**Instituto Politécnico Nacional**

**Centro de estudios Científicos y Tecnológicos**

**“Juan de Dios Bátiz”**

**Unidad de aprendizaje:** Introducción a Ingeniería de Pruebas

**Docente:** Rubén Enaín Pérez Montes

**Equipo 4 J&H:**

* Aguilar Sousa Iker Itzae
* Araujo Pérez Kitzia María
* Fuentes Jarquín Jesús Angel
* Hernández Ramírez Edher

**Grupo:** 5IM10

**JUnit**

**Fecha: 13/10/2022**

**JUnit**

JUnit es un marco de pruebas unitarias para Java. Es principalmente útil en el desarrollo basado en pruebas repetibles además de ser una instancia de marcos de pruebas unitarias de arquitectura XUnit.

Este framework provee al usuario de herramientas, clases y métodos para facilitar la realización de pruebas.

También, JUnit promueve la idea de probar y luego codificar: probando un poco, codificar un poco, probar un poco, codificar un poco; con el fin de aumentar la estabilidad del código y la productividad del programador, además de reducir a la larga el tiempo de depuración.

***MÉTODOS***

Una clase de prueba posee 4 métodos:

* Metodo setUp: se asignan los valores iniciales a las variables antes de la ejecución de la prueba
* Método tearDown: se llama después de cada test y se puede usar para liberar recursos
* Métodos Test: contienen las pruebas a realizar
* Métodos auxiliares

***ANOTACIONES***

Se llama anotaciones a caracteres especiales, que se han incluido en la versión 4, para intentar simplificar más la labor del programador. Se trata de palabras clave que se colocan delante de los métodos definidos antes y que indican a las librerías JUnit instrucciones concretas.

* @Before
  + Significa que se ejecutara antes de cada método.
* @BeforeClass
  + Significa que se ejecutara antes de todos los métodos.
* @After
  + Significa que se ejecutara después de cada método.
* @AfterClass
  + Significa que se ejecutara después de todos los métodos.
* @Test
  + La annotation @Test es simplemente para identificar que el método sera un método a testear.
* @Ignore
  + La annotation @Ignore sirve para avisarle al JUnit que ignore este método, esta anotación se utiliza generalmente cuando por alguna razón hemos modificado el código y estamos trabajando en el test. Puede ser que todavía nosotros no hallamos incluido código o que dependemos que algún desarrollador termine algo.
* @Test(timeout)
  + La annotation @Test(timeout) sirve para decirle a JUnit que si el método tarda más del tiempo esperado el mismo tiene que fallar, esto es ideal para verificar procesos con demora aceptable. El tiempo se debe asignar en milisegundos. En el caso que el test no termine en el tiempo deseado será marcado como fallido.
* @Test(expected)
  + La annotation @Test(expected) sirve para avisar que el test debe lanzar algún tipo de excepción. En el caso que no se cumpla la condición el test será marcado como fallido.
* @RunWith
  + La anotación @RunWith se utiliza para indicar que dicha clase se correrá como una extensión o tipo de otra clase.
* @Suite.SuiteClasses
  + La annotation @Suite.SuiteClasses se utiliza para indicar que un Test Suite correrá los siguientes test.

***FUNCIONES DE ACEPTACIÓN/RECHAZO***

Una vez hemos creado las condiciones para probar que una funcionalidad concreta funciona es necesario que un validador nos diga si estamos obteniendo el resultado esperado o no. Para esta labor se definen una lista de funciones (incluidas en la clase Assert) que se pueden ver detalladas en el javadoc de JUnit.

* assertArrayEquals
  + Comprueba si 2 arrays son iguales. Devuelve assertionError en caso de no serlo.
* assertEquals
  + Comprueba que 2 valores de tipo numérico sean iguales. Devuelve assertionError en caso de no serlo.
* assertTrue
  + Comprueba que la condición se cumple. Devuelve assertionError si no se produce el resultado esperado.
* Fail
  + Devuelve una alerta informando que el test ha fallado.

***CREACIÓN DE UNA CLASE DE PRUEBA***

**Paso 1.- Crear un proyecto con la clase que queremos probar.**

Abrimos Eclipse, creamos un proyecto JUnitLab y a continuación la clase Java que queremos probar.

Ejemplo:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteUna instancia de esta clase representa una suscripción a una publicación. La variable precio contiene el precio de la suscripción. El precio recoge el total en euros y céntimos. El periodo de suscripción se representa en meses y se recoge en la variable periodo.

**Paso 2.- Crear una clase de prueba.**

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamentePara crear una clase de prueba, nos posicionamos sobre la clase que queremos probar, pulsamos el botón derecho y seleccionamos la opción “New”>>”JUnit Test Case” .

