Tarea Unidad 5

1. En los tipos de pruebas, ¿que es una caja blanca y una caja negra?

Una caja blanca es el tipo de pruebas de software que se realiza sobre las funciones internas de un módulo, mientras que una caja negra es el tipo de pruebas que ejercitan los requisitos funcionales desde el exterior del módulo (funciones externas).

1. ¿Que se debería probar en un test unitario?

* La cardinalidad.
* El orden.
* El rango.
* La referencia.
* La existencia.
* La conformidad.
* El tiempo.
* Otras cosas.

1. ¿Que es un caso de prueba?

Un caso de prueba es un conjunto de condiciones o variables bajo las cuales un analista determinará si una [aplicación](https://es.wikipedia.org/wiki/Aplicación_informática), un sistema software o una característica de éstos es parcial o completamente satisfactoria.

1. ¿Cuáles son los principios que debe seguir un caso de prueba?

* La salida deseada es una parte necesaria para cada caso de prueba.
* Un programador debe evitar probar su propio programa; al programar se construye y al probar se destruye (a corto plazo).
* Una organización no debe probar sus propios programas.
* Inspeccionar mucho los resultados de cada prueba.
* Escribir casos de pruebas para: Entradas inválidas e inesperadas, y además de para las válidas y esperadas.
* Intentar ver si no hace lo que se supone que debe de hacer y si hace lo que no se supone que debe de hacer.
* Evitar usar caso de prueba de usar y tirar, a no ser que el programa sea de usar y tirar
* Guardar siempre los casos de prueba para no perder tiempo reinventando casos de prueba y Posibilitar efectuar pruebas de regresión.
* No plantear una prueba asumiendo que no se van a encontrar errores.
* La probabilidad de existencia de errores en una sección de programa es proporcional al número de errores ya encontrados en esa sección.

1. JUnit utiliza una serie de anotaciones en la cabecera de cada método de test, ¿cuáles son las más comunes o utilizadas y para qué sirve cada una?

* @Test: Marca un método como método de prueba.
* @Before: Marca un método que se ejecutara antes de cada prueba; es útil para inicializar los datos de entrada y de salida que se vayan a utilizar en las pruebas.
* @After: Se ejecuta después de cada test y nos servirá para liberar recursos que se hubiesen inicializado en un método @Before.
* @BeforeClass: Se ejecuta una sola vez antes de ejecutar todos los test de la clase y se utilizarán para crear estructuras de datos y componentes que vayan a ser necesarios para todas las pruebas.
* @AfterClass: Se ejecuta una única vez después de todos los test de la clase y se utiliza para liberar los recursos inicializados por el @BeforeClass.

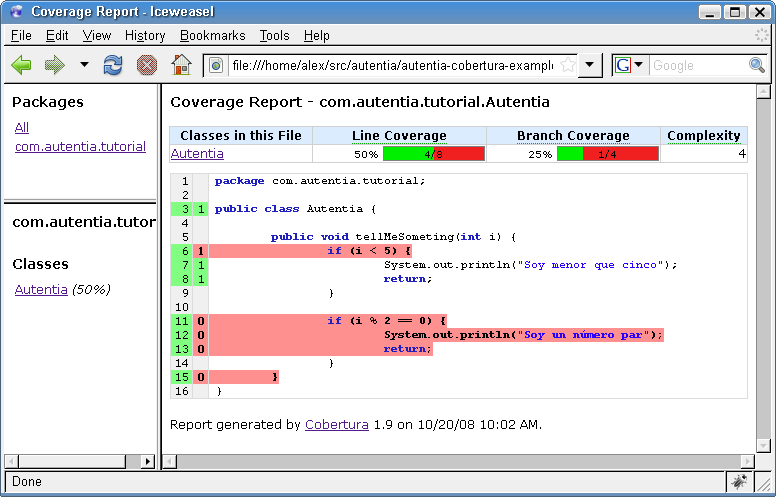
1. **JUnit** proporciona métodos en la clase **Assert** para probar ciertas condiciones. Estos métodos de afirmación típicamente comienzan con assert y le permiten especificar el mensaje de error, el esperado y el resultado real.

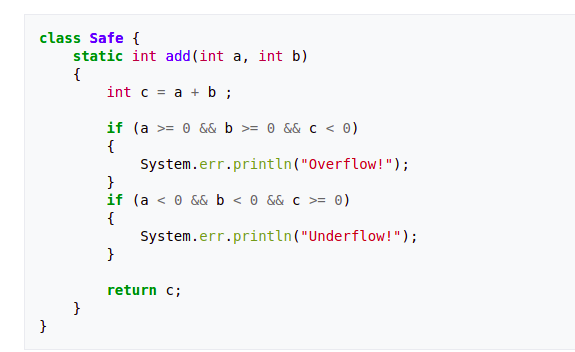
En el siguiente enlace tienes la información de la clase Assert:

<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

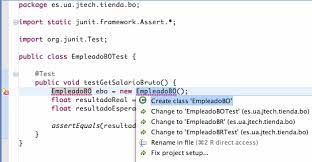
Indica para qué sirven cada uno de los siguientes asserts:

* **assertArrayEquals():** Comprueba si dos arrays son iguales.
* **assertEquals():** Comprueba si dos variables son iguales.
* **assertTrue():** Comprueba si una condición se cumple (es verdadera).
* **assertFalse():** Comprueba si una condición no se cumple (es falsa).
* **assertNull():** Comprueba si un objeto es null.
* **assertNotNull():** Comprueba que un objeto no sea null.
* **assertSame():** Comprueba si dos objetos son el mismo objeto.
* **assertNotSame():** Comprueba si dos objetos no son el mismo objeto.
* **assertThat():** Comprueba si el resultado cumple una condición dada.

