Informe Procesos Software Practica 1

Santiago Sañudo Martínez

Carlos Díez Manjón

# Introducción

La práctica consiste en realizar pruebas de aceptación y sistema, unitarias y de integración al código proporcionado en el enunciado de manera que se cumpla una cobertura máxima del código con las mismas, respectivamente la cobertura de estas pruebas superara el 95% para la capa de Vista con las pruebas de aceptación y sistema, para la capa de business con las pruebas unitarias, y para la capa DAO y business con las pruebas de integración.

La aplicación a testear consiste en una aplicación de 3 capas bastante sencilla en la que tendremos una BBDD(en nuestro caso empleamos un XML), una vista con JFrame y la lógica del producto, que se encarga de dar de alta, dar de baja, y consultar contribuyentes y los vehículos asociados a estos de un ayuntamiento, asi como poder calcular el precio del importe que debería de pagar cada contribuyente según sus vehículos.

Para comenzar la práctica tuvimos que realizar una serie de modificación al material proporcionado ya que podemos usar las librerías de mockito, fest y junit de manera dinámica, es decir, importándolas automáticamente con Maven independientemente del ordenador en donde estemos si las añadimos al fichero pom.xml del proyecto. Por tanto, no es necesario separar las pruebas en 3 proyectos y podríamos tenerlo todo en un único proyecto. Y para dejar claro de qué tipo son las pruebas hemos creado 3 clases en la carpeta de tests independientes para cada tipo.

Además, antes de comenzar tuvimos que solucionar algunos problemas de compilación ya que debido a algunos errores el programa no ejecutaba correctamente.

Una vez solucionados todos los problemas anteriores poseemos un proyecto el cual es capaz de ejecutar la aplicación sin ningún tipo de problema, y por tanto procedemos a realizar las pruebas:

# Aceptación y Sistema

En este tipo de pruebas realizamos un uso de la librería FEST para poder automatizar las pruebas empleando la interfaz de la aplicación.

Dado que la interfaz de la misma se encuentra incompleta hemos realizado tan solo los casos de uso referentes a la consulta del contribuyente cuando existe y cuando no existe. Mencionar que durante la elaboración encontramos errores en el código de manera que los resultados mostrados no eran correctos, ya que en el campo que debería mostrar el valor del precio a pagar la aplicación mostraba otra vez el nombre del contribuyente. Este error se situaba en el método rellenaContribuyente y se ha solucionado para conseguir que los tests pasaran sin problemas.

# Pruebas Unitarias

En este tipo de pruebas hemos empleado la librería mock, ya que es necesario obviar parte de la arquitectura de la aplicación en este caso la capa de modelo en la que se encuentra el DAO y el XML como tal, por tanto, podemos comprobar que la lógica de la capa business de la aplicación funciona independientemente de la implementación de la capa DAO.

Para ello hemos definido todos los elementos necesarios para cada test como son los contribuyentes que vamos a utilizar y los vehículos, así como el comportamiento de los mock de la capa DAO cuando se emplean en la capa business.

Las pruebas realizadas según los casos de uso han sido:

* U3.a Un contribuyente intenta darse de alta y se da de alta correctamente
* U3.b Un contribuyente intenta darse de alta y se da un error porque ya existe
* U4.a Un contribuyente intenta darse de baja y se da de baja correctamente
* U4.b Un contribuyente intenta darse de baja y se da un error porque tiene vehículos asociados
* U4.c Un contribuyente intenta darse de baja y se da un error porque no existe
* U5.a Un contribuyente intenta consultarse y el sistema muestra los datos correctamente porque el contribuyente existe
* U5.b Un contribuyente intenta consultarse y se da un error porque el contribuyente no existe
* U6.a Se intenta consultar el total de un contribuyente y como el contribuyente existe, retorna 239.8.
* U6.b Se intenta consultar el total de un contribuyente y como no existe, retorna un valor que indique fallo.
* U6.c Se intenta consultar el total de un contribuyente que no tiene vehículos y retorna 0.
* U7.a Un vehículo intenta darse de alta para un contribuyente y se da de alta correctamente
* U7.b Un vehículo intenta darse de alta para un contribuyente y se da un error porque no existe el contribuyente
* U7.c Un vehículo intenta darse de alta para un contribuyente y se da un error porque ya existe el vehículo
* U8.a Un vehículo intenta darse de baja para un contribuyente y se da de baja correctamente
* U8.b Un vehículo intenta darse de baja para un contribuyente y se da un error porque no existe el vehículo
* U8.c Un vehículo intenta darse de baja para un contribuyente y se da un error porque no existe el contribuyente
* U8.d Un vehículo intenta darse de baja para un contribuyente y se da un error porque el vehículo no pertenece al contribuyente
* U9.a Se intenta consultar un vehículo y el sistema muestra los datos correctamente porque el vehículo existe
* U9.b Se intenta consultar un vehículo y se da un error porque el vehículo no existe
* U10.a Intentamos actualizar un contribuyente cambiándole su segundo apellido y se actualizan los datos correctamente porque el contribuyente existe y se devuelve el contribuyente actualizado.
* U10.b Intentamos actualizar un contribuyente cambiándole su segundo apellido y se indica que ha habido un error porque no existe un contribuyente con dicho DNI y no se actualiza nada.
* U11.a Se intenta consultar el listado de contribuyentes cuando no hay ninguno y se devuelve una lista vacía
* U11.b Se intenta consultar el listado de contribuyentes cuando sólo hay uno y se devuelve una lista con un elemento
* U11.c Se intenta consultar el listado de contribuyentes cuando hay varios y se devuelve una lista con todos los contribuyentes

