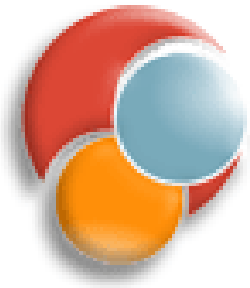


Java y Herramientas de Desarrollo

Sesión 7: JUnit

Puntos a tratar

- Introducción a JUnit
- Modos de utilizar JUnit
- Un ejemplo sencillo
- Ejecución de pruebas
- Múltiples pruebas de una clase

JUnit, casos y suites de prueba

- JUnit es una librería que permite definir casos de prueba para probar los diferentes módulos de un programa Java

- *Caso de prueba*: clase o módulo con métodos para probar los métodos de una clase o módulo concreto
- *Suite de prueba*: organización de casos de prueba, en forma de una jerarquía determinada (normalmente árbol)

Características de JUnit

- Para aplicaciones grandes se tendrá un árbol de casos de prueba, del cual podremos ejecutar cualquier rama

- Se permite la *regresión*: al cambiar un módulo ya probado, podremos afectar a otros módulos, y con la estructura anterior podremos reutilizar las pruebas necesarias

Página de JUnit

- Más información de JUnit en:

http://www.junit.org

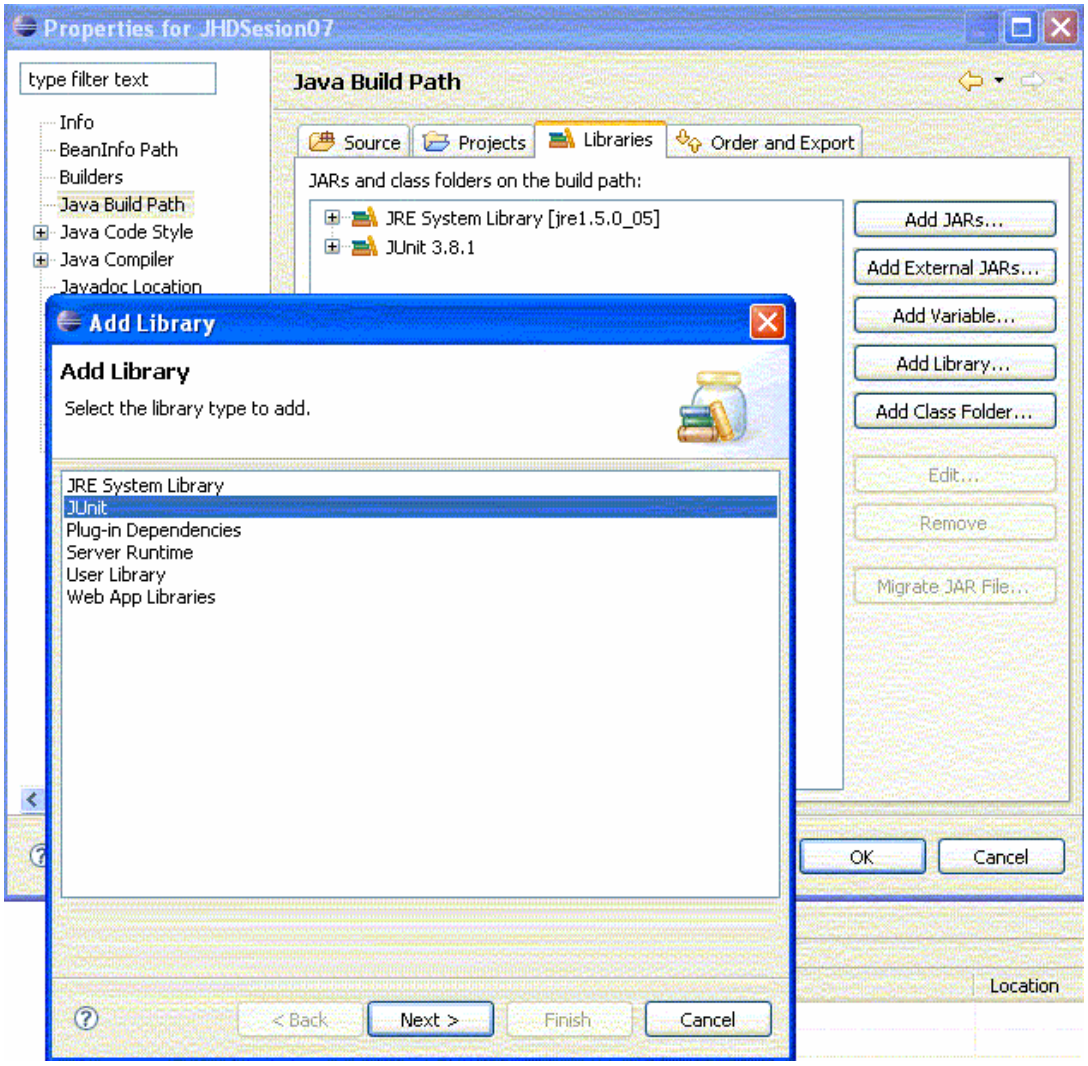
- Utilizamos el fichero *junit.jar* de la distribución, para emplear las clases necesarias de JUnit.

