Pruebas de programas Java mediante JUnit

Macario Polo Usaola
Grupo Alarcos
Escuela Superior De Informática
Universidad De Castilla-la Mancha
http://www.inf-cr.uclm.es/www/mpolo

Contenidos

- El framework JUnit (I)
- Un ejemplo sencillo
- El framework JUnit (II)
- El TestRunner
- Términos
- Instalación de JUnit
- Objetos Mock

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);

   Str
   lis


   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);


   thi
}
```

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"};
   listaAlReves=new Lista(e3);


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

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

El framework JUnit (II)

- Destaquemos algunos elementos:

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

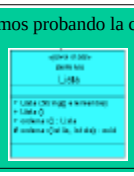
13

El framework JUnit (II)

- Destaquemos algunos elementos:

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

   String[] e3=
   listaAlReves=


   this.assertE                                    );
}
```

Estamos probando la clase Lista



14

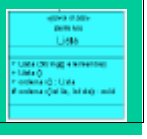
El framework JUnit (II)

- Destaquen

```
public void test
   String[] ex={"
   Lista expected

   String[] e3={"
   listaAlReves=new Lista(e3);


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

Estamos probando la clase Lista

- Lista(String[])
- Lista()
- ordenar()
- ordenar(int, int)



No tiene método "assertEquals(...)"

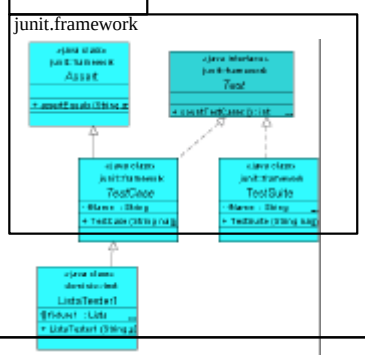
15

El framework JUnit (II)

- ¿Dónde está el código anterior?
- En una clase *ListaTester*, creada ex profeso para realizar las pruebas de *Lista*
- ListaTester especializa a la clase *TestCase* definida en JUnit
- En TestCase está definido el método *assertEquals* antes mencionado, y muchos otros más

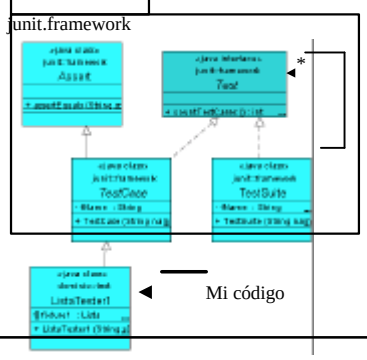
16

Clases fundamentales



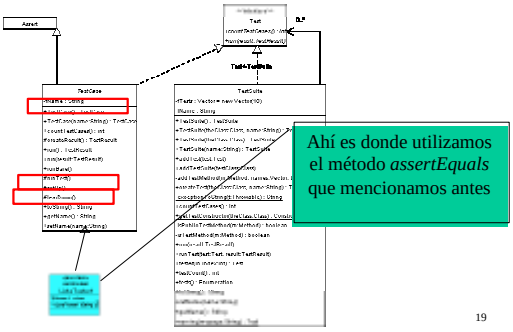
17

Clases fundamentales



18

Clases fundamentales



19

Clases fundamentales: Assert

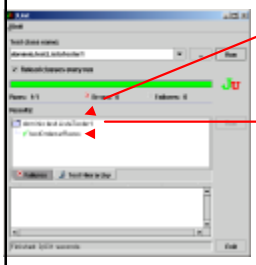
20

El framework JUnit

```
public class ListaTester1 extends TestCase
{
  public ListaTester1(String sTestName)
  {
    super(sTestName);
  }

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

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

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

}
```

21

El TestRunner



```
public class ListaTester1 extends TestCase
{
  public ListaTester1(String sTestName)
  {
    super(sTestName);
  }

