Nombres: Samuel, Valeria Caro, Nicolas

Taller 7 – Documento de Diseño.

El propósito de este documento es dar contexto y explicar la implementación para el manejo de excepciones de la aplicación Almacén; más específicamente, a las clases almacén y categoría. Todo lo anterior, con el fin de dar mayor cobertura al programa y todos los posibles casos a los que se pueda enfrentar para garantizar un mayor desempeño.

1. **Plan de pruebas:**

Para esta parte, lo primordial era identificar que partes del código funcionan con eficacia, cuales podrían fallar y las posibles formas de mejorar el algoritmo.

Para la clase almacén, identificamos los servicios que debe ofrecer revisando la documentación y es así como concluimos que es primordial que esta clase sea capaz de realizar una carga de datos, una adición o sustracción de elementos y consultores sobre los elementos de la misma.

Partiendo del anterior análisis, identificamos que los siguientes métodos son muy vitales para la aplicación y debemos verificar su correcto funcionamiento.

- **Almacen(File):** Este método es la encargada de crear una instancia del almacén, es decir, el objeto base que ofrece los servicios de la aplicación. Para esta parte, se hace una carga de datos al recibir un archivo desde el directorio del./Data. En un caso hipotético, podría recibir como parámetro un elemento inexistente. Lo mejor sería probar el funcionamiento de esta clase con el SetUp.

- **AgregarProducto():** Para este caso, podemos evidenciar un metodo que sera capaz de, dados unos parámetros –Nombre, marca, categoría, etc-, añadir al codigo un nuevo producto. Debemos verificar que este método si está agregando un producto, lo cual se lograría al realizar la agregación y una posterior búsqueda al elemento, de tal manera que se garantice su existencia en la aplicación.

- **AgregarNodo():** Este método se encargará de añadir un nuevo nodo. La estructura interna del almacén debería actualizarse cuando se utiliza el método -lo cual se puede lograr con una posterior búsqueda sobre la estructura-, sin embargo, entrara en conflicto

cuando se trate de agregar un elemento ya existente.

- **EliminarProducto**(): Este método se encargará de eliminar un producto ya existente en la estructura de la aplicación. Para probar la eficacia, podriamos utilizar un elemento que sabemos que existe y realizar el metodo para eliminarlo; posteriormente, utilizariamos un AssertNull para garantizar su desaparición.

- **EliminarNodo**(): Este método se encarga de eliminar un nodo existente a la estructura interna del sistema. Al igual que en el método anterior, esta clase debe ser capaz de actualizar el estado –eliminar un nodo dado- y puede fallar cuando se intenta eliminar un elemento inexistente.

- **BuscarNodo**(): Este método se encarga de, dado un ID, buscar un nodo en la estructura y retornara el nodo. Para esta parte, consideramos que lo primordial es garantizar que el nodo que retorne sea el mismo que se pasó en ID, así que para verificar debemos comparar el retorno con el nombre correspondiente al ID dado como parámetro.

- **TieneHijo()/BuscarPadre()**: Estas clases reciben como parámetro un nodo (O una subclase de este) y verifica si tiene nodos hijos o nodos padres. Esto funciona para verificar la procedencia de un nodo y determinar de qué tipo de elemento se está hablando y sus elementos (Por ejemplo, a que categoría pertenece un televisor o verificar que tipo de electrodomésticos se manejan en el almacén). Nos interesa saber si este método efectivamente está verificando esta información teniendo en cuenta que conocemos la estructura general y los elementos que deberían estar presentes en cada Categoria y Marca. Por otro lado, debemos garantizar la aplicación sea capaz de responder cuando un método este siendo utilizado para consultar sobre un nodo inexistente.

Posteriormente, nos dimos cuenta de que la mayoría de los servicios anteriormente descritos son prácticamente métodos que utilizan partes de la clase Categoría, recordemos que la categoría, en la estructura, es lo que se encuentra justo debajo de la raíz de la raiz; así que, podemos concluir que estos los métodos homónimos tendrán el mismo comportamiento, razón por las que omitiremos su análisis en esta sección.

