Formulario Práctica JUnit

Debe rellenar este formulario y subirlo al Blackboard de la asignatura para poder evaluar la práctica.

1. Con la información proporcionada para la realización de la práctica, ¿qué tipo de pruebas se llevan a cabo en la práctica? Justifique su respuesta:

En proyectos de ingeniería del software existen diversos tipos de pruebas: unitarias, funcionales, de integración, de validación, de sistema, de caja negra o de caja blanca (sistema), de regresión, de carga y/o prestaciones. Las realizadas en la práctica han sido fundamentalmente 2:

* + Pruebas de caja negra (clases *Gestor y Producto)*: pruebas centradas en lo que se espera de un módulo (no en su especificación). Se basan en una fuente de información externa como un *JAVADOC* que nos proporcionan, como en el caso de la práctica de *Gestor y Producto*.
  + Pruebas de caja blanca (clase *Buscador)*: pruebas muy exhaustivas que intentan recorrer todas las posibles secuencias de sentencias basándose en clases creadas previamente y a las que tenemos acceso.

1. En el ejercicio 2 de la práctica ejecutó la batería de pruebas pruebasINSW3.jar, ¿cuántos tests se realizaron con la batería? ¿qué resultados obtuvo? ¿qué deduce a partir de los resultados obtenidos?

En el ejercicio 2 de la práctica ejecuté un total de 11 tests de la clase *Gestor* y un total de 2 test de la clase *Producto.*

En la clase *Gestor*, existen 2 método para la implementación de las pruebas:

*AñadirProducto* es un método que añade un nuevo producto en el almacén con las siguientes características del producto:

* + Nombre: se admite cualquier nombre del producto.
  + Precio: debe ser mayor de 0 (no hay límite máximo) para poder incluirse dentro del almacén.
  + Cantidad: el almacén admite de 0 a 300 unidades de un mismo producto.
  + Si se añade un producto de nombre X y X ya existe en el almacén, se sustituye el producto X que había por el nuevo No pueden existir productos repetidos, con el mismo nombre, en el almacén.

Las pruebas de dicho método han concluido en que en rasgos generales el producto es añadido si tiene especificaciones normales. Si es uno repetido también lo añade, pero suponemos que por dentro no está repetido en el almacén si no que lo sustituye. Sin embargo, si tiene un precio o una cantidad negativa devuelve *true*, lo que podría llevar a confusión.

*QuitarXProducto* es un método que quita X unidades del producto indicado. Además, no podrán existir cantidades de productos negativos.

En general parece ser que existe un problema grave con el método en la casuística de que exista un solo producto en el almacén, dando lugar al error:

*java. lang. IndexOutOfBoundsException: Index 0 out of bounds for length 0,*

esto indica que no está bien gestionada la iteración de los productos del almacén.

Cabe destacar, que el anterior error evita hacer otro tipo de pruebas como intentar quitar x cantidades de uno de los productos, intentar poner una cantidad negativa o al intentar poner una cantidad que es mayor que la que está almacenada del producto, ya que aparece el error comentado anteriormente.

Por otra parte, si intentamos borrar cantidades de un producto que no existe en el almacén, también nos devuelve *true*, un grave error ya que no puede quitar cantidades de un producto no existente.

Por último, si le entregamos un *null product* nos aparece el error de *NullPointerException.*

1. En el ejercicio 3, ha creado dos tests y uno de ellos produce error. ¿Cuál de ellos? ¿Qué quiere decir el error que produce el test ejecutado?

En el ejercicio 3, es decir, el ejercicio del *Buscador,* existen varios métodos. Uno de los que producía error era *buscarFrase (),* concretamente al intentar buscar una frase que se encontraba en la última posición de la lista. Esto se debe a que el método únicamente comprueba si la frase se encuentra en la primera posición de la lista y posteriormente deja de iterar. En cambio, la función *BuscarPalabra ()* utiliza la función predeterminada de *List.java.*

En segundo lugar, el método de *devolverPalabra ()* falla al indicarle una posición de la palabra muy alta con la excepción *java.lang.ArrayIndexOutOfBoundsException*. Cuando le das una lista vacía devuelve la excepción *java.lang.NullPointerException*. Habría que añadir las condiciones para cada uno de los casos para poder evitarlos.

En cuanto a *devolverPrimerElemento ()* también da error al pasarle una lista vacía o nula, al igual que en el método *devolverUltimoElemento ().* En este último también es necesario indicarle bien la última posición, es decir, *list. size ()-1.*

1. ¿En qué consiste el criterio de clases de equivalencia y cómo lo ha aplicado para realizar su batería de pruebas?