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

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

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

22

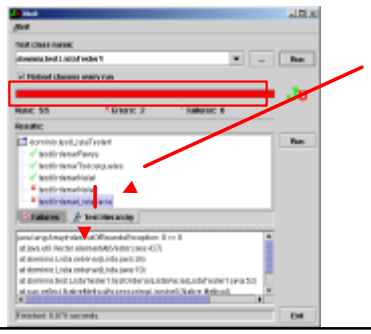
El TestRunner

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

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

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

 public void testOrdenarTodosIguales() {
   String[] e2={"a", "a", "a", "a", "a"};
   Lista listaTodosIguales=new Lista(e2);

   String[] ex={"a", "a", "a", "a", "a"};
   Lista expected=new Lista(ex);
   this.assertEquals(expected,
   listaTodosIguales.ordenar());
 }
```

```
public void testOrdenarNula1() {
   Lista listaNula1=null;
   this.assertNull(listaNula1);
 }

 public void testOrdenarNula2() {
   String[] e4=null;
   Lista listaNula2=new Lista(e4);
   String[] ex=null;
   Lista expected=new Lista(ex);
   this.assertEquals(expected,
   listaNula2.ordenar());
 }

 public void testOrdenarListaVacia() {
   String[] e5={};
   Lista listaVacia=new Lista(e5);
   String[] ex={};
   Lista expected=new Lista(ex);
   this.assertEquals(expected,
   listaVacia.ordenar());
 }
```

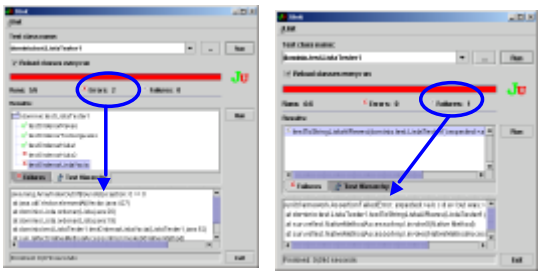
23

El TestRunner



24

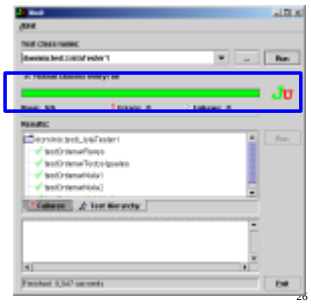
El TestRunner



25

El TestRunner

Una vez que la clase *Lista* ha sido corregida...



26

El TestRunner

- Es importante notar que todos los métodos *test* que vamos implementando se quedan guardados en *ListaTester*
- Si añadimos, borramos o modificamos el código de *Lista*, los casos de prueba habidos en *ListaTester* siguen disponibles y pueden volver a ser ejecutados
- Se aconseja reejecutarlos cada vez que se modifique el código

27

Términos

- En muchos casos, pueden ser utilizad pruebas
- Supongamos que método *toString()*:

```
public String toString()
 {
   String s="";
   for (int i=0; i<size(); i++)
     s+=" " + elementAt(i);
   return s;
 }
```

- También nos interesará probar el toString() con la lista nula, la lista vacía, etc.

28

Términos

```
public void testOrdenarReves() {
  String[] ex={"a", "b", "c", "d", "e"};
  Lista expected=new Lista(ex);
  String[] e3={"e", "d", "c", "b", "a"};
  Lista listaAlReves=new Lista(e3);
  this.assertEquals(expected, listaAlReves.ordenar());
}

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

29

Términos: *fixture*

- En casos como el anterior creamos *fixtures* (~ elementos fijos)
- Son variables de instancia de la clase de Test
- Se les asigna valor en el método *setUp()*, heredado de *TestCase*
- Se liberan en *tearDown()*
- *setUp* y *tearDown* se ejecutan antes y después de cada el *TestRunner* llame a cada método *test*

30

Términos: *fixture*

```
public void setUp() {
  String[] e1={"a", "a", "a", "a", "a"};
  listaTodosIguales=new Lista(e1);
  String[] e2={"a", "b", "c", "d", "e"};
  listaOrdenada=new Lista(e2);
  String[] e3={"e", "d", "c", "b", "a"};
  listaAlReves=new Lista(e3);
  listaNula1=null;
  String[] e4=null;
  listaNula2=new Lista(e4);
  String[] e5={};
  listaVacia=new Lista(e5);
}
```

