Pruebas de programas Java mediante JUnit

ENTORNOS DE DESARROLLO

Marcos Fernández Sellers

IES "Valle del Jerte"
PLASENCIA
BILINGUAL SECTION

Informática y Comunicaciones
IES Valle del Jerte
Plasencia
informatica.iesvalledeljerteplasencia.es

El framework JUnit

- JUnit es un *framework* para automatizar las pruebas de programas Java
- Escrito por Erich Gamma y Kent Beck
- Open Source, disponible en http://www.junit.org
- Adecuado para el Desarrollo dirigido por las pruebas (*Test-driven development)*

El framework JUnit

- Consta de un conjunto de clases que el programador puede utilizar para construir sus casos de prueba y ejecutarlos automáticamente
- Los casos de prueba son realmente programas Java. Quedan archivados y pueden ser reejecutados tantas veces como sea necesario

Un ejemplo sencillo

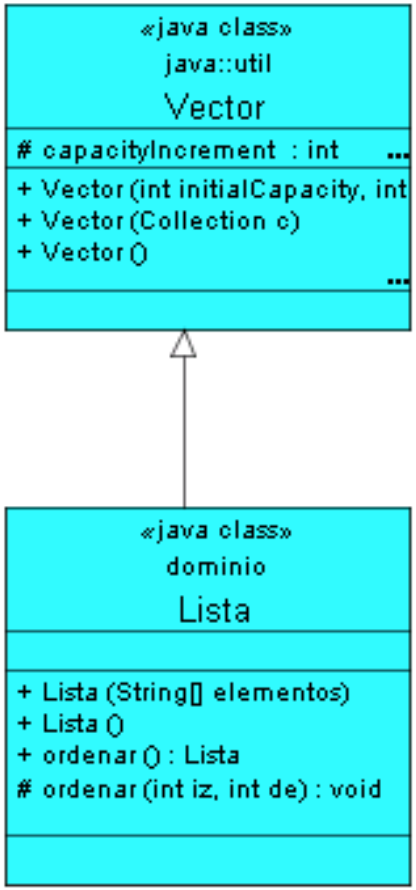
```
package dominio;
import java.util.Vector;

public class Lista extends Vector {
   public Lista() { ... }
   public Lista(String[] elementos) {...}
   public Lista ordenar() {...}
   protected void ordenar(int iz, int de) {
      ...
   }
}
```

← Representa una lista ordenable de forma creciente.

Se ordena llamando al método público *ordenar()*, que llama a su vez a *ordenar(0, size()-1)*

Un ejemplo sencillo



Un posible caso de prueba es el siguiente:

```
String[] e3={"e", "d", "c", "b", "a"};
Lista reves=new Lista(e3);
Lista derecha=reves.ordenar();
```

...y el resultado esperado:

```
"a", "b", "c", "d", "e"
```

Un ejemplo sencillo

```
String[] e3={"e", "d", "c", "b", "a"};
Lista reves=new Lista(e3);
Lista derecha=reves.ordenar();
```

Si *derecha* es igual al resultado esperado, entonces el caso de prueba ha sido superado:

```
{"a", "b", "c", "d", "e"}
```

Un ejemplo sencillo

Construyamos manualmente un objeto *expected* y comparémoslo con el obtenido:

```
String[] e3={"e", "d", "c", "b", "a"};
Lista reves=new Lista(e3);
Lista derecha=reves.ordenar();
Lista expected={"a", "b", "c", "d", "e"};
if (derecha.equals(expected))
   ResultadoCorrecto();
else
   ResultadoIncorrecto();
```

El framework JUnit (II)

El ejemplo anterior (*obtained* frente a *expected*) es una idea fundamental de Junit.

