**Diseño de pruebas**

Para empezar nuestro diseño de pruebas deberemos tener en cuenta que no realizaremos una sola prueba por método, esto para asegurar que no solamente vamos a tener una cobertura amplia, sino también que tendríamos en cuenta casos que lo mas probable es que ocurran por improbables que sean. Para empezar, como bien sabemos, vamos a realizar pruebas a las clases categoría y almacén, las cuales van a ser realizadas en JUnit.

Nuestro método de diseño será explicar la descripción de cada uno, los prerrequisitos, pasos a seguir, lo que se espera y los resultados que se obtienen al operar la prueba. Todo esto teniendo en cuenta que hay 2 escenarios para almacén y 2 escenarios para categorías, lo cual nos indica que las pruebas tendrán una extensión la cual nos indica que habrá mayor cobertura.

**Para empezar, mostraremos el diseño de las pruebas de la categoría almacén.**

Tendremos dos variables globales las cuales serán almacenVacío y almacenLleno, una contiene solamente una categoría sin nodos y la otra el almacén completo. Tenemos el método setUp() el cual simplemente se encarga de cargar estas dos variables.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_numCategories()** |  |  | |
| **Descripción:**  Prueba si el número de categorías cargadas son las esperadas. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Procedimos inicialmente a obtener el número de categorías por medio del método preorden y luego comparando este con el número que sabíamos que tenia cada almacén cargado. Si coincidía el número no habría ningún error. | | |
| **Resultado esperado:**   * Si es el número de categorías correcto. Ningún error * Si el numero no corresponde. Un fail() indicándonos lo que sucedió | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_darCategoriaRaiz()** |  |  | |
| **Descripción:**  Prueba si la categoría raíz es la esperada. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se obtiene la categoría raíz por medio del método darCategoriaRaiz() y se compara con la que sabemos gracias al txt que es la categoría raíz. | | |
| **Resultado esperado:**   * Si era la categoría esperada. Ningún error. * Si no era la categoría esperada. Un fail() indicándonos lo que sucedía | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_eliminarNodoCasoRaiz()** |  |  | |
| **Descripción:**  Intenta eliminar el nodo raíz para ver el funcionamiento del método. | | |
| **Prerrequisitos**  Se necesitan los datos cargados. | | |
| **Pasos:**  <P. Procedemos a intentar eliminar el nodo raíz por medio del método eliminarNodo(), si no obtenemos una excepción almacén sabemos que ocurrió un error, en caso de que si concluimos con un final esperado. | | |
| **Resultado esperado:**   * Ningún error si no se logra eliminar el nodo raíz * Un fail() en caso de que se llegara a eliminar el nodo raíz. | | |
| **Resultado obtenido:**  Ningún error. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_eliminarNodoCasoNoRaiz** |  |  | |
| **Descripción:**  Intentar eliminar un nodo diferente al nodo raíz. | | |
| **Prerrequisitos**  Se necesitan los datos cargados. | | |
| **Pasos:**  <P. Por medio de un ciclo se intentan eliminar todos los nodos contenidos en la lista obtenida por el método darPreorden  Usando el método eliminarNodo. Si obteníamos una excepción es que el programa no esta funcionando correctamente ya que si esto sucede es que no se esta eliminando bien el nodo. | | |
| **Resultado esperado:**   * Ningún error si se lograba eliminar todos los nodos. * Un fail() en caso de que no se logre eliminar un nodo diferente a raíz. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarNodoNoExistente()** |  |  | |
| **Descripción:**  Se intenta agregar un nodo no existente a nuestro almacenVacio ya que no contiene nodos. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se agrega un nodo a almacenVacio usando el método agregarNodo, si se lanza una excepción sabemos que no funciona de forma correcta ya que estamos agregando un nodo no existente. | | |
| **Resultado esperado:**   * Ningún error si se logra agregar de forma correcta el nodo. * Un fail() en caso de que no se lograra añadir el nodo. | | |
| **Resultado obtenido:**  Ningún error. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarNodoExistente()** |  |  | |
| **Descripción:**  Se intenta agregar un nodo existente, en caso de que si se logre añadir sabemos que no funciona de forma correcta ya que se espera que no se pueda agregar. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se agrega un nodo a almacenLleno y deliberadamente se escoge un nodo existente para agregar. Si no se puede agregar es porque funcionó. | | |
| **Resultado esperado:**   * Ningún error si no se agrega el nodo * Un fail() si se logra añadir el nodo | | |
| **Resultado obtenido:**  Ningún error. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_buscarNodo()** |  |  | |
| **Descripción:**  Prueba el sistema de búsqueda de nodos. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Por medio del método buscarNodo se busca un nodo existente dentro de almacenLleno. Si es null el nodo significa que no lo encontró y por ende no funciona el método. | | |
| **Resultado esperado:**   * Ningún error si se encuentra el nodo * Un fail() si no se encuentra el nodo | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_venderProducto()** |  |  | |
| **Descripción:**  Se prueba el método de venta de productos | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se usa el método darCategoriaRaiz() para obtener la categoría raíz la cual luego será sujeto del método darProductos de la clase categoría la cual nos permite obtener todos los productos. Después de esto usando el método venderProducto se venden todos los productos de la lista usando un ciclo. Si al final la lista esta vacía significa que funcionó, sino es que no el método no funciona. | | |
| **Resultado esperado:**   * Ningún error si la lista es vacía * Un fail() si no se eliminan todos los elementos de la lista productos. | | |
| **Resultado obtenido:**  Ningún error. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarProductoExistente()** |  |  | |
| **Descripción:**  Se prueba si es posible agregar un producto existente. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se intenta agregar un elemento existente usando el método agregarProducto() en caso de que este no funcione y  L lance una excepción significa que funciono la prueba. | | |
| **Resultado esperado:**   * Ningún error si no se pudo agregar el producto existente * Un fail() si se pudo agregar el producto | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarProductoNoExistente()** |  |  | |
| **Descripción:**  Prueba si es posible agregar un producto no existente. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se intenta agregar un producto que no existe y revisar si se agregar correctamente. Se hace usando el método agregarProducto() si es posible significa que el método funciona. | | |
| **Resultado esperado:**   * Ningún error si se añade el producto * Un fail() si no se añade el producto | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_EliminarProducto()** |  |  | |
| **Descripción:**  Prueba si es posible eliminar un producto | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Al igual que eliminar nodo, se obtiene la lista de productos y por medio del método eliminarProducto() se eliminan todos y se verifica si la lista esta vacia o contiene algún elemento. | | |
| **Resultado esperado:**   * Ningún error si se eliminan todos los productos de la lista * Un fail() si no se logran eliminar todos los productos | | |
| **Resultado obtenido:**  Ningún error | | |