31

Términos: *TestSuite*

- En otras ocasiones será bueno agrupar casos de prueba: por ejemplo, tener un grupo de pruebas en el que ponemos las pruebas realizadas a listas vacías y nulas

32

Términos: *TestSuite*

```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raíz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
}
```

33

Términos: *TestSuite*

raiz

```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raíz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
}
```

34

Términos: *TestSuite*

raiz

suite1

```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raíz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
}
```

35

Términos: *TestSuite*

raiz

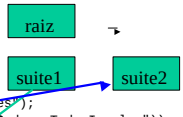
suite1

testOrdenarTodosIguales

```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raíz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
}
```

36

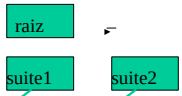
Términos: *TestSuite*




```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raiz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
 }
```

37

Términos: *TestSuite*

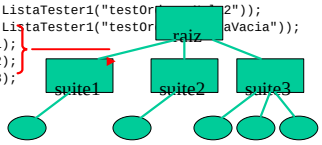



```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raiz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
 }
```

38

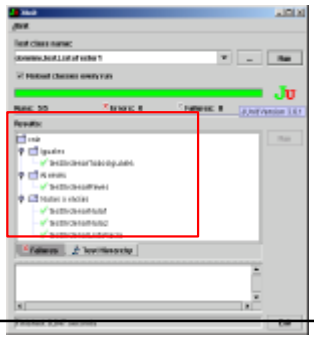
Términos: *TestSuite*

```
public static TestSuite suite() {
   TestSuite raiz=new TestSuite("raiz");
   TestSuite suite1=new TestSuite("Iguales");
   suite1.addTest(new ListaTester1("testOrdenarTodosIguales"));
   TestSuite suite2=new TestSuite("Al revés");
   suite2.addTest(new ListaTester1("testOrdenarReves"));
   TestSuite suite3=new TestSuite("Nulas o vacías");
   suite3.addTest(new ListaTester1("testOrdenarNula1"));
   suite3.addTest(new ListaTester1("testOrdenarNula2"));
   suite3.addTest(new ListaTester1("testOrdenarListaVacia"));
   raiz.addTest(suite1);
   raiz.addTest(suite2);
   raiz.addTest(suite3);
   return raiz;
 }
```



39

Términos: *TestSuite*



40

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

41

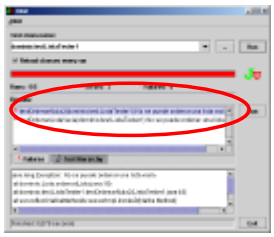
Pruebas de excepciones (*fail*)

- 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;
}
```

42

Pruebas de excepciones (*fail)*



```
public void testOrdenarNula2()
                throws Exception {
  String[] ex=null;
  Lista expected=new Lista(ex);
  this.assertEquals(expected,
            listaNula2.ordenar());
}

public void testOrdenarListaVacia()
                  throws Exception {
    String[] ex={};
    Lista expected=new Lista(ex);
    this.assertEquals(expected,
                listaVacia.ordenar());
}
```

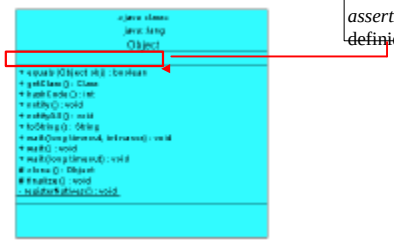

Pruebas de excepciones (*fail*)

- 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
    }
 }
```


Redefinición del método *equals*

- Todas las clases Java son especializaciones de *Object*

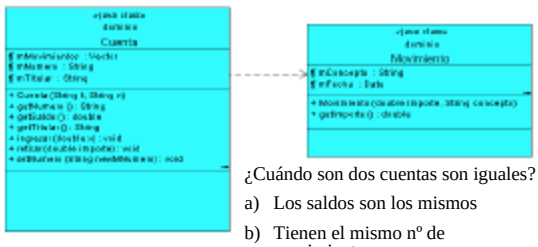


Llamado por los *assertEquals(...)* definidos en *Assert*

Redefinición del método *equals*

- Por tanto, en muchos casos tendremos que redefinir *equals(Object):boolean* en la clase que estamos probando