Ocurre que:

- JUnit nos va a permitir mantener de forma separada los casos de prueba
- JUnit permite ejecutarlos (y reejecutarlos) de forma automática
- Nos permite construir “árboles de casos de prueba” (*suites*)

El framework JUnit (II)

Para el ejemplo anterior:

```
public void testOrdenarReves() {
String[] ex={"a", "b", "c", "d", "e"};
Lista expected=new Lista(ex);

String
listaA

this.assertEquals(expected, listaAlReves.ordenar());
}
```

Construcción manual del objeto esperado

El framework JUnit (II)

Para el ejemplo anterior:

```
public void testOrdenarReves() {
 String[] ex={"a", "b", "c", "d", "e"};
 Lista expected=new Lista(ex);

 String[] e3={"e", "d", "c", "b", "a"};
 listaAlReves=new Lista(e3);
 this.a
}
```

Construcción manual del objeto obtenido haciendo uso de los métodos de la clase que estamos probando

El framework JUnit (II)

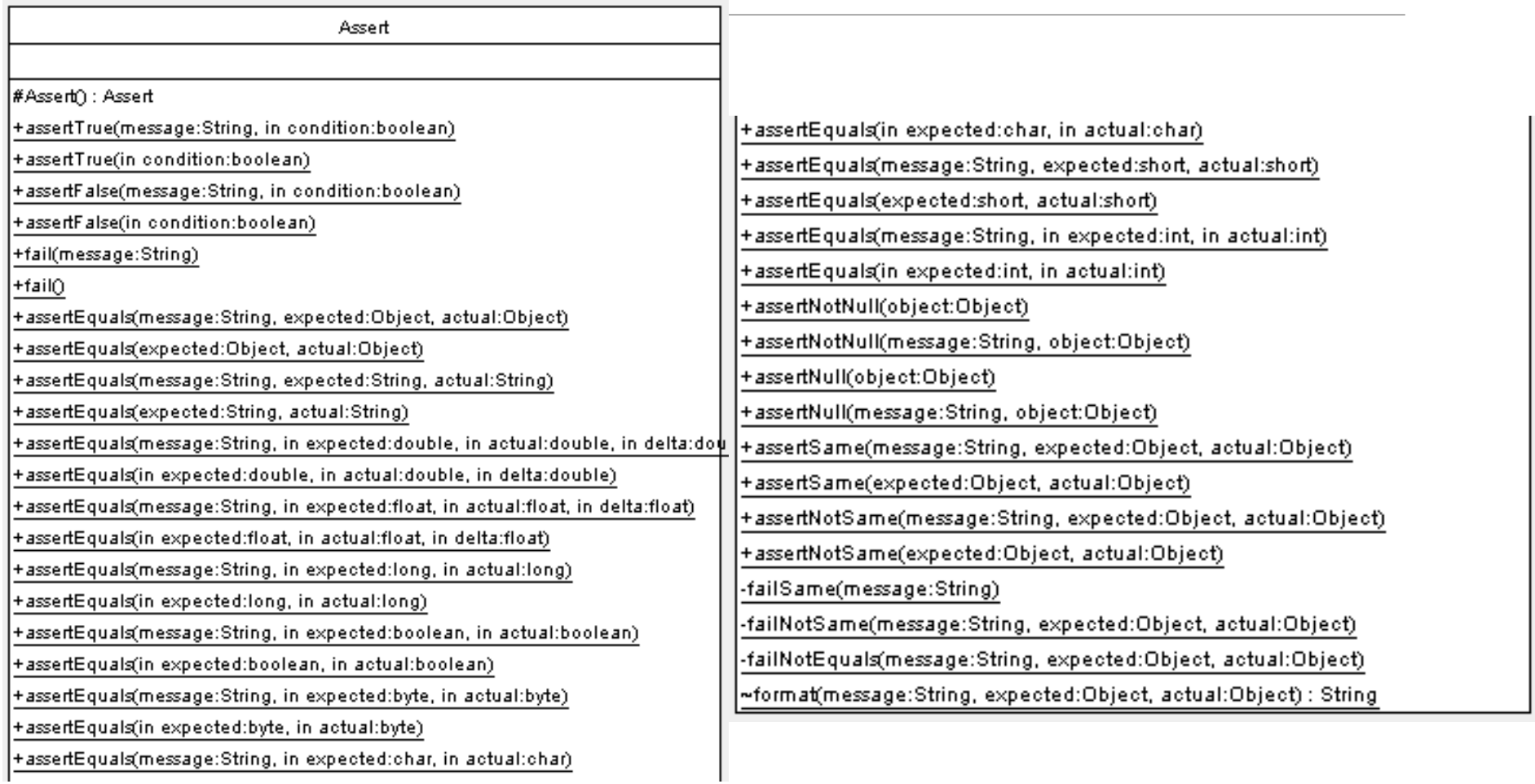
Para el ejemplo anterior:

```
public void testOrdenarReves() {
 String[] ex={"a", "b", "c", "d", "e"};
 Lista expected=new Lista(ex);

 String[] e3={"e", "d", "c", "b", "a"};
 lista
 this.assertEquals(expected, listaAlReves.ordenar());
}
```

