PRUEBAS DE VERIFICACIÓN CON JUNIT

PREVIO Y OBJETIVO:

Nada más acabar el grado que estás cursando te contrata una empresa para trabajar en el departamento de calidad del software. El responsable de tu departamento te encarga como primera tarea la verificación del funcionamiento de unas clases ya desarrolladas en Java. Para ello utilizarás la herramienta **JUnit** que te permitirá realizar la implementación de pruebas que verifiquen el correcto o incorrecto funcionamiento de las clases.

La práctica contiene algunas preguntas en color de texto azul. Además de resolver la práctica, debes crear un fichero de texto (que entregarás) en el que vayas respondiendo las preguntas.

PREPARATIVOS:

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de una clase es como se espera. Dicho de otra manera, JUnit es la herramienta utilizada para realizar pruebas unitarias en Java.

Aunque el enunciado está muy guiado, debes consultar la documentación en Internet que necesites sobre **JUnit** para poder resolver los ejercicios propuestos en la práctica.

Lo primero que debes hacer es crear un proyecto nuevo. Crea un paquete, llámalo por ejemplo p4 y sitúa en él las clases facilitadas en el aula virtual. Revisa el código para entender un poco de qué tratan las clases.

EJERCICIOS:

1. Añade un método en la clase *ShoppingCart* que permita buscar un producto por su título (título pasado como parámetro) y nos devuelva *true* o *false* según exista o no en el carrito. Llámalo: *findProduct(String).* Simplemente debes implementar el método. No debes probarlo desde el código del *main*. Lo probaremos posteriormente usando JUnit.
   (Este primer apartado no correspondería al testeo, sino al departamento de programación).

2. Haz un test JUnit de la clase Product:

   a. En el explorador de ficheros de la izquierda, sobre la clase a testear (Product) pulsar botón derecho y escoger: (En Eclipse) *New / Other / Java / JUnitTestCase*
   (En Net Beans) Herramientas / Create JUnit Test

   b. Escribe 'test' en la casilla Package para tener los test organizados.
   (En *'Class under test'* aparece la clase para realizar el test, en este caso 'Product')

   c. Marca todos los stubs (setUpBeforeClass, setUp, tearDownAfterClass, tearDown).

   δ. Marca todos los métodos de Product

   e. Pulsa 'siguiente' o 'next'

Ahora pedirá confirmación para añadir la librería JUnit al build path, pulsar OK. (Fíjate JUnit no es más que una librería y como tal se puede añadir en cualquier momento a un proyecto en Java).

Acepta para que el asistente añada por nosotros la línea:

import static org.junit.Assert.;*

3. Observa que se ha generado una nueva clase llamada '*ProductTest'. Fíjate en lo siguiente*:

   a. Es una clase java estándar.

   b. Los métodos de la clase llevan anotaciones del paquete "org.junit".

   c. Existen cuatro métodos especiales para inicialización y liberación de recursos anotados como *@BeforeClass, @AfterClass, @Before y @After*.

   d. El resto son métodos de test, lo cual se consigue simplemente con la anotación @Test e incluyendo *asserts* en su interior para realizar las comprobaciones que queramos en el test. Vemos el *assert* más simple, que es *fail*() y cómo se usa.

   e. Fíjate en la convención que se usa para nombrar las clases y métodos de test. La clase es el "*test case*" y se nombra con el mismo nombre que la clase a testear más el sufijo *Test,* en este caso *ProductTest*.

   f. Los test unitarios propiamente dichos son los métodos anotados con @Test y normalmente corresponden a un método de la clase original (un método igual a la unidad más pequeña de software que se puede testear), aunque también puede haber más de un test por cada test de la clase original. Se nombran normalmente como test<MetodoProbado>, aunque se puede añadir alguna palabra descriptiva si hay más de un test para un solo método de la clase principal.

   Ahora conviene eliminar el código que nos haya creado en los métodos de test y comentar todos los fail que aparezcan en el código, para que al probarlo, no nos den esos fallos.

4. Vamos a ejecutar el test. Para ello:

   Sobre el test, pulsa botón derecho del ratón *Run As -> JUnit Test*

   Observa la pestaña de resultados de la ejecución.

   *1 ¿Qué diferencia piensas que hay entre un Error y un Failure? (Puedes usar Internet)

5. Realiza con ayuda del asistente (o sin él) una clase de test para la clase *ShoppingCart* (es lo mismo que hemos hecho en el paso 1, pero ahora con la clase *ShoppingCart*).

