservadorTestListener*). Cuando cambia el texto del componente Sujeto se llama a `textValueChanged()` de su objeto escuchador automáticamente.

1.1.1 Programación de la prueba del patrón Observador

Se ha de crear una clase `ListaObservadoresObservablesTest` que contiene todos los casos de prueba del observable:

1. `testNotificarObservadores()`,
2. `testInsertaObservador()`,
3. `testEliminaObservador()` y
4. el método `setUp()`, constructor de la *fixture*, necesario para programar la clase de test.

En la clase `ListaObservadoresObservablesTest` se crea un objeto `ObservadorTestListener` y un objeto `TestResult`. Al objeto `TestResult` se le añade el objeto `ObservadorTestListener` como su escuchador:

```
1  import static org.junit.Assert.*;
2  import junit.framework.TestCase;
3  import junit.framework.TestListener;
4  import junit.framework.TestResult;
5  import org.junit.Before;
6  import org.junit.Test;
7
8  /**Prueba de un observable utilizando un escuchador de eventos ficticio
9   * que implementa la interfaz TestListener.*/
10  *    ....*/
11         Observable observable;
12         ObservadorTestListener observador;
13         TestResult res;
14         @Before
15         public void setUp() throws Exception {
16                 observable = new ListaObservadoresObservables();
17                 observador = new ObservadorTestListener();
18                 /* Crear un objeto TestResult asignandole el observador
19                  * como listener. Se le indica que va a comenzar un test para
20                  * que almacene el evento startTest.*/
21                 res = ref_objeto_TestCase.createResult();
22                 res.addListener((TestListener)observador);
23                 res.startTest(this);
24         }
25   ... resto de metodos tests
```

Posteriormente, al escribir el método de prueba `testNotificarObservadores()` (Figura 3) se crea una lista con los objetos que se espera que contenga la lista de eventos, es decir: un objeto correspondiente al inicio del test, seguido de otro correspondiente a la llamada al método `actualizar()` del observador.

Dentro del paquete *Simulacion* de la aplicación SCACV, que se ha proporcionado con el guión de la práctica, programamos la clase *ObservadorTestListener* (Figura 4), como escuchador de los eventos necesarios para realizar la pruebas unitarias del patrón de diseño *Observador* de la mencionada aplicación.

```java
@Test
public void testNotificarObservadores() {
 testAniadirObservador();
 observable.notificarObservadores();
 List<List<Object>>esperados= new ArrayList<List<Object>>();
 esperados.add(observador.nuevoObjetoEventoInicioTest(this));
 esperados.add(observador.nuevoObjetoEventoObservable());
 assertEquals(esperados, observador.listaEventos() );
}
```

Figura 3: Implementación de *testNotificarObservadores()*.

Esta clase cuenta con una lista de eventos que se va completando con objetos que se generan cada vez que se recibe un evento y que identifican al evento que los produce utilizando, entre otros elementos, una cadena de caracteres. La clase `ObservadorTestListener` implementa cada uno de los métodos de la interfaz `TestListener`, que se invocan cuando se produce un evento correspondiente a un *inicio de test*, *fin de test*, *error* o *fallo*, durante la ejecución de la aplicación SCACV. Para cada uno de estos eventos se inserta un objeto diferente en la lista.

```java
public class ObservadorTestListener
          implements TestListener, \emph{\emph{Observador}} {
 private List<List<Object>> eventos;
 public ObservadorTestListener() {   //Constructor
     eventos = new ArrayList<List<Object>>();
 }
 @Override
 public boolean actualizar() {
     eventos.add(nuevoObjetoEventoObservable());
     return true;
 }
 @Override
 public void startTest(Test test) {
     eventos.add(nuevoObjetoEventoInicioTest(test));
 } ...
 public List<Object> nuevoObjetoEventoObservable() {
     return Arrays.asList(new Object[ ]{"actualizar"});
 }
 public List<Object> nuevoObjetoEventoInicioTest(Test test){
     return Arrays.asList(new Object[]{"startTest",test});
 }...
 //Devuelve la lista de eventos acumulados
 public List<List<Object>> listaEventos() {
    return eventos;}
 }
}
```

Figura 4: Implementación de la clase *ObservadorTestListener*.

El método `actualizar` de la interfaz *Observador* (Figura 4) también está implementado de forma que se inserta un objeto identificativo en la lista de eventos cada vez que es invocado durante la ejecución del test en `notificarObservadores()` de la clase `ListaObservadoresObservables`, ver Figura 3.

1.1.2 En qué consiste la prueba unitaria del Observador

Se programa comparando el contenido de la lista de eventos de `testNotificarObservadores()` con el de la lista `eventos` del objeto escuchador `ObservadorTestListener`. Si ambas listas coinciden, entonces concluimos que el patrón *Observador* de nuestra aplicación ha pasado la prueba con éxito.

Créditos

- Unit Testing with JUnit - Tutorial (2016) `http://www.vogella.com/tutorials/JUnit/article.html`
- Test Listeners `https://github.com/powermock/powermock/wiki/testlisteners`
- Interface TestListener en Oracle: `https://docs.oracle.com/javase/8/docs/api/java/awt/event/TextListener.html` `http://www.it.uc3m.es/celeste/docencia/java/docs/api/java/awt/event/TextListener.html`
- Ejemplo de programación de un TestListener: `http://www.javadocexamples.com/java/awt/event/java.awt.event.TextListener.html`