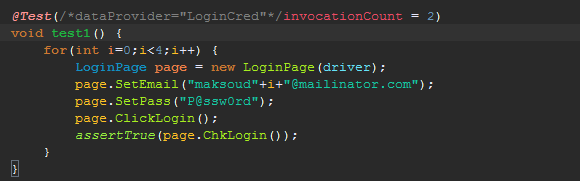
1. Indica para que sirven los siguientes tipos de pruebas:
   1. Funcionales
      1. Pruebas de usabilidad: Se utilizan para determinar cómo las personas interactúan con un determinado producto.
      2. Pruebas de rendimiento: Se utilizan para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo.
      3. Pruebas de stress: Se realiza para determinar la solidez de la aplicación en los momentos de carga extrema y ayuda a los administradores para determinar si la aplicación rendirá lo suficiente en caso de que la carga real supere a la carga esperada.
      4. Pruebas de seguridad: Sirven para encontrar fallas y vulnerabilidades en las aplicaciones, buscando disminuir el impacto de ataques a ellas y pérdida de información importante.
      5. Pruebas de compatibilidad: Se utilizan para garantizar el correcto funcionamiento de la aplicación en todos aquellos entornos en los que tiene que funcionar de forma correcta.
      6. Pruebas de portabilidad: Se usan para determinar el grado de facilidad o dificultad con el que un componente de software o una aplicación puede transferirse de manera eficaz y eficiente de un hardware, software u otro entorno operativo o de uso a otro.
   2. Funcionales
      1. Pruebas de regresión: Sirven para descubrir errores ([bugs](https://es.wikipedia.org/wiki/Error_de_software)), carencias de funcionalidad, o divergencias funcionales con respecto al comportamiento esperado del software, causados por la realización de un cambio en el programa.
      2. Pruebas de integración: Se utiilizan para probar que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupo.
      3. Pruebas del sistema: Se utilizan para asegurar que cada componente unitario o módulo (identificado en las pruebas unitarias) interactúe con otros componentes o módulos, tal como fue diseñado.
      4. Pruebas de humo (smoke test): Son una revisión rápida de un producto de software para comprobar que funciona y no tiene defectos evidentes que interrumpan la operación básica del mismo. Son pruebas que pretenden hacer una evaluación inicial de la calidad de un producto de software previo a una recepción formal
      5. Pruebas alfa y beta: Sirven para descubrir errores que parezca que sólo el usuario final puede descubrir.
      6. Pruebas de aceptación (validación por parte del cliente): Se utilizan para determinar si un software cumple con las necesidades y/o requerimientos de las empresas y sus usuarios.
2. Indica para que sirven las siguientes pruebas de código y un ejemplo de código en Java de cada una:
   1. Cubrimiento: Sirven para determinar la calidad del test que se lleve a cabo y para determinar las partes críticas del código que no han sido comprobadas y las partes que ya lo fueron.
   2. valores límite: están diseñadas para verificar los valores límite (maximos y minimos) en un rango.



* 1. clases de equivalencia: Su objetivo es identificar los casos de prueba usando un elemento de cada clase de equivalencia (entradas del programa que se clasifican en clases).



* 1. de bucles: Se utilizan para validar la construcción de bucles.

****

**WEBGRAFIA.**

[https://es.wikipedia.org](https://es.wikipedia.org/)

<https://www.youtube.com/watch?v=B0jf0ENdFsY>

<http://di002.edv.uniovi.es/~fanjul/isofg/curso2006-2007/descargas/ISOFG-PRUv2006.pdf>

<http://www.jtech.ua.es/j2ee/publico/lja-2012-13/sesion04-apuntes.html>

https://www.unab.edu.co/sites/default/files/MemoriasGrabadas/papers/capitulo7\_paper\_13.pdf

[https://es.qaz.wiki/wiki/Portability\_testing#:~:text=La%20prueba%20de%20portabilidad%20es,o%20de%20uso%20a%20otro](https://es.qaz.wiki/wiki/Portability_testing" \l ":~:text=La prueba de portabilidad es,o de uso a otro)

[https://rfolivares.wordpress.com/2010/09/08/pruebas-alfa-y-beta/#:~:text=La%20mayor%C3%ADa%20de%20los%20desarrolladores,el%20usuario%20final%20puede%20descubrir.&text=El%20cliente%20registra%20todos%20los,a%20intervalos%20regulares%20al%20desarrollador](https://rfolivares.wordpress.com/2010/09/08/pruebas-alfa-y-beta/" \l ":~:text=La mayoría de los desarrolladores,el usuario final puede descubrir.&text=El cliente registra todos los,a intervalos regulares al desarrollador)

https://www.adictosaltrabajo.com/2008/10/18/maven-cobertura/