Tecnología de la Programación

Junit

Departamento de Computación

Facultad de Informática
Universidade da Coruña

Curso 2006/2007

Junit

- Herramienta de validación para Java.
- Valida a nivel de componentes: clases, paquetes, . . .
- Usa aserciones (assert) para caracterizar la salida.
- Codificamos cada unidad de prueba como una clase.
- Desarrollado por Kent Beck y Erich Gamma.
- CLI y GUI.

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $$\texttt{Rational(1,3)} = \frac{1}{3}$$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (I)

- ¿ Cómo construirmos una prueba en Java ?
- Tenemos una clase `Rational` que representa fracciones.

  $\texttt{Rational(1,3)} = \frac{1}{3}$

- Propiedades importantes:
  - Identidad. ¿ $\frac{1}{3} = \frac{1}{3}$?
  - Representaciones diferentes. ¿ $\frac{2}{6} = \frac{1}{3}$?
  - Enteros. ¿ $\frac{3}{3} = 1$?
  - Desigualdad. ¿ $\frac{1}{3} \neq \frac{2}{3}$?

Rationale (II)

- Construimos una clase con los tests:

```
class RationalAssert {
  public static void main(String args[]) {
    assert new Rational(1,3).equals(new Rational(1,3));
    assert new Rational(2,6).equals(new Rational(1,3));
    assert new Rational(3,3).equals(new Rational(1,1));
    assert !new Rational(2,3).equals(new Rational(1,3));
  }
}
```

- Compilamos y ejecutamos el test.

```
$ javac -source 1.4 RationalAssert.java
$ java -ea RationalAssert
```

Rationale (II)

- Construimos una clase con los tests:

```
class RationalAssert {
  public static void main(String args[]) {
    assert new Rational(1,3).equals(new Rational(1,3));
    assert new Rational(2,6).equals(new Rational(1,3));
    assert new Rational(3,3).equals(new Rational(1,1));
    assert !new Rational(2,3).equals(new Rational(1,3));
  }
}
```

- Compilamos y ejecutamos el test.

```
$ javac -source 1.4 RationalAssert.java
$ java -ea RationalAssert
```

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Problemas

- Problemas de hacer los tests “a mano”:
  - Se para en el primer fallo.
  - Mezclamos la ejecución de los tests.
  - No hay grupos de test (testsuites).
  - No tenemos informes (¿ ha fallado algún test ? ¿ cuál ?)
- Solución: herramienta.

Tests en Junit (I)

- Los tests se organizan en *test cases*.
- *Test case* = conjunto de pruebas.
- Cada *test case* se implementa en una clase java que deriva de *junit.framework.TestClase*

```
public class RationalTest extends TestCase {
  public RationalTest(String name) {
    super(name);
  }
  ...
}
```

Tests en Junit (I)

- Los tests se organizan en *test cases*.
- *Test case* = conjunto de pruebas.
- Cada *test case* se implementa en una clase java que deriva de

junit.framework.TestClase

```
public class RationalTest extends TestCase {
  public RationalTest(String name) {
    super(name);
  }
  ...
}
```

Tests en Junit (I)

- Cada prueba se implementa en un método público cuyo nombre comienza por `test`.
- Para las aserciones usamos los método de Junit:

Método	Comprueba que
fail(*msg*)	Nada. Falla siempre
assertTrue(*msg*, b)	b es cierto
assertFalse(*msg*, b)	b es falso
assertEquals(*msg*, v1, v2)	v1 = v2
assertEquals(*msg*, v1, v2, e)	| v1 - v2 | $\leq$ e
assertNull(*msg*, obj)	obj es null
assertNonNull(*msg*, obj)	obj no es null
assertSame(*msg*, o1, o2)	o1 y o2 son el mismo objeto
asserNotSame(*msg*, o1, o2)	o1 y o2 no son el mismo objeto

Tests en Junit (I)

- Cada prueba se implementa en un método público cuyo nombre comienza por `test`.
- Para las aserciones usamos los método de Junit:

Método	Comprueba que
`fail(`*msg*`)`	Nada. Falla siempre
`assertTrue(`*msg*`, b)`	b es cierto
`assertFalse(`*msg*`, b)`	b es falso
`assertEquals(`*msg*`, v1, v2)`	v1 = v2
`assertEquals(`*msg*`, v1, v2, e)`	$\lvert v1 - v2 \rvert \leq e$
`assertNull(`*msg*`, obj)`	obj es null
`assertNonNull(`*msg*`, obj)`	obj no es null
`assertSame(`*msg*`, o1, o2)`	o1 y o2 son el mismo objeto
`asserNotSame(`*msg*`, o1, o2)`	o1 y o2 no son el mismo objeto

Tests en Junit (I)

- Volvemos al ejemplo de los números racionales.

```
public void testEquality() {

  assertTrue(new Rational(1,3).equals(new Rational(1,3)));
  assertTrue(new Rational(2,6).equals(new Rational(1,3)));
  assertTrue(new Rational(3,3).equals(new Rational(1,1)));
}

public void testNonEquality() {
  assertFalse(new Rational(2,3).equals(new Rational(1,3)));
}
```

- Usamos el GUI:

```
public static void main(String args[]) {
  String[] testCaseName = { RationalTest.class.getName() };
  junit.swingui.TestRunner.main(testCaseName);
}
```

Tests en Junit (I)

- Volvemos al ejemplo de los números racionales.

```
public void testEquality() {

  assertTrue(new Rational(1,3).equals(new Rational(1,3)));
  assertTrue(new Rational(2,6).equals(new Rational(1,3)));
  assertTrue(new Rational(3,3).equals(new Rational(1,1)));
}

public void testNonEquality() {
  assertFalse(new Rational(2,3).equals(new Rational(1,3)));
}
```

- Usamos el GUI:

```
public static void main(String args[]) {
  String[] testCaseName = { RationalTest.class.getName() };
  junit.swingui.TestRunner.main(testCaseName);
}
```

Tests en Junit (II)

- Compilar y ejecutar

```
javac -classpath .:/path_to/junit.jar RationalTest.java
java -classpath .:/path_to/junit.jar RationalTest
```

  - La interfaz de texto.

```
public static void main(String args[]) {
  ...
  junit.textui.TestRunner.main(testCaseName);
}
```

Tests en Junit (II)

- Compilar y ejecutar

```
javac -classpath .:/path_to/junit.jar RationalTest.java
java -classpath .:/path_to/junit.jar RationalTest
```

- La interfaz de texto.

```
public static void main(String args[]) {
   ...
   junit.textui.TestRunner.main(testCaseName);
}
```

Contexto de las pruebas

- Algunas pruebas necesitan ejecutarse en un estado determinado.
- Métodos de inicialización y finalización.

```
public class RationalTest extends TextCase {
  private Rational a_third;

  protected void setUp() {
    a_third = new Rational(1,3);
  }

  protected void tearDown() {
    a_third = null;
  }

  public void testEquality() {
    assertEquals(new Rational(1,3), a_third);
    assertEquals(new Rational(2,6), a_third);
...
}
```

Contexto de las pruebas

- Algunas pruebas necesitan ejecutarse en un estado determinado.
- Métodos de inicialización y finalización.

```
public class RationalTest extends TextCase {
  private Rational a_third;

  protected void setUp() {
    a_third = new Rational(1,3);
  }

  protected void tearDown() {
    a_third = null;
  }

  public void testEquality() {
    assertEquals(new Rational(1,3), a_third);
    assertEquals(new Rational(2,6), a_third);
...
}
```

Test Case Suites

- Las pruebas también se puede n ejecutar sin la interfaz de Junit.

```
TestResult result =
    (new RationalTest("testEquality")).run();
```

- Es más habitual ejecutar conjuntos de pruebas.

```
TestSuite suite = new TestSuite();
suite.addTest(new RationalTest("testEquality"));
suite.addTest(new RationalTest("testNonEquality"));
TestResult result = suite.run();
```

Test Case Suites

- Las pruebas también se puede n ejecutar sin la interfaz de Junit.

```
TestResult result =
    (new RationalTest("testEquality")).run();
```

- Es más habitual ejecutar conjuntos de pruebas.

```
TestSuite suite = new TestSuite();
suite.addTest(new RationalTest("testEquality"));
suite.addTest(new RationalTest("testNonEquality"));
TestResult result = suite.run();
```