UD04: Junit en Netbeans

1. Introducción.

Las pruebas de software son parte esencial del ciclo de desarrollo. La elaboración y mantenimiento de unidad, pueden ayudarnos a asegurar que los los métodos individuales de nuestro código, funcionan correctamente. Los entorno de desarrollo, integran frameworks, que permiten automatizar las pruebas.

En el caso de entornos de desarrollo para Java, como NetBeans y Eclipse, nos encontramos con el framework JUnit. JUnit es una herramienta de automatización de pruebas que nos permite de manera rápida y sencilla, elaborar pruebas. La herramienta nos permite diseñar clases de prueba, para cada clase diseñada en nuestra aplicación. Una vez creada las clases de prueba, establecemos los métodos que queremos probar, y para ello diseñamos casos de prueba. Los criterios de creación de casos de prueba, pueden ser muy diversos, y dependerán de lo que queramos probar.

Una vez diseñados los casos de prueba, pasamos a probar la aplicación. La herramienta de automatización, en este caso Junit, nos presentará un informe con los resultados de la prueba (imagen anterior). En función de los resultados, deberemos o no, modificar el código.

Los entornos de desarrollo más extendidos, que se utilizan para implementar aplicaciones Java, tales como NetBeans o Eclipse, incorporan un plugin para trabajar con Junit Junit nos va a servir para realizar pruebas unitarias de clases escritas en Java, dentro de un entorno de pruebas. Es un framework con muy pocas clases fácil de aprender y de utilizar.

Una vez que hemos diseñado nuestra aplicación, y la hemos depurado, procedemos a probarla. En el caso del ejemplo, disponemos de una clase, de nombre cVector, donde se han definido una serie de métodos.

El objetivo va a ser el diseño y ejecución de algunos casos de prueba.

2. Inicio de Junit

Para iniciar Junit, seleccionada en la ventana de proyectos la clase a probar, abrimos el menú contextual y seleccionamos Herramientas > Crear pruebas Junit.

Nos aparece un formulario donde nos da a elegir entre JUnit 3.x y JUnit 4.x. Son las dos versiones de JUnit disponibles en NetBeans 6.9.1. En nuestro caso, elegimos JUnit 3.x.

Puesto que vamos a probar la clase CVector, por convenio es recomendable llamar a la clase de prueba CVectorTest. Esta clase se va a insertar en un nuevo paquete de nuestro proyecto, denominado Paquete de prueba.

Como se aprecia en el formulario, JUnit va a generar los métodos que aparecen seleccionados. En nuestro caso lo vamos a dejar tal cual, aunque luefo van a ser modificados en el código.

Al pulsar el botón Aceptar nos aparecen un nueva clase de nombre CVectorTest, que contiene los métodos que estaban seleccionados en el formulario anterior, con un código prototipo. Es en ese código en el que el programador creará sus casos de prueba.

El diseño de los casos de prueba, requiere que se establezcan criterios que garanticen que esa prueba tiene muchas probabilidades de encontrar algún error no detectado hasta el momento.

3. Casos de prueba

En primer lugar, vamos a ver la función de los Inicializadores y Finalizadores. El método SetUp y el método tearDown, se utilizan para inicializar y finalizar las condiciones de prueba, como puede ser la creación de un objeto, inicialización de variables, etc. En algunos casos, no es necesario utilizar estos métodos, pero siempre se suelen incluir.

El método setUp es un método de inicialización de la prueba y se ejecutan antes de cada caso de prueba, en la clase de prueba. Este método no es necesario para ejecutar pruebas, pero si es necesario para inicializar algunas variables antes de iniciar la prueba.

El método tearDown es un método finalizador de prueba, y se ejecutará después de cada test en la clase prueba. Un método finalizador no es necesario para ejecutar las pruebas, pero si necesitamos un finalizador para limpiar algún dato que fue requerido en la ejecución de los casos de prueba.