Ejemplo “equals” (I)

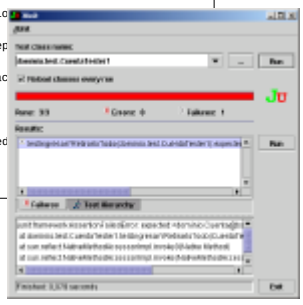


¿Cuándo son dos cuentas son iguales?

a) Los saldos son los mismos
b) Tienen el mismo nº de movimientos
c) Opción b y todos son iguales
d) ...

Ejemplo “equals” (II)

```
public void testIngresarYRetirarlo
{
  Cuenta expected=new Cuenta("Pep

  Cuenta obtained=new Cuenta("Mac
  obtained.ingresar(1000.0);
  obtained.retirar(1000.0);

  assertEquals(expected, obtained
}
```

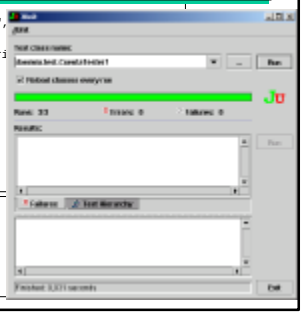


Ejemplo "equals"

Si redefinimos *equals(Object): boolean* en Cuenta de ese modo...

```
public void testIngresarYReti
 {
   Cuenta expected=new Cuenta("Pepe",

   Cuenta obtained=new Cuenta("Macari
   obtained.ingresar(1000.0);
   obtained.retirar(1000.0);

   assertEquals(expected, obtained);
 }
```

```
public boolean equals(Object o){
  if (!Cuenta.class.isInstance(o))
      return false;
  Cuenta c=(Cuenta) o;
  return getSaldo()==c.getSaldo());
}
```



Otros métodos *assertX*

- assertTrue(boolean)

```
public void testIngresar()
 {
   Cuenta obtained=new Cuenta("Pepe", "123");
   obtained.ingresar(100.0); obtained.ingresar(200.0);
   obtained.ingresar(300.0);
   assertTrue(obtained.getSaldo()==600.0);
 }
```

- assertNull(Object)

```
public void testNull()
 {
   Cuenta c=null;
   assertNull(c);
 }
```

50

Otros métodos *assertX*

- assertSame(Object, Object)/assertNotSame(Object, Object)

```
public void testDiferentesReferencias() throws
Exception
{
  Cuenta cuenta1=new Cuenta("Macario", "123456");
  cuenta1.ingresar(1000.0);
  cuenta1.retirar(1000.0);

  Cuenta cuenta2=new Cuenta("Macario", "123456");
  cuenta2.ingresar(1000.0);
  cuenta2.retirar(1000.0);

  assertEquals(cuenta1, cuenta2);
  assertNotSame(cuenta1, cuenta2);
}
```

51

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

52

Clases de prueba abstractas

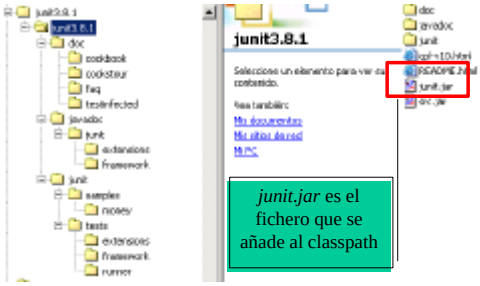
```
public abstract class TarjetaTester1 extends TestCase
{
  public TarjetaTester1(String sTestName)
  {
    super(sTestName);
  }

  public abstract Tarjeta getTarjeta();
  public abstract Tarjeta prepararTarjetaEsperada();

  public void testRetirar()
  {
    Tarjeta obtained=getTarjeta();
    obtained.retirar(100.0);

    Tarjeta expected=prepararTarjetaEsperada();
    assertEquals(expected, obtained);
  }
}
```

