JUnit 4

Framework para pruebas de unidades en Java

Programación I

Grado en Ingeniería Informática

MDR, JCRdP y JDGD

Pruebas de unidades

- **Deben ser:**
  - Automatizables (sin intervención humana)
  - Lo más completas posibles (probar todo el código)
  - Repetibles o reutilizables
  - Independientes de otras pruebas
  - Profesionales como el resto del código

Pruebas de unidades

- **Ventajas**
  - Facilita el cambio
  - Previas a pruebas de integración, integración continua
  - Definen la funcionalidad
  - Separación de la interfaz y la implementación
- **Tienen limitaciones**
  - No se detectan todos los defectos

Qué es JUnit

- Es un paquete de clases para hacer pruebas de unidades en Java
- Código abierto http://www.junit.org desarrollado por Erich Gamma y Kent Beck
- Se basa en "**una función un caso de prueba**". Se escribe una función para cada caso a probar
- Proceso de una prueba:
  1. Se ejecuta el caso a probar
  2. Se comparan los resultados con los esperados

Características de JUnit 4.x

- La comparación se realiza mediante métodos estáticos de la clase "org.junit.Assert", a los que se les pasa la comparación del resultado obtenido y el esperado o, los dos valores por separado, para comprobarlos.
  - assertTrue(Boolean), assertFalse(Boolean)
  - assertNull(Object)
  - assertEquals(Object, Object)
  - assertArrayEquals(Object, Object)
  - fail(): Genera un error directamente
  - Se puede añadir un mensaje a mostrar en caso de error

Características de JUnit 4.x

```
import org.junit.*;
import static org.junit.Assert.*;
…
   @Test
   public void pruebaConstructor() {
         Fraccion f1 = new Fraccion(1,2);
         Fraccion f2 = new Fraccion(30,6);
         …
         assertTrue(f1.dameNumerador()  == 1);
         assertEquals("Constructor de (30/6)", 5, f2.dameNumerador());
         …
   }
```

Características de JUnit 4.x

- Para establecer y parametrizar qué métodos realizan las pruebas se usan **anotaciones**
- Las anotaciones se establecen precediendo el elemento a cualificar (clase, método, atributo)
- El establecimiento de una anotación comienza con @ seguida de su nombre y, opcionalmente, entre paréntesis, parámetros. Ejemplos:

  @Deprecated: indica que es obsoleto (aviso al compilar)

  @Override: redefine un método

  @SuppressWarnings: para que el compilador no dé avisos

Características de JUnit 4.x

- Las anotaciones de la JUnit se encuentran en el paquete **org.junit**
- Se escribe una clase con la pruebas
- Un método con cada caso a probar (tienen que ser públicos, sin parámetros y devolver **void**)
- La anotación **@Test** indica que un método contiene pruebas a ejecutar
- Es posible indicar que una prueba debe lanzar una excepción o limitar su tiempo de ejecución
  - @Test(expected = IndexOutOfBoundsException.class)
  - @Test(timeout = 1000) Tiempo máximo de ejecución de un caso
- Para evitar que se ejecuten casos de prueba durante el proceso de prueba se usa la anotación **@Ignore**

Características de JUnit 4.x

- Con frecuencia, en cada caso de prueba se usan objetos inicializados a datos predefinidos
- En vez de replicar en cada caso código para usar estos **objetos de prueba** (Fixture), se añaden atributos a la clase que contiene los métodos de prueba
- Para gestionar los objetos comunes de prueba se usa:
  - **@Before**: función que inicializa los atributos. Se ejecuta antes de ejecutar cada método @Test
  - **@After**: función que los desinicializa. Se ejecuta después de cada método @Test

Características de JUnit 4.x

- En caso de que se usen recursos que no es necesario o es muy costoso inicializar, antes de ejecutar cada @Test, se pueden establecer métodos para inicializar estos elementos una sola vez:
  - **@BeforeClass** and **@AfterClass** definen funciones "static" que se ejecutan una vez, antes y después de todas las pruebas

Características de JUnit 4.x

- Para ejecutar desde la línea de comandos pruebas de una o varias clases se usa:

```
java org.junit.runner.JUnitCore clase1 clase2 ...
```

- Para ejecutar desde el programa una o varias clases se usa:

```
String[] clases={"clase1","clase2", ...};
org.junit.runner.JUnitCore.main(clases);
```

- Para ejecutar desde el programa y obtener información manejable programáticamente:

```
org.junit.runner.JUnitCore.runClasses(clase1.class,...);
```

```
import org.junit.*;
import org.junit.runner.JUnitCore;
import static org.junit.Assert.*;
public class TestFraccion {
    private Fraccion f1;
    private Fraccion f2;

    @Before
    public void inicializa() {
      f1 = new Fraccion(1,1);
      f2 = new Fraccion(60,24);
    }

    @Test
    public void pruebaConstructor() {
      assertEquals(1, f1.dameNumerador());
      assertEquals(1,f1.dameDenominador());
      assertEquals(5, f2.dameNumerador());
      assertEquals(2,f2.dameDenominador());
    }

     @Test
    public void pruebaSuma() {
      Fraccion f3 = f1.suma(f2);
      assertEquals(7, f3.dameNumerador());
      assertEquals(2, f3.dameDenominador());
      …
    }
    …
    static public void main(String[] args){
      JUnitCore.runClasses(TestFraccion.class);
    }
}
```

Referencias

- **JUnit** http://www.junit.org
- **JUnit tutorial** http://www.linux.ie/articles/tutorials/junit.php
- **JUnit (wikipedia)** http://es.wikipedia.org