En segundo lugar, es necesario conocer las aserciones. Los método assertXXX(), se utilizan para hacer las pruebas. Estos métodos, permiten comprobar si la salida del método que se está probando, concuerda con los valores esperados. Las principales son:

- `assertTrue()` evalúa una expresión booleana. La prueba pasa si el valor de la expresión es true.
- `assertFalse()` evalúa una expresión booleana. La prueba pasa si el valor de la expresión es false.
- `assertNull()` verifica que la referencia a un objeto es nula.
- `assertNotNull()` verifica que la referencia a un objeto es no nula.
- `assertSame()` compara dos referencias y asegura que los objetos referenciados tienen la misma dirección de memoria. La prueba pasa si los dos argumentos son el mismo objeto o pertenecen al mismo objeto.
- `assertNotSame()` Compara dos referencias a objetos y asegura que ambas apuntan a diferentes direcciones de memoria. La prueba pasa si los dos argumentos suplidos son objetos diferentes o pertenecen a objetos distintos.
- `assertEquals()` Se usa para comprobar igualdad a nivel de contenidos. La igual de tipos primitivos se compara usando “==”, la igual entre objetos se compara con el método equals(). La prueba pasa si los valores de los argumentos son iguales.
- `fails()` causa que la prueba falle inmediatamente. Se puede usar cuando la prueba indica un error o cuando se espera que el método que se está probando llame a una excepción.

En este punto, nos disponemos a diseñar los métodos que necesitemos para los casos de prueba. Ejemplos de casos de prueba pueden ser:

```
1 public void test_posicion (int pos){
2 vectorTest.insertar(3);
3 vectorTest.insertar(5);
4 try{
5 assertTrue (vectorTest.posicion(7)==1);
6 }catch (Exception e){
7 fail(“El método test_posicion no funciona”);
```

```
}
}
/**
* Test del método ordenar_vector, de la clase CVector. Si ordena bien,
 la
comparación entre el
* vector vOrdenado y prueba, debe ser cierta y la prueba un éxito
*/
public void test_ordenar_vector(){
int [] vOrdenado = {7,36,45,52,85};
int [] prueba = new int[5];
try{
vectorTest.insertar(36);
vectorTest.insertar(85);
vectorTest.insertar(45);
vectorTest.insertar(52);
vectorTest.insertar(7);
vectorTest.ordenar_vector();
for (int i=0;i<5;i++)
prueba[i]=vectorTest.posicion(i);
assertEquals(vOrdenado,prueba);
}catch (Exception e){
fail("El método vector _lleno no funciona");
}
}
/**
* Test del método vector_lleno(), de la clase cVector. Debería devolver true, al
* insertar 100 elementos
*/
public void test_vector_lleno(){
try{
for(int i=0;i<100;i++)
vectorTest.insertar(i);
assertTrue(vectorTest.vector_lleno());
}
catch(Exception e){
fail(·El método vector_lleno no funciona");
}
}
```

Estos tres métodos intentan probar algunos métodos de la clase CVector. Para ello, teniendo seleccionado el proyecto, accederemos al menú contextual y pulsamos la opción Probar.

Como se puede comprobar, la prueba sobre el método vector_lleno ha sido un éxito, pero ha fallado la prueba sobre ordenar_vector. Con esta información, debemos comprobar que el caso de prueba está diseñado, en cuyo caso, lo que se ha encontrado es un error en el diseño del método ordenar_vector, y hay que rediseñarlo. La ventaja de utilizar herramientas automatizadas, es que se facilita la regresión, ya que tenemos diseñado el caso de prueba para el método, así que una vez rediseñado, podemos volver a probarlo con el mismo caso de prueba.

Para saber más
En el siguiente enlace nos encontramos con un ejempo completo de prueba de la unidad con NetBeans Creación de Casos de Prueba en NetBeans con Junit

4. Fuentes de información

- Wikipedia
- Code&Coke (Fernando Valdeón)
- Apuntes IES El Grao (Mª Isabel Barquilla?)
- Apuntes IOC (Marcel García)
- Apuntes José Luis Comesaña
- Apuntes IES Luis Vélez de Guevara 17-18 (José Antonio Muñoz Jiménez)