El criterio de clases de equivalencia es una técnica de diseño de pruebas que divide el conjunto de datos de entrada en clases de manera que se asuma que, si un caso de prueba funciona para una clase, también funcionará para todos los demás casos de esa clase. El objetivo es reducir la redundancia en la creación de casos de prueba y garantizar una cobertura adecuada del espacio de entrada. Por tanto, consiste en lo siguiente:

1. Identificación de clases de equivalencia: agrupar las entradas en clases que se espera que se comporten de manera similar.
2. Definición de casos de prueba: seleccionar un representante de cada clase de equivalencia como caso de prueba.
3. Ejecución de pruebas: probar cada caso de prueba y verificar si el comportamiento del sistema es conforme a las expectativas.

En el caso de la práctica lo he aplicado de diferentes formas según la clase.

En la clase *Gestor* tenemos las siguientes clases de equivalencia del método añadirProducto:

* + Condiciones normales.
  + Con un producto ya existente.
  + Caso inválido: producto nulo.
  + Caso inválido: precio menor a 0.
  + Caso inválido: cantidad menor a 0 o mayor a 300.

En la clase *Gestor* tenemos las siguientes clases de equivalencia del método QuitarXProducto:

* + Condiciones normales.
  + Caso límite: cuando hay una cantidad de un solo producto en el almacén.
  + Caso inválido: producto nulo.
  + Caso inválido: cuando el producto no existe en el almacén.
  + Caso inválido: cantidad menor a 0 o mayor a 250.

En la clase *Producto* tenemos las siguientes clases de equivalencia del método añadirCantidad:

* + Condiciones normales.
  + Cantidad negativa.

1. Complete la siguiente tabla con los tests que ha realizado en su batería de pruebas, justifique la utilidad del test realizado y los resultados obtenidos en cada uno de los tests. Puede añadir todas las filas que considere necesarias en la tabla:

|  |  |  |
| --- | --- | --- |
| Test | Justificación del test | Resultado |
| AÑADIR PRODUCTO (*Gestor class)* | | |
| Se añade un producto correctamente | Comprobar funcionalidad básica del módulo | Expected: true  Result: true |
| Se añade un producto ya existente sustituyendo al otro correctamente (sin duplicación) | Comprobar funcionalidad básica del módulo | Expected: true  Result: true |
| Se lanza excepción al introducir producto = null | Comprobar que al introducir un elemento null en el almacén se lanza excepción | Expected: false  Result: false |
| No se añade un producto porque tiene un precio negativo | Comprobar que, al introducir un elemento con precio negativo, no se añada. | Expected: false  Result: true |
| No se añade un producto porque tiene una cantidad negativa | Comprobar que, al introducir un elemento con cantidad negativo, no se añada. | Expected: false  Result: true |
| No se añade un producto porque tiene una cantidad superior a 250. | Comprobar que, al introducir un elemento con cantidad >250, no se añada. | Expected: false  Result: false |
| QUITAR X PRODUCTO (*Gestor class)* | | |
| Se quitan X unidades de un producto correctamente | Comprobar funcionalidad básica del módulo | java. lang. RuntimeException: java. lang. IndexOutOfBoundsException: Index 0 out of bounds for length 0 |
| Se lanza excepción al introducir producto = null | Comprobar que al introducir un elemento null en el almacén se lanza excepción | NullPointerException |
| Se lanza una excepción si le das un producto que no existe en el almacén | Comprobar que al introducir un producto que no está en el almacén no le quite ninguna cantidad ya que no existe. | Expected: false  Result: TRUE |
| No se elimina la cantidad de un producto porque tiene una cantidad negativa | Comprobar que al introducir una cantidad negativa te avise o directamente no quite cantidades. | java. lang. RuntimeException: java. lang. IndexOutOfBoundsException: Index 0 out of bounds for length 0 |
| No se elimina la cantidad de un producto porque tiene una cantidad superior a la almacenada en el almacén. | Comprobar que al superar el límite de la cantidad te avise. | java. lang. RuntimeException: java. lang. IndexOutOfBoundsException: Index 0 out of bounds for length 0 |
| AÑADIR CANTIDAD (*Producto class)* | | |
| Se añade una cantidad y un producto correctamente | Comprobar funcionalidad básica del módulo | Expected: true  Result: true |
| Se añade/elimina una cantidad negativa de un producto correctamente | Comprobar funcionalidad básica del módulo | Expected: true  Result: true |