53

Instalación de JUnit

- http://www.junit.org

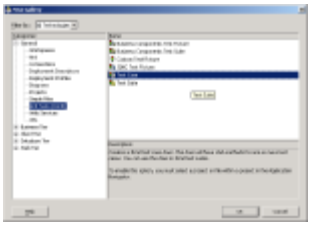
Instalación de JUnit



55

Instalación de JUnit

- Algunos IDEs ya ofrecen integración directa con JUnit



56

Objetos Mock (~ falsos)

- Basados en JUnit
- Sustituyen a clases complejas, dispositivos, etc.
- Ejemplos: servlets, páginas jsp, bases de datos...

57

Objetos Mock: ejemplo

```
public class temperature extends HttpServlet
{
  private static final String CONTENT_TYPE = "text/html";

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }

  public void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException
  {
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    String str_f=request.getParameter("Fahrenheit");

    try {
      int temp_f=Integer.parseInt(str_f);
      double temp_c=(temp_f-32)*5/9.0;
      out.println("Fahrenheit: " + temp_f + ", Celsius: " + temp_c);
    }
    catch (NumberFormatException e) {
      out.println("Invalid temperature: " + str_f);
    }
  }
}
```

58

Objetos Mock: ejemplo

```
import com.mockobjects.servlet.*;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class TemperatureTester extends TestCase
{
  public TemperatureTester()
  {
  }

  public void test_bad_parameter() throws Exception {
    temperature s = new temperature();
    MockHttpServletRequest request=new MockHttpServletRequest();
    MockHttpServletResponse response=new MockHttpServletResponse();
    request.setupAddParameter("Fahrenheit", "boo!");
    response.setExpectedContentType("text/html");
    s.doGet(request, response);
    response.verify();
    assertTrue(response.getOutputStreamContents().startsWith("Invalid
   temperature"));
  }
```

59

Tomado y adaptado de: Thomas y Hunt (2002). Mock Objects. IEEE Software, nº de mayo/junio, pp. 22-24.

Objetos Mock: ejemplo

```
import com.mockobjects.servlet.*;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class TemperatureTester extends TestCase
{
  public TemperatureTester()
  {
  }

  public void test_bad_parameter() throws Exception {
    temperature s = new temperature();
    MockHttpServletRequest request=new MockHttpServletRequest();
    MockHttpServletResponse response=new MockHttpServletResponse();
    request.setupAddParameter("Fahrenheit", "boo!");
    response.setExpectedContentType("text/html");
    s.doGet(request, response);
    response.verify();
    assertTrue(response.getOutputStreamContents().startsWith("Invalid
   temperature"));
  }
  ...
}
```

60

Objetos Mock

- En el caso anterior, el *MockHttpServletRequest* y el *MockHttpServletResponse* son objetos *HttpServletRequest* y *HttpServletResponse*, ya que el servlet que estamos probando trabaja con objetos de estos tipos

61

Objetos Mock

62

Objetos Mock

```
...
request.setupAddParameter("Fahrenheit", "boo!");
response.setExpectedContentType("text/html");
s.doGet(request, response);
response.verify();
...
```

Operaciones específicas para probar

Objetos Mock

- De forma general, todos los objetos Mock comparten la misma estructura:
  - Especializan a la clase que se usa realmente (implementan por tanto todas sus posibles operaciones abstractas)
  - Contienen un conjunto de operaciones adicionales *addExpected...* o *setupExpected...*, que van indicando al objeto el estado en que quedará tras ejecutar la operación de "dominio"
  - Pueden implementar la interfaz *Verifiable* (método *verify()*)

64

Objetos Mock

- Difíciles de usar (poca documentación)
- Descargas y más información en www.mockobjects.com

65

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
- Extensible (p.ej.: Mock), abierto, gratuito

66

Pruebas de programas Java mediante JUnit

Macario Polo Usaola
Grupo Alarcos
Escuela Superior De Informática
Universidad De Castilla-la Mancha
http://www.inf-cr.uclm.es/www/mpolo

67