   Para rellenar el código de los distintos test que se solicitan en los siguientes apartados de la práctica, debes realizar llamadas a métodos de la clase que ya existan. Es decir, no debes añadir ni modificar código de las clases que se facilitan en la práctica, sino usar los métodos ya existentes, llamándolos para realizar las comprobaciones de funcionamiento.

6. Rellena el test unitario *testGetBalance*. En él debes comprobar que la suma de los precios de los productos que introduzcas en el carrito sea igual al resultado ofrecido por la llamada al método *getBalance*(). Usa cualquiera de los assert disponibles en la librería (por ejemplo *assertEquals*) para realizar la comprobación.

*2 Explica por qué este test no es unitario.

7. Rellena el test unitario *testAddItem*. Debes pensar cómo puedes probar que el método funciona correctamente. No realices llamada al método creado anteriormente: *findProduct*

8. Crea un nuevo método de test llamado *testAddItemFindProduct* que realice la comprobación del correcto funcionamiento del método *addItem*, pero esta vez realizando llamada o llamadas al método *findProduct*.

9. Implementa el test *testRemoveItem*: en este test debes comprobar la funcionalidad del método *removeItem*, y concretamente que la excepción no salta cuando no debe hacerlo. Es decir, el test debe fallar (debes llamar a *fail*(mensaje)) en caso de que salte la excepción de forma inadecuada. Si la excepción no salta, entonces el método está funcionando correctamente, y por tanto no debes llamar a *fail().*

10. Crea e implementa un nuevo método que debes llamar *testRemoveItem2* con otra versión de test de la funcionalidad del método *removeItem,* con una comprobación equivalente, pero usando *assertTrue*.

11. Crea e implementa un nuevo método que debes llamar *testRemoveItemCarroVacio* con otra versión de test de la funcionalidad del método *removeItem.* En este caso debes tener en cuenta lo siguiente:

> Según los requisitos del cliente, el funcionamiento correcto del método *RemoveItem* implica que, en caso de intentar eliminarse un producto de un carro vacío, éste lance una excepción. En este test comprobaremos esto. Para ello, este test debe fallar cuando se intente eliminar de un carro vacío y no salte la excepción. Es decir, el test falla en caso de que no se lance la excepción cuando debería haberse lanzado.

12. Escribe el test *testGetItemCount*. Comprueba que funciona correctamente. Dentro del test realiza varias comprobaciones (no te quedes sólo con una comprobación).

13. Escribe el test del constructor por defecto (de la clase *ShoppingCart*).

> Realiza las comprobaciones que creas convenientes, es decir, ¿qué debe cumplirse tras llamar al constructor de la clase `ShoppingCart`? (eso es lo que deberías comprobar).

14. Escribe el código del método *testEmpty*.

15. Escribe el código del método *testIsEmpty*.

16. Escribe el código del método *testFindProduct*.

17. Realiza una versión parametrizada de la clase *ShoppingCartTest*. Investiga en Internet cómo se construyen test parametrizados con JUnit.

Ayuda: crea una nueva clase parametrizada llamada `ParamShoppingCartTest` (debe contener dos atributos: precio1 y precio2). En dicha clase deben aparecer los métodos de test creados anteriormente. Modifica el código de los métodos en los que aparezcan precios de productos para que los test se realicen con los valores parametrizados (es decir, con los valores de los atributos precio1 y precio2).

*3 ¿Qué utilidad crees que tiene realizar una clase de test parametrizada? Pon algún ejemplo.

18. Por último vamos a comprobar la cobertura alcanzada por nuestros test. Para ello:

- En Eclipse:
  Descarga el plugin *eclEmma* (desde la opción de Eclipse Help / Install new software, buscando eclEmma, marcándolo y pulsando varias veces a Next para instalarlo). Ejecuta la cobertura (utiliza Internet si lo requieres para ver su funcionamiento).
- En NetBeans:
  Selecciona Tools > Plugins y escoge "JaCoCo Library Code Coverage" en la lista de plugins disponibles. Para activar el plugin pulsa botón derecho en el explorer de proyectos y pulsa en: Coverage > Activate Coverage Collection. Ejecuta el test Junit para probarlo.

*4 Captura las pantallas y adjúntalas al fichero de respuestas. Explica qué son las líneas verdes y rojas, los números y porcentajes que se muestran.

ENTREGA:

La práctica deberá entregarse mediante la tarea asociada al aula virtual, antes del inicio de la primera sesión de la práctica siguiente (únicamente por uno de la pareja de prácticas). Recuerda que debes poner el nombre de la pareja en el código.

Únicamente se subirá un fichero por pareja, comprimido en formato zip o rar y que incluirá:

- Directorio completo src con las clases de Java.
- Fichero llamado 'respuestas.pdf' con las respuestas a las cuestiones planteadas.