Para que todos estos tests se cumplieran tuvimos que solucionar errores en el código modificando los métodos de la clase GestionImpuestosCirculacion, los métodos modificados son:

Los métodos modificados han sido:

* altaContribuyente: ya que el método inicial no comprobaba si el contribuyente ya existía.
* bajaContribuyente: ya que el método no comprobaba que la lista de vehículos del contribuyente estuviese vacía antes de eliminarlo.
* totalContribuyente: la cuenta del total comenzaba desde 1 en vez de desde 0 y en el caso de que el contribuyente no tenga vehículos debería devolver un valor negativo.
* contribuyente: ya que la llamada que hacía era a eliminaContribuyente en vez de datosContribuyente.
* altaVehiculo: ya que no se comprobaba que existiese el contribuyente, ni si existía ya el vehículo en la lista de vehículos.
* actualizaContribuyente: método añadido para poder realizar una actualización de un contribuyente desde gestión.
* contribuyentes: método añadido que devuelve la lista de contribuyentes.
* actualizaVehiculo: método añadido para poder realizar una actualización de un vehículo desde gestión.

# Pruebas de Integración

Este tipo de pruebas consiste en hacer uso de todos los componentes de la aplicación, de manera que hagamos uso de la capa business y esta acceda al modelo, la capa DAO llegando a modificar sin problemas el XML. Con este tipo de pruebas observamos un comportamiento de la aplicación de manera verídica a como se realizaría en caso de hacer un uso de la misma siendo un usuario normal y no un desarrollador.

Para la realización de estas pruebas hemos empleado todos los casos de uso aplicados en las pruebas de integración y en las pruebas de aceptación y sistema de manera que tenemos máxima cobertura.

Antes de comenzar estas pruebas solucionamos el problema principal que era que nuestra aplicación no sobrescribía en el XML las operaciones realizadas y se debía a que no se hacia uso del método finaliza() desde la clase GestionImpuestosCircualcion. Una vez solucionado este problema principal procedimos a la realización de las pruebas.

Mencionar que dado que son pruebas con la base de datos debe permanecer intacta al final de los test tal y como se encontraba al finalizarse y por tanto al comenzar realizamos un backUp del XML con los métodos añadidos:

overwriteXML(), writeBackUpXML(), readBackUpXML().

De manera que al terminar todas las pruebas el XML permanece intacto.

Las pruebas realizadas en este caso son las siguientes y consisten en:

* I3.a Un contribuyente intenta darse de alta y se da de alta correctamente
* I3.b Un contribuyente intenta darse de alta y se da un error porque ya existe
* I4.a Un contribuyente intenta darse de baja y se da de baja correctamente
* I4.b Un contribuyente intenta darse de baja y se da un error porque tiene vehículos asociados
* I4.c Un contribuyente intenta darse de baja y se da un error porque no existe
* I5.a Un contribuyente intenta consultarse y el sistema muestra los datos correctamente porque el contribuyente existe
* I5.b Un contribuyente intenta consultarse y se da un error porque el contribuyente no existe
* I6.a Se intenta consultar el total de un contribuyente y como el contribuyente existe, retorna 239.8.
* I6.b Se intenta consultar el total de un contribuyente y como no existe, retorna un valor que indique fallo.
* I6.c Se intenta consultar el total de un contribuyente que no tiene vehículos y retorna 0.
* I7.a Un vehículo intenta darse de alta para un contribuyente y se da de alta correctamente
* I7.b Un vehículo intenta darse de alta para un contribuyente y se da un error porque no existe el contribuyente
* I7.c Un vehículo intenta darse de alta para un contribuyente y se da un error porque ya existe el vehículo
* I8.a Un vehículo intenta darse de baja para un contribuyente y se da de baja correctamente
* I8.b Un vehículo intenta darse de baja para un contribuyente y se da un error porque no existe el vehículo
* I8.c Un vehículo intenta darse de baja para un contribuyente y se da un error porque no existe el contribuyente
* I8.d Un vehículo intenta darse de baja para un contribuyente y se da un error porque el vehículo no pertenece al contribuyente
* I9.a Se intenta consultar un vehículo y el sistema muestra los datos correctamente porque el vehículo existe
* I9.b Se intenta consultar un vehículo y se da un error porque el vehículo no existe
* I10.a Intentamos actualizar un contribuyente cambiándole su segundo apellido y se actualizan los datos correctamente porque el contribuyente existe y se devuelve el contribuyente actualizado.
* I10.b Intentamos actualizar un contribuyente cambiándole su segundo apellido y se indica que ha habido un error porque no existe un contribuyente con dicho DNI y no se actualiza nada.
* I11.a Se intenta consultar el listado de contribuyentes cuando no hay ninguno y se devuelve una lista vacía
* I11.b Se intenta consultar el listado de contribuyentes cuando sólo hay uno y se devuelve una lista con un elemento
* I11.c Se intenta consultar el listado de contribuyentes cuando hay varios y se devuelve una lista con todos los contribuyentes

Para la realización de todos estos test hemos modificado los métodos de la capa DAO, es decir, la clase ImpuestosDAO de manera que su comportamiento fuera el adecuado y las modificaciones han sido las siguientes:

* actualizaContribuyente: ya que cuando se actualizaba con éxito no se devolvía el objeto de contribuyente, siempre devolvía null.
* creaVehiculo: el método inicial solo devolvía el propio vehículo y ahora se comprueba que el vehículo no existe previamente, si no existe se añade correctamente al contribuyente.
* eliminaVehiculo: el método inicial solo devolvía el propio vehículo y ahora comprueba que el vehículo existe y lo elimina correctamente.
* actualizaVehiculo: el método inicial solo devolvía el propio vehículo y ahora busca el vehículo y si lo encuentra lo actualiza corretamente.