Comparación de ambos objetos haciendo uso de las funcionalidades suministradas por JUnit

Clases fundamentales: Assert



Anotaciones

Junit está basado en anotaciones para añadir metadatos al código

- @Test → Indica que el siguiente método es un Test
- @Before → Ejecuta el siguiente método antes de cada test
- @After → Ejecuta el siguiente método después de cada test

Anotaciones

Junit está basado en anotaciones para añadir metadatos al código

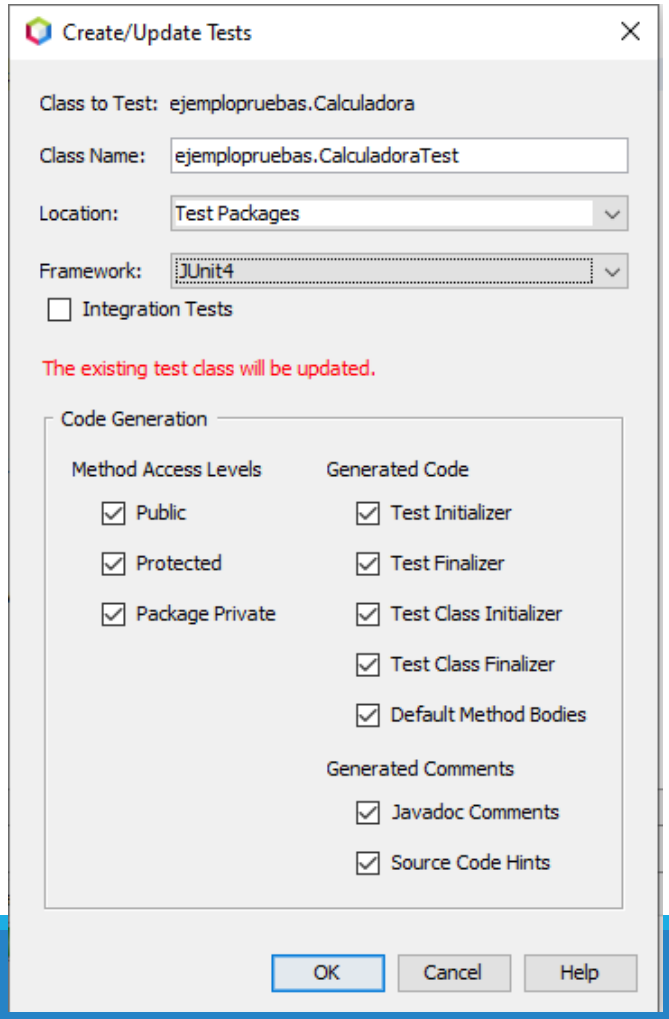
- @BeforeClass → Ejecuta el siguiente método al inicio de la clase test
- @AfterClass → Ejecuta el siguiente método al final de la clase test