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteEn el cuadro que aparece pulsamos el botón Next, y en la última pantalla aparecerán los métodos de la clase sobre la que queremos crear un caso de prueba, seleccionamos el método a probar y pulsamos finish.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteLa herramienta crea una clase de prueba, donde incluye las librerías de Junit necesarias para su ejecución, así como un método de prueba vacío donde deberemos escribir nuestro caso de prueba.

**Paso 3.- Crear los casos de prueba.**

Interfaz de usuario gráfica, Aplicación

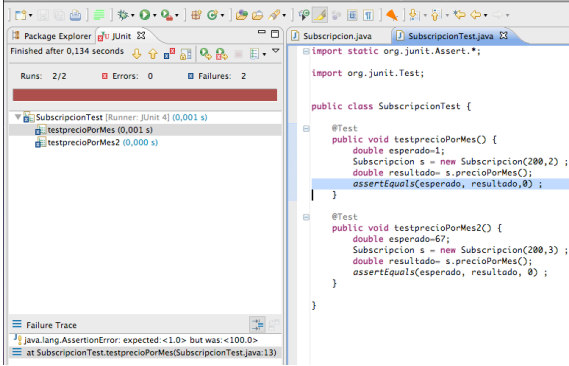
Descripción generada automáticamenteComo ejemplo, podemos rellenar la clase anterios con este código:

Todo caso de prueba se compone básicamente de 3 partes:

1. En una variable se indica cuál es el valor esperado por el método que queremos comprobar.
2. Se ejecuta el método que queremos probar con los datos de entrada adecuados y se guarda el resultado en otra variable.
3. Se comprueba la relación entre el valor esperado y el resultado del método, a través de una aserción. En este caso la aserción assertEquals indica que el caso de prueba es correcto sólo si el valor de las variables esperado y resultado son iguales.

**Paso 4.- Ejecutar paso de prueba.**

Se ejecutan las clases de prueba, las cuales son una serie de tests en cada clase del código.



Si hay fallas, estas salen a flote, indican dónde y por qué fallaron.

Los errores que las clases de prueba no se limitan a un solo tipo, Pueden ser como tal un error en el programa, un error de definición, o un error de lógica.

Las clases de pruebas de clases permiten modificar el código para en un siguiente test, confirmar si ese era el origen del problema o no.

Por ejemplo:



Es decir, que esperaba 1.0 y ha recibido como respuesta 100.0. En este punto podemos observar que el valor esperado no estaba correctamente definido, y que en vez de un error en el programa, es un error de la definición del caso de prueba.

**Paso 5.- Medición de la cobertura del código.**

La efectividad de los tests se mide en su cobertura.

La cobertura se refiere a cuántas líneas leen los test respecto al total. Si el test pasa por el 100% no garantiza que no hayan errores, pero sigue siendo un testeo riguroso.

El software que ayuda a este tipo de problemas puede medir la cobertura de estos tests. Esta medición de cobertura ocurre cuando se hacen las pruebas de caja blanca (que usa el código fuente). Este tipo de software es muy útil, ya que permite correr las clases de prueba varias veces, y poco a poco se cubre más código el cual se considerará dentro de los próximos testeos. La idea es llegar al 100% de cobertura.

BIBLIOGRAFÍA

* Java, E. L. O. de. (2022, 13 octubre). JUnit - Anotaciones / Annotation JUnit. Recuperado 11 de octubre de 2022, de http://java-white-box.blogspot.com/2014/06/junitanotacionesAnnotationJUnit.html?m=1
* JUnit | Marco de Desarrollo de la Junta de Andalucía. (s. f.). Recuperado 11 de octubre de 2022, de <https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/248>
* JUnit Tutorial | Testing Framework for java - javatpoint. (s. f.). www.javatpoint.com. Recuperado 11 de octubre de 2022, de <https://www.javatpoint.com/junit-tutorial>
* López, J. L. (2022, 12 julio). Pruebas unitarias en Java con JUnit. Profile Software Services. Recuperado 11 de octubre de 2022, de <https://profile.es/blog/pruebas-unitarias-con-junit/>
* OpenCourseWare. (2018, octubre). Pruebas en eclipse con JUnit y EclEmma. https://ocw.ehu.eus/. Recuperado 12 de octubre de 2022, de <https://ocw.ehu.eus/pluginfile.php/51924/mod_resource/content/3/lab1-JUnit.pdf>
* ¿Qué es jUnit? (2012, 9 julio). La paradita. Recuperado 11 de octubre de 2022, de <https://elrincondeaj.wordpress.com/2012/07/09/junit/>