- **Categoria**(): Este método se encarga de cargar los datos desde un archivo para crear una marca. Al igual que en la clase anterior, este método se puede probar con el SetUp. Si el archivo no existe es muy posible recibir una excepción.
- Dar listas: Hay métodos que se encargan de retornar las listas, entonces debemos garantizar que si estén cumpliendo con esta función. Lo mejor seria revisar que las listas retornadas no estén vacías, así garantizamos que la operación se realizó con éxito.

2. Diseño de pruebas.

Teniendo en cuenta todo aquello que debemos probar y las cosas que podrían salir mal y creemos necesitar manejar, pudimos llegar a la conclusión de tomar las siguientes pruebas:

- Constructores; debemos garantizar que la construcción de cada una de las clases funcione, razon por la cual las concentraremos en una sola prueba. Si todo se ejecuta correctamente, habremos probado que el método cumple con su función. Usa BeforeEach para ejecutarse primero y permitir la realización de más pruebas.

- Búsqueda de nodos: Como su nombre lo indica, será una prueba enfocada únicamente en revisar si la búsqueda de nodos esta retornando el nodo por medio del ID dado como parámetro. Este método utiliza AssertEquals(), dado que ya tenemos conocimiento de cada uno de los nodos y todo se reduce a comparar el nombre del nodo que buscamos y el nombre del nodo retornado.

- Agregación y eliminación de nodos: Esta parte se encargará de utilizar las operaciones con nodos y revisar que este efectuando un cambio. Para esto, utilizamos ambos métodos: iniciamos agregando un nuevo nodo para posteriormente utilizar AssertEquals() para buscar el elemento agregado y comparar su nombre con el nombre que le dimos al objeto recién añadido. Posteriormente, debemos deshacer el código para que no se conserven los cambios dentro de la aplicación, asi que utilizamos la eliminacion de nodo y llamamos AssertNull() para llamar la clase y corroborar su inexistencia.

- Agregación y eliminación de productos: Funciona de la misma manera que el caso con nodos.

- Venta de productos: Para esta parte, creamos un nuevo producto y le asignamos un número fijo de unidades y posteriormente vendemos una cantidad especifica. Para probar su funcionamiento, utilizamos AssertEquals() sobre el objeto utilizando la cantidad de ventas por producto; es decir, se compara las cantidades vendidas que ingresamos con las unidades vendidas del producto.

- Valor de ventas: Implementa un AssertEquals() para determinar que el precio que posee es el mismo que nosotros conocemos.

- Dar listas: Se limita a utilizar AssertFalse() sobre los métodos darNodos(), darProductos(), darPosorden() y darPreorden() para garantizar que al menos haya un elemento tras la carga de datos.

Pruebas para métodos que lanzan excepción:

Sabemos que algunos métodos pueden lanzar excepciones por el hecho de que se están realizando operaciones en situaciones donde no deberían y es por esto que separamos los casos de fallos en esta parte. Todo esto gracias a que implementamos AlmacenException que se encargara de notificar esto.

- Agregar nueva categoría (Nodo): Para esta parte, solo es necesario verificar que logre cumplir con su responsabilidad, así que nos limitamos a llamar al método y lanzar la excepción correspondiente.

- Eliminar la raíz: Este sería una terrible decisión porque acabaría con la estructura del sistema. Ahora, para evitar que el usuario realice esto, vamos a lanzar una excepción que será atrapada por AssertThrows() y garantizar que esto no se logre.

- Agregar un elemento que ya existe: Como ya sabemos, no podemos agregar algo ya existente en una estructura y es por esto que lanzamos una excepción cuando el usuario trate de hacer esto. Para manejar la prueba, utilizamos AssertThrows() para garantizar que no se permita esto.

- Eliminar un elemento que ya existe: Por último, no podemos eliminar algo que no existe. Así que optamos por lanzar una excepción cuando se elimine un producto/nodo que no exista. Al igual que antes, utilizamos AssertThrows().

Finalmente, con todo lo anterior conseguimos una cobertura bastante alta para el programa.