Creación de Clases Test

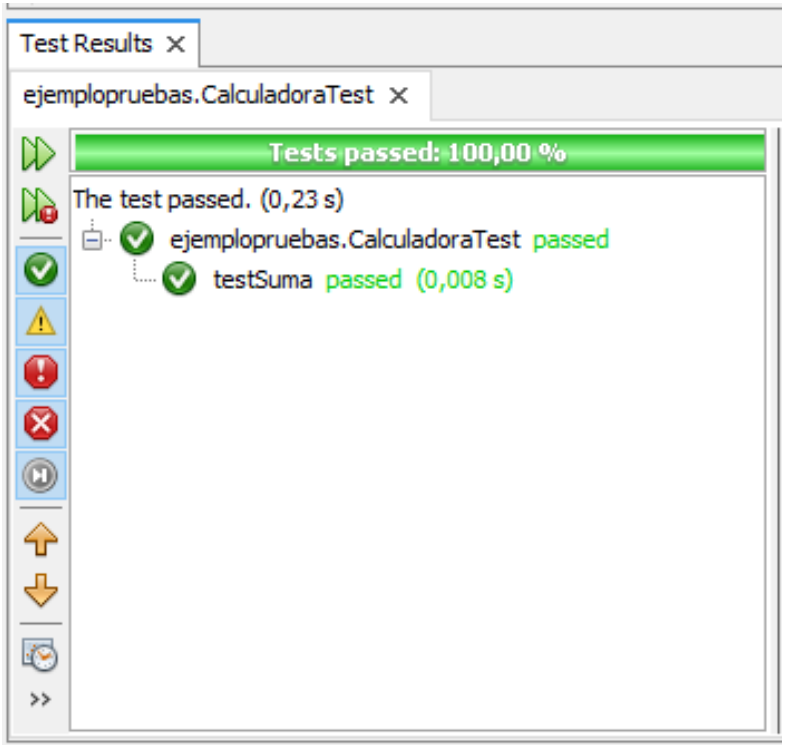
Botón derecho sobre la clase Java → Tools → Create/Update Tests

NOTA: JUnit 5 no funciona para proyectos Java ANT. Los ejemplos realizados en clase se elaborarán con JUnit 4

Creación de Clases Test



Visualización de resultados



Pruebas de excepciones (*fail)*

- Igual que es necesario comprobar cómo se comporta el programa en situaciones idóneas, es también importante probarlo en situaciones en que se producen errores.
- Es decir, que a veces el comportamiento correcto de nuestro programa consisten en se produzca un error

Pruebas de excepciones (*fail)*

Podemos desear que *ordenar()* dé un error cuando la lista esté vacía:

```
public Lista ordenar() throws Exception {
   if (size()==0)
     throw new Exception("No se puede ordenar una lista vacía");
   ordenar(0, size()-1);
   return this;
 }
```

Pruebas de excepciones (*fail*)

Modificamos los dos métodos *test*

```
public void testOrdenarNula2() throws Exception {
    try
    {
      String[] ex=null;
      Lista expected=new Lista(ex);
      this.assertEquals(expected, listaNula2.ordenar());
      fail("Debería haberse lanzado una excepción");
    }
    catch (Exception e)
    {
      // Capturamos la excepción para que el caso no falle
    }
 }
```

Clases de prueba abstractas

Se pueden posponer las pruebas hasta que se tengan especializaciones concretas de la clase abstracta

Pero también puede construirse una clase de Test abstracta

Referencia JUnit

https://junit.org

Conclusiones

- Marco de pruebas semiautomático
- Automatiza las pruebas de regresión
- Los casos de prueba documentan el propio código fuente
- Adecuado para Desarrollo dirigido por las pruebas (TDD, Test-Driven Development)

Créditos Diapositivas: Macario Polo Usaola http://www.inf-cr.uclm.es/www/mpolo *Modificaciones y adaptaciones por Marcos Fernández Sellers*