JUnit 3 vs. JUnit 4

David Alonso Ríos

Facultad de Informática
Universidade da Coruña

2008/2009

Novedades de JUnit 4

- Utiliza anotaciones de Java 5 para:
  - Identificar los tests, en vez de usar convenciones de nombres.
  - Tratar excepciones, en vez de capturarlas manualmente.
  - Limitar el tiempo.
  - Ignorar tests.
  - Inicializar y limpiar el entorno, en vez de usar setUp() y tearDown().
- Cambian las aserciones.
- Desaparece la interfaz gráfica de tests.

Identificar los tests (i)

JUnit 3:

Los métodos deben empezar por “test”:

```
import junit.framework.TestCase;
public class MayorTest extends TestCase {
      ...
   public void testOrden() {
      ...
  }
}
```

Identificar los tests (ii)

JUnit 4:

Usa anotaciones:

```
import org.junit.Test;
import junit.framework.TestCase;
public class MayorTest extends TestCase {
...
  @Test public void miTestOrden() {
    ...
  }
}
```

Identificar los tests (iii)

JUnit 4:

No es necesario extender TestCase. Podemos importar (los métodos de) la clase Assert:

```
import static org.junit.Assert.*;
public class MayorTest {
...
  @Test public void primerTestOrden() { ...
  }
}
```

Tratar excepciones (i)

JUnit 3:

Manualmente:

```
public void testVacia() {
    try {
        Ejemplo.calcularMayor(new int[] {});
        fail("Should have thrown an exception");
    } catch (RuntimeException e) {
        ...
    }
}
```

Tratar excepciones (ii)

JUnit 4:

Podemos indicar qué excepción esperamos. Si no se produce, el test fallará. (Si queremos hacer algo más específico con la excepción, siempre podemos capturarla manualmente)

```
@Test(expected=RuntimeException.class)
public void testVacia() {
   Ejemplo.calcularMayor(new int[] {});
}
```

Limitar el tiempo

JUnit 4:

Podemos anotar un test para que falle si tarda más de x milisegundos:

```
@Test(timeout=500) public void
  testRecuperarDatos() {
   ...
}
```

Ignorar tests

JUnit 4:

Podemos anotar un test para que no se ejecute de momento. El sistema nos avisará de que ha sido ignorado.

```
@Ignore public void testMuylento() {
   ...
}
```

Inicializar y limpiar el entorno (i)

JUnit 3:

```
protected void setUp() {
    ...
}
protected void tearDown() {
    ...
}
```

Inicializar y limpiar el entorno (ii)

JUnit 4:

Podemos anotar uno o varios métodos para que se ejecuten antes / después de cada método de la clase test:

```
@Before protected void crearEntorno() {
    ...
}
@After protected void deshacerEntorno() {
    ...
}
```

Inicializar y limpiar el entorno (iii)

JUnit 4:

También podemos anotar métodos para que se ejecuten antes / después del conjunto de métodos de la clase test:

```
@BeforeClass protected void crearEntorno() {
    ...
}
@AfterClass protected void deshacerEntorno() {
    ...
}
```

Cambios en las aserciones (i)

JUnit 4.1: Añade igualdad de arrays:

- `public static void assertEquals(`**`Object[]`** `esperado,` **`Object[]`** `real)`
- `public static void assertEquals(String mensaje, Object[] esperado, Object[] real)`

JUnit 4.3.1: Redefine igualdad de arrays:

- **`assertArrayEquals`**`(java.lang.Object[] experado, java.lang.Object[] real)`
- **`assertArrayEquals`**`(java.lang.String mensaje, Object[] esperado, Object[] real)`
- Añade varios métodos para arrays de numéricos.

Cambios en las aserciones (ii)

JUnit 4.5: Cambios en la igualdad de numéricos:

- `assertEquals(double esperado, double real)` está deprecated.
- Usar `assertEquals(double esperado, double real, double epsilon)`

Ejemplo JUnit 3 (i)

```
import calc.Calculadora;
import junit.framework.TestCase;

public class CalculadoraTest extends TestCase {

  private static Calculadora Calculadora =
                new Calculadora();
```

Ejemplo JUnit 3 (ii)

```
@Override
protected void setUp() {
    Calculadora.limpiar();
}


public void testSumar() {
  Calculadora.sumar(2);
  Calculadora.sumar(2);
  assertEquals(Calculadora.obtenerResultado(),
4);
}
```

Ejemplo JUnit 3 (iii)

```
public void testRestar() {
   Calculadora.sumar(5);
   Calculadora.restar(3);
   assertEquals(Calculadora.obtenerResultado(),
2);
 }


public void testDividir() {
   Calculadora.sumar(10);
   Calculadora.dividir(2);
   assertEquals(Calculadora.obtenerResultado(),
5);
 }
```

Ejemplo JUnit 3 (iv)

```
  public void testDividirEntreCero()  {
     try {
        Calculadora.dividir(0);
        fail();
     } catch (ArithmeticException e) {
     }
  }

}
```

Ejemplo JUnit 4 (i)

```
import calc.Calculadora;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculadoraTest {

  private static Calculadora Calculadora =
              new Calculadora();

  @Before
  public void limpiarCalculadora() {
    Calculadora.limpiar();
  }
```

Ejemplo JUnit 4 (ii)

```
@Test
public void sumar() {
  Calculadora.sumar(2);
  Calculadora.sumar(2);
  assertEquals(Calculadora.obtenerResultado(),
4);
}

@Test
public void restar() {
  Calculadora.sumar(5);
  Calculadora.restar(3);
  assertEquals(Calculadora.obtenerResultado(),
2);
}
```

Ejemplo JUnit 4 (iii)

```
@Test
public void dividir() {
  Calculadora.sumar(10);
  Calculadora.dividir(2);
  assertEquals(Calculadora.obtenerResultado(),
5;
}

@Test(expected = ArithmeticException.class)
public void dividirEntreCero() {
  Calculadora.dividir(0);
}
```

Ejemplo JUnit 4 (iv)

```
  @Ignore("todavia no implementado")
  @Test
  public void multiplicar() {
    Calculadora.sumar(3);
    Calculadora.multiplicar(3);
    assertEquals(Calculadora.obtenerResultado(),
  9);
  }
}
```