Ahora, para la clase categoría se realizaron pruebas similares, todas fueron implementadas usando el mismo estilo de prueba. Adicionalmente, se tienen dos casos para las pruebas, cuando la categoría no tiene nodos y cuando tiene.

**Categoría:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_listaNodosVacia()** |  |  | |
| **Descripción:**  Prueba si es posible obtener una lista de una categoría sin nodos | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se obtiene la lista por medio de darNodos() si es posible obtener una lista entonces el método no funciona de forma correcta. | | |
| **Resultado esperado:**   * Ningún error si no se crea una lista * Un fail() si se crea una lista | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_listaNodos()** |  |  | |
| **Descripción:**  Prueba si es posible obtener una lista de nodos de una categoría que si contiene nodos | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Al igual que la prueba anterior se obtiene la lista por medio de darNodos(). Sin embargo, si la lista es dada de forma correcta significa que el método funciona. | | |
| **Resultado esperado:**   * Ningún error si se obtiene una lista * Un fail() si no se logra obtener la lista. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_buscarPadreVacio()** |  |  | |
| **Descripción:**  Prueba si es posible obtener el nodo padre de una categoría sin nodos. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se intenta buscar el padre con un nodo no valido y una categoría sin nodos, si es posible obtener un nodo significa que el método no esta funcionando de forma correcta | | |
| **Resultado esperado:**   * Ningún error si no se obtiene un nodo valido. * Un fail() si se obtiene un nodo valido. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_buscarPadreNoVacio()** |  |  | |
| **Descripción:**  Prueba si es posible obtener el padre de una categoría con múltiples nodos. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Al igual que la prueba anterior, se usa buscarPadre con el identificador de un nodo valido que se sabe que es padre. Si no se logra conseguir el nodo el método no está funcionando correctamente. | | |
| **Resultado esperado:**   * Ningún error si se encuentra el nodo * Un fail() si no se encuentra el nodo | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_buscarNodo()** |  |  | |
| **Descripción:**  Prueba si es posible buscar un nodo | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se obtiene una lista de nodos usando darNodos() luego se recorre y se usa el método buscarNodo() sobre cada nodo para ver si es capaz de dar cada nodo de forma correcta. | | |
| **Resultado esperado:**   * Ningún error si encuentra todos los nodos * Un fail() si no se logran encontrar todos los nodos | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarNodoExistente()** |  |  | |
| **Descripción:**  Prueba si es posible agregar un nodo que ya existe | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <PS Se prueba deliberadamente si es posible agregar un nodo que ya existe usando categoriaMore ya que esta contiene nodos. | | |
| **Resultado esperado:**   * Ningún error si no se agrega el nodo. * Un fail() si se agrega el nodo. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarNodoNoExistenteDifCategoria()** |  |  | |
| **Descripción:**  Prueba si es posible agregar un nodo a una categoría con otros nodos y una categoría nueva. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se intenta agregar un nodo con un identificador y categoría nueva la categoría, si es posible agregar es que el método funciona correctamente. | | |
| **Resultado esperado:**   * Ningún error si se agrega el nodo * Un fail() si no se logra agregar el nodo. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_agregarNodoNoExistente()** |  |  | |
| **Descripción:**  Prueba si es posible agregar un nuevo nodo a una categoría con otros nodos y misma categoría. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Al igual que el método anterior se intenta agregar un nodo con el método agregarNodo(), se usa una categoría existente en uno de los parámetros y así se agrega a esa categoría también. | | |
| **Resultado esperado:**   * Ningún error si se logra añadir el nodo correctamente * Un fail() si no se logran agregar el nodo. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_EliminarNodo()** |  |  | |
