FORMULARIO JUNIT

Introducción a la ingeniería de software

Paula Sáenz de Santa María Diez
Universidad Europea de Madrid 2023-2024

Formulario Práctica JUnit

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

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

Las pruebas que se han llevado a cabo son pruebas de caja negra y caja blanca. Estas pruebas tienen enfoques diferentes para evaluar y mejorar la calidad de un software. En las pruebas de caja negra, se examina la funcionalidad del sistema sin conocer los detalles internos de su implementación. Se centran en la entrada y salida del programa, estas las hemos usado para hacer las pruebas sobre los códigos de pruebasINSW.jar, ya que contábamos con la documentación del código, pero no el código en sí (teníamos el código, pero no podíamos cambiarlo).

Por otro lado, las pruebas de caja blanca conocemos con detalle la estructura interna del código fuente. Se diseñan casos de prueba basándose en la lógica interna del programa, explorando diferentes caminos de ejecución y asegurándose de que todas las instrucciones sean ejecutadas correctamente, esto lo hemos tenido con el código Buscador en el cual podíamos cambiar sus métodos y ver el código entero.

- **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 mi batería de pruebas, tengo un total de ocho pruebas definidas en la clase "*AlmacenProductosTestCase*" para detectar posibles errores en el código y verificar su funcionalidad. En cuanto a los resultados obtenidos, la mayoría de las pruebas salieron correctas, pero tuve un problema en la implementación del caso específico en la prueba TestQuitarXProductos_MasDeLoQueHay. Esta prueba no funciona correctamente debido a cómo está implementado el código en la clase de Gestor.

A partir de los resultados obtenidos se deduce que la implementación del método *"quitarXProducto"* contiene unos errores y que se tiene que mejorar el caso en el que se intenta quitar más productos de los que hay en el almacén. También deducimos que para probar de manera efectiva un método, hay que realizar pruebas de todos los posibles casos que tenga el método.

- **En el ejercicio 1, 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 código original hay algún error, pero de verdad nos damos cuenta de todo lo que podemos mejorar cuando hacemos las pruebas. Esto se debe, a que, como he dicho anteriormente, hay que realizar pruebas de todos los caminos posibles que se pueden tomar cuando se llama a un método. En cuanto empezamos a hacer pruebas, nos damos cuenta de que por ejemplo en el método "*devolverUltimoElemento*" falta un -1

cuando cogemos el tamaño del array, esto ocurre con varios métodos de la clase Buscador. El error que hay cuando se ejecuta el test nos quiere decir que un comportamiento observado durante la ejecución del test que no coincide con el resultado esperado. Este resultado inesperado nos deice que hay algún problema en el código.

- **¿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 casos de prueba que se utiliza en pruebas de software. Esta técnica divide el conjunto de datos de entrada en clases de equivalencia de manera que cada clase represente un conjunto igual de casos de prueba. La idea es que, si un caso de prueba es válido o no para una clase de equivalencia, lo será para todos los casos de esa clase. De esta manera, la prueba de un solo caso de cada clase de equivalencia no da una garantía que conjunto completo tendrá el mismo comportamiento.

Como ejemplo de la clase de *AlmacenProducto:*

Método *AgregarCantidad (testAñadirCantidadPositiva, testAñadirCantidadCero, testAñadirCantidadNegativa):*

Clase válida: Cantidades positivas.

Clase límite: Cantidad igual a cero.

Clase inválida: Cantidades negativas.

Un ejemplo de la clase *Buscador*:

Método *buscarFrase: (testBuscarFrase_Existente, testBuscarFrase_NoExistente, testBuscarFrase_Null, testBuscarFrase_ListaVacia):*

Clase Válida: La frase buscada está presente en la lista.

Clase Inválida: La frase buscada no está presente en la lista.

Clase Límite: La lista está vacía.

Clase Límite: La frase buscada es nula.

- **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
testBuscarFrase_Existente	Verificar que el método buscarFrase detecta frases existentes en la lista.	Éxito (assertTrue)
testBuscarFrase_NoExistente	Comprobar que el método buscarFrase devuelve falso cuando la frase no está presente.	Éxito (assertFalse)
testBuscarFrase_Null	Validar el comportamiento cuando la frase buscada es nula.	Éxito (assertFalse)
testBuscarFrase_ListaVacia	Verificar el manejo de una lista vacía en buscarFrase.	Éxito (assertFalse)
testBuscarFrase_ListaNull	Comprobar cómo se maneja el caso de una lista nula en buscarFrase.	Éxito (assertFalse
testBuscarPalabra	Asegurarse de que buscarPalabra funcione correctamente.	Éxito (assertTrue y assertFalse)
testDevolverPalabra_PosicionValida	Verificar que el método devolverPalabra devuelve la palabra correcta para una posición válida en la lista.	Éxito (assertEquals)
testDevolverPalabra_PosicionInvalida	Asegurarse de que se lanza la excepción correcta al intentar devolver una palabra en una posición fuera de rango.	Éxito (assertThrows)
testDevolverPalabra_ListaNull	Comprobar cómo se maneja el caso de una lista nula en devolverPalabra	Éxito (assertThrows)
testDevolverPrimerElemento	Verificar que devolverPrimerElemento devuelve el primer elemento de la lista.	Éxito (assertEquals)
testDevolverPrimerElemento_ListaNula	Comprobar cómo se maneja el caso de una lista nula en devolverPrimerElemento.	Éxito (assertThrows)
testDevolverPrimerElemento_ListaVacia	Asegurarse de que se lanza la excepción correcta al intentar devolver el primer elemento de una lista vacía.	Éxito (assertThrows)
testDevolverUltimoElemento	Validar que	Éxito

	devolverUltimoElemento devuelve el último elemento de la lista.	(assertEquals)
testDevolverUltimoElemento_ListaNula	Comprobar cómo se maneja el caso de una lista nula en devolverUltimoElemento.	Éxito (assertThrows)
testDevolverUltimoElemento_ListaVacia	Asegurarse de que se lanza la excepción correcta al intentar devolver el último elemento de una lista vacía.	Éxito (assertThrows)
testAñadirCantidadPositiva	Verificar que añadir una cantidad positiva actualiza correctamente la cantidad de productos.	Éxito (assertEquals)
testAñadirCantidadCero	Asegurarse de que añadir una cantidad de cero no afecta la cantidad de productos.	Éxito (assertEquals)
testAñadirCantidadNegativa	Validar que añadir una cantidad negativa reduce correctamente la cantidad de productos.	Éxito (assertEquals)
testAñadirProductoExitoso	Comprobar que se puede añadir un producto correctamente al almacén.	Éxito (assertTrue)
testAñadirProductoNulo	Verificar el manejo adecuado de un intento de añadir un producto nulo.	Éxito (assertFalse)
testAñadirProductoExcedeCantidadTotal	Asegurarse de que el sistema maneje correctamente el escenario de agregar productos hasta exceder la cantidad total permitida.	Éxito (assertTrue)
testQuitarXproductosFunciona	Validar que quitar una cantidad específica de productos del almacén funciona correctamente.	Éxito (assertTrue)
TestQuitarXProductos_NombreNullYCantidad0	Comprobar el comportamiento al intentar quitar un producto con nombre nulo o cantidad igual a cero.	Éxito (assertTrue)
TestQuitarXProductos_MasDeLoQueHay	Verificar que el sistema maneje adecuadamente el intento de quitar más productos de los que hay disponibles.	Fallo (Error)