Ejercicios de JUnit

Índice

1. Pruebas del gestor de filmotecas (2 puntos)

Vamos a implementar un conjunto de casos de prueba para probar `IPeliculaDAO`. En nuestro caso, la implementación que someteremos a nuestras pruebas será `MemoryPeliculaDAO`. Vamos a implementar en JUnit los siguientes casos de prueba:

Método a probar	Entrada	Salida esperada
`addPelicula( Pelicula p)`	Lista actual: vacía `p.id`: 1 `p.titulo`: "El Resplandor"	Lista: {1, "El Resplandor"}
`addPelicula( Pelicula p)`	Lista actual: {1, "El Resplandor"} `p.id`: 3 `p.titulo`: "Casablanca"	Lista: {1, "El Resplandor"}, {3, "Casablanca"}
`addPelicula( Pelicula p)`	Lista actual: {1, "El Resplandor"} `p.id`: 1 `p.titulo`: "El Resplandor"	`DAOException`
`addPelicula( Pelicula p)`	Lista actual: {3, "Casablanca"} `p.id`: 4 `p.titulo`: null	`DAOException`
`addPelicula( Pelicula p)`	Lista actual: {3, "Casablanca"} `p.id`: 5 `p.titulo`: ""	`DAOException`
`addPelicula( Pelicula p)`	Lista actual: {3, "Casablanca"} `p`: null	`DAOException`
`delPelicula(int i)`	Lista actual: vacía `i`: 1	`DAOException`
`delPelicula(int i)`	Lista actual: {1, "El Resplandor"} `i`: 1	Lista: vacía
`delPelicula(int i)`	Lista actual: {3, "Casablanca"} `i`: 1	`DAOException`
`delPelicula(int i)`	Lista actual: {1, "El Resplandor"}, {3, "Casablanca"} `i`: 1	Lista: {3, "Casablanca"}
`delPelicula(int i)`	Lista actual: {1, "El Resplandor"}, {3, "Casablanca"} `i`: 2	`DAOException`

Utiliza *fixtures* para inicializar los distintos tipos de entradas y salidas esperadas que vamos a probar. Fíjate en que en muchas de las pruebas se repiten las estructuras de datos que se tienen como entrada y que se esperan como salida. Podemos aprovechar esto para simplificar nuestro código, dejando dichas estructuras predefinidas como *fixtures*.

Podría ser de ayuda no probar directamente la clase `MemoryPeliculaDAO`, sino crear una subclase a la que añadamos un método que nos permita establecer el contenido de la lista actual de películas. Esta clase siempre deberá estar en el directorio de fuentes de pruebas, ya que será una clase que sólo servirá para realizar estas pruebas, y en ella nunca deberemos sobrescribir los métodos a probar, ya que lo que nos interesa es probar los métodos reales, no los que creemos como ayuda para las pruebas.

2. Desarrollo guiado por pruebas (1 punto)

Vamos a utilizar la metodología de desarrollo guiado por pruebas para implementar las operaciones `calculaSalarioBruto` y `calculaSalarioNeto` especificadas en el tema de teoría. Tenemos en las plantillas de la sesión un proyecto `lja-junit` en el que podemos encontrar la aplicación parcialmente implementada. Tenemos una clase `EmpleadoBR` con el esqueleto de los métodos anteriores, que deberán ser implementados. Se pide:

a) Tenemos implementada una clase `EmpleadoBO` con las operaciones de negocio de los empleados. En este caso únicamente tenemos el método `getSalarioBruto` que nos proporciona el salario bruto de un empleado dado su identificador. Para ello primero utiliza el DAO para obtener los datos del empleado de la base de datos, y después utiliza las reglas de negocio de los empleados, contenidas en `EmpleadoBR`, para calcular el salario a partir de dichos datos. Tenemos también definida una prueba para esta clase en `EmpleadoBOTest`. El DAO JDBC para acceso a los datos de los empleados en una base de datos no está implementado, pero tenemos un *mock* que lo sustituye en las pruebas (`MockEmpleadoDAO`), así que eso no supone ningún problema para poder probar el método. Sin embargo, nos falta implementar la regla de negocio necesaria, y como es evidente, si ejecutamos la prueba veremos que falla.

b) Implementar los casos de prueba especificados en el tema de teoría para la operación `calculaSalarioBruto`. Ejecutar dichos casos de prueba y comprobar que el test falla.

c) Implementar la operación `calculaSalarioBruto` con el código más sencillo posible que consiga que las pruebas tengan éxito. Tras implementarlo, comprobar que JUnit nos da luz verde. Si fuese necesario, refactorizar el código para dejarlo limpio y bien organizado, y comprobar que las pruebas sigan funcionando.

d) Ahora podríamos ejecutar las pruebas de `EmpleadoBO`. Al estar implementada la regla de negocio necesaria debería funcionar.

e) Repetir el proceso para la operación `calculaSalarioNeto`. Para obtener la información de los tramos de retención utilizaremos la clase `ProxyAeat`. El acceso al

servidor no está implementado todavía en esta clase, pero en lugar de implementarlo lo que haremos será crear un *mock* para poder ejecutar las pruebas. En el tema de teoría puedes encontrar ayuda para la implementación de este método y del *mock* necesario para las pruebas.

f) Implementar una suite de pruebas que agrupe tanto las pruebas de `EmpleadoBRTest` como las de `EmpleadoBOTest`. La suite se llamará `AllTests`, y se encontrará en el paquete `es.ua.jtech.lja.tienda`.