| **Descripción:**  Prueba si es posible eliminar un nodo | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se obtiene la lista de nodos de la categoría con múltiples nodos usando el método darNodos(), se recorre y se eliminan todos los nodos. Si es posible eliminarlos el método eliminarNodo() funciona. | | |
| **Resultado esperado:**   * Ningún error si se eliminan todos los elementos de la lista. * Un fail() si no se logran eliminar todos los nodos. | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_buscarProducto()** |  |  | |
| **Descripción:**  Prueba si es posible buscar y retornar un producto | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se obtiene la lista de productos recorriendo la lista de nodosAlmacen y aplicando darProductos sobre cada nodo. Se recorre la lista de productos y se obtiene si es posible buscar todos los productos. | | |
| **Resultado esperado:**   * Ningún error si se logran buscar todos los productos * Un fail() si no se logran buscar todos los productos | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_darMarcas()** |  |  | |
| **Descripción:**  Prueba si es posible obtener la lista de marcas | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se usa el método darMarcas, si la lista es vacia significa que el método no funciona. | | |
| **Resultado esperado:**   * Ningún error si se obtiene la lista de marcas. * Un fail() si no se obtiene la lista de marcas | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_darPreorden()** |  |  | |
| **Descripción:**  Prueba si es posible obtener la lista de preorden | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se usa el método darPreorden, si la lista es vacia significa que el método no funciona. | | |
| **Resultado esperado:**   * Ningún error si se obtiene la lista de preorden. * Un fail() si no se obtiene la lista de preorden | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_darPosorden()** |  |  | |
| **Descripción:**  Prueba si es posible obtener la lista de posorden | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P. Se usa el método darPosorden, si la lista es vacia significa que el método no funciona. | | |
| **Resultado esperado:**   * Ningún error si se obtiene la lista de posorden. * Un fail() si no se obtiene la lista de posorden | | |
| **Resultado obtenido:**  Ningún error | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test\_darValorPrecioVenta()** |  |  | |
| **Descripción:**  Prueba si es posible obtener el valor de precio de venta de los productos de una categoría. | | |
| **Prerrequisitos**  Se necesitan los datos cargados | | |
| **Pasos:**  <P Se obtienen los nodos usando darNodos() se recorren y se suma el precio de venta de cada uno de ellos. La suma se guarda en una variable que se compara con el valor encontrado usando el método darValorVentas(). | | |
| **Resultado esperado:**   * Ningún error si se encuentra que son el mismo valor * Un fail() si no son el mismo valor. | | |
| **Resultado obtenido:**  Ningún error | | |

Con estas pruebas podemos obtener un índice de cobertura de alrededor de 70% en cada clase. Después de cierto análisis podemos concluir que esto sucede debido a que hay muchas instrucciones que no se toman en cuenta. Muchas de ellas pueden ser casos donde archivos no carguen correctamente, o cuando se crean las listas no se verifica cual es el contenido de estas. Para una examinación a más profundidad se realiza una comparación de los elementos y demás. Por otro lado, también en los casos de error no se profundiza mucho, esto se hace intencionalmente para no estar causando errores en la consola muy seguido.

****

****

Ahí podemos ver la cobertura en ambas pruebas.

Un dato curioso es que así sean pruebas para una sola clase cuando usamos el sistema de cobertura de Eclipse podemos ver que es posible ver la integración de todas las clases. Por ejemplo, en testAlmacen:

A picture containing application

Description automatically generated

Y en testCategoria:

Graphical user interface, application

Description automatically generated with medium confidence

Con esto ya podemos concluir que las pruebas fueron un gran éxito y muchas de ellas dieron los resultados esperados.