Taller 7 - Diseño de pruebas
ISIS-1226

David Valderrama Herrera
Martín Daniel Rincón Molina

En el presente documento haremos la descripción del trabajo realizado para la ejecución de un esquema de pruebas sobre un proyecto de Java.

Pruebas

Todas las pruebas se hicieron con base en los requerimentos funcionales consignados en la documentación del proyecto debía ser probado.

Pruebas unitarias y de integración de la clase Almacén

Se diseñaron pruebas para validar el funcionamiento de los siguientes métodos:

- BuscarNodo()
- AgregarNodo()
- EliminarNodo()
- AgregarProducto()
- EliminarProducto()
- VenderProducto()

Para separar los tests referentes a los nodos de los referentes a los productos, los 2 conjuntos se separaron en 2 archivos diferentes: *AgregarNodoTest* y *ProductosTest*. Adicionalmente, cada uno de los archivos cuenta con un método *SetUp()*, el cual carga el archivo de datos que permite crear un nuevo almacén sobre el cual ejecutar las pruebas.

```
1 public void SetUp() {
2     try {
3         almacenTest= new Almacen(new File( "./data/datos.txt" ));
4     } catch (AlmacenException e) {
5         fail("No se pudo cargar el archivo");
6     }
7 }
```

En cuanto a la estrategia de desarrollo, no se le dio mucha importancia a la velocidad de los tests ya que se trata de un programa sencillo y que trata cantidades relativamente pequeñas de datos.

Para garantizar la independencia de cada test, el método *SetUp()* se ejecuta antes de cada uno, de modo que su resultado no se pueda ver afectado por las modificaciones que algún otro test haya hecho a la instancia de Almacén.

Dado el alto acoplamiento de las clases del programa, en muchos casos no fue posible realizar pruebas a una sola unidad, por lo que se tuvo que usar varios métodos para lograr obtener un resultado. Dado que los métodos a probar no retornaban nada, fue necesario hacer uso de métodos getters ajenos al método principal que se quería probar; para los tests simplemente se asumió que estos métodos funcionaban correctamente.

Dado el caso, en un mismo test se hicieron varios llamados al método a probar con diferentes inputs. Lo anterior para verificar el comportamiento del método bajo diferentes condiciones; dada una falla en alguna de estas sub-pruebas, se le puede caracterizar por el mensaje que se le asigna.

A continuación se describe con mayor detalle la prueba de cada método. Cabe recordar que todas las pruebas ejecutan el *SetUp()* antes de ejecutarse.

1. **Agregar producto:** Esta prueba evalúa la respuesta del método al intentar agregar un producto bajo una ID que ya este en uso y bajo una nueva ID. La prueba recibe una excepción si no fue posible crear el nuevo producto; de no recibirse una excepción, se considera que el producto ha sido creado exitosamente.

2. **Eliminar producto:** La prueba envía la señal de eliminar uno de los productos cargados. Como el método EliminarProducto es de tipo void, es necesario usar la instancia de categoría del Almacén y su método *BuscarProducto()*. Si al llamar *BuscarProducto()* sobre el ID del producto eliminado y esté retorna *null*, se considera que la prueba fue exitosa.

3. **VenderProducto:** La prueba consta de 2 casos: la venta de una cantidad positiva de unidades y de una cantidad negativa de unidades. En el primer caso, se espera que la diferencia de unidades vendidas antes y después de ejecutar el método sea el entero positivo con el que se alimentó al método (en el código esta implementado con el numero 3). Para el segundo caso (vendiendo -3 unidades), si luego de ejecutar la prueba la diferencia de unidades vendidas es -3, se considera que la prueba falló.

   Esta prueba es de integración, ya que fue necesario usar el método *darCantidadUnidadesVendidas():* de la clase Producto.

4. **BuscarNodo:** Se buscan 2 nodos diferentes de los cargados; un nodo al azar y el ultimo nodo del archivo. Se considera que si la marca y el nombre de los nodos coinciden con los esperados entonces la prueba fue exitosa.

5. **AgregarNodo():** Se dan 2 casos; agregar un nodo con un padre existente y uno sobre un padre inexistente. En el segundo caso se espera que al buscar el nodo agregado se retorne un *null*. En el primer caso se espera un objeto de tipo Nodo, cuyo nombre sea igual al ingresado.

6. **EliminarNodo:** Se elimina uno de los nodos cargados. Luego de invocar al método EliminarNodo, se usa el método *BuscarNodo()* para buscar el nodo eliminado; solo si el método retorna *null* se considera que la prueba fue exitosa.

Prueba clase Categoría

Se diseñaron pruebas para validar el funcionamiento de los siguientes métodos:

- darNodos()
- tieneHijo()
- buscarProducto()
- darMarcas()
- darPreorden()
- darPosorden()
- darValorVentas()

Tal como se requirió en la guía del taller, se hicieron pruebas sobre los métodos consignados dentro de la clase Categoria. En este caso se hizo una clase homónima dedicada a probar cada uno de los métodos mostrados arriba, estos usan el metodo setUp() para garantizar que haya una instanciación de la clase Almacen, sin la cual sería imposible cargar la información y hacer las pruebas; y una instancia de la clase a probar obtenida a través del método buscarNodo() con parámetro "1"(nodo raíz). Esta fue una decisión de diseño influida por el corto tiempo disponible para desarrollar el taller. Además, vale la pena resaltar que para las pruebas se usó el archivo que viene por default con el taller y lograron completa cobertura según la herramienta de JUnit para Eclipse.
Deliberadamente no se hicieron pruebas sobre los métodos relacionados a la manipulación de nodos, en tanto su funcionamiento ya se probó en la clase de prueba AgregarNodoTest.

1. **Dar nodos:** Esta prueba evalúa la respuesta del método al intentar pedirle a la instancia de la clase Categoria que arroje la lista de nodos hijos, con lo que, se hizo una aserción negativa para comprobar que el retorno no esté vacío.

2. **Tiene hijo:** La prueba usa una aserción positiva al método con parámetro "11.$^{\text{el}}$ cual es un nodo hijo de la instancia sobre la que se hace la prueba.

3. **Buscar producto:** La prueba comprueba por medio de una aserción si el nombre del producto con código 24881271, que resulta de ejecutar este método para la instancia de la clase Categoria, concuerda con el nombre que viene en el archivo.

4. **Dar marcas:** Esta prueba evalúa la respuesta del método al intentar pedirle a la instancia de la clase Categoria que arroje la lista de marcas que son nodos hijos, con lo que, se hizo una aserción negativa para comprobar que el retorno no esté vacío.

5. **Dar preorden:** La prueba confirma por medio de una aserción que, al llamar al método, la instancia de la clase se encuentre en la primera posición.

6. **Dar posorden:** La prueba confirma por medio de una aserción que, al llamar al método, la instancia de la clase se encuentre en la última posición.

7. **Dar valor de ventas:** La prueba confirma por medio de una aserción que, al llamar al método, el valor de ventas para la instancia de la clase concuerde con el valor inicial, el cual debe ser 0.