Integración de JUnit en Eclipse

- Eclipse incorpora JUnit como herramienta interna para facilitar la integración de los casos de prueba en nuestros proyectos

- Normalmente sólo hace falta añadir la librería de JUnit interna de Eclipse al proyecto, aunque en algunas versiones anteriores de Eclipse sí es necesario algún paso previo de configuración

Añadir la librería de JUnit al proyecto



Utilizar JUnit como librería independiente

- Además de tenerlo integrado en Eclipse, podemos utilizar JUnit fuera de él, añadiendo el fichero *junit.jar* de la distribución oficial al CLASSPATH del proyecto que estemos creando.
- En las siguientes secciones explicaremos cómo crear y ejecutar pruebas de JUnit tanto dentro como fuera de Eclipse

Un ejemplo sencillo

- Definimos la clase *Matriz*, con una operación de suma de matrices:

```
public class Matriz
{
   int[][] elem;

   public Matriz(int[][] elem) {
        // Código del constructor
   }
   public Matriz suma(Matriz m) {
        // Código del método
   }
   public boolean equals(Matriz m) {
        // Devuelve true si son iguales, false si no
   }
}
```

Clase de prueba asociada

- La clase de prueba hereda de *junit.framework.TestCase*, y define métodos *testXXX(...)* para probar los métodos necesarios:

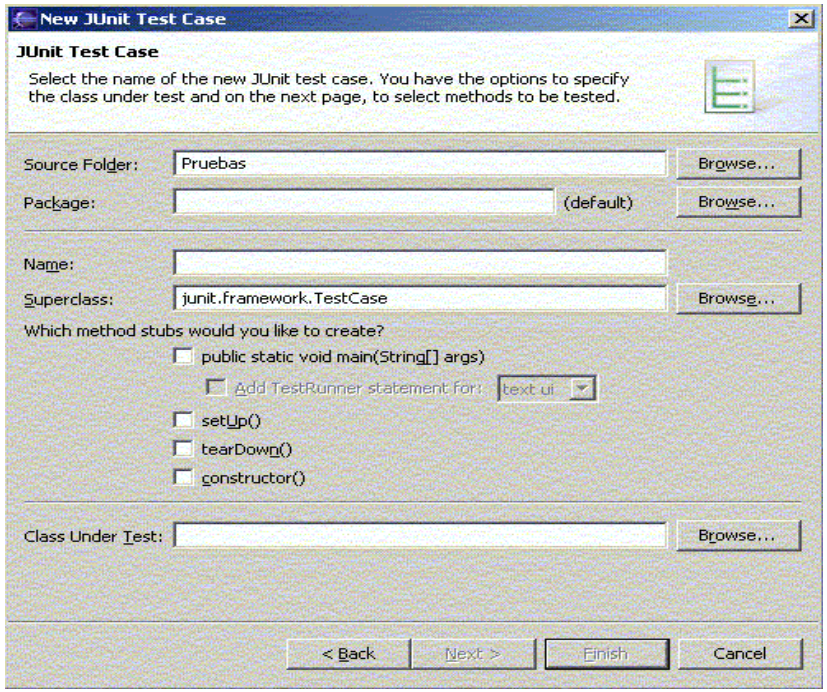
```
public class MatrizTest extends junit.framework.TestCase {
   ...
   public static int[][] MATRIZ1 = {{1,2,4},{3,5,7}, {2,2,6}};
   public static int[][] MATRIZ2 = {{3,3,4},{5,6,7}, {1,2,4}};
   public static int[][] SUMA = {{4,5,8},{8,11,14}, {3,4,10}};

   public void testSuma() {
        Matriz m1 = new Matriz(MATRIZ1);      // Matriz ejemplo
        Matriz m2 = new Matriz(MATRIZ2);      // Matriz ejemplo
        Matriz mSumaOK = new Matriz(SUMA);    // Resultado OK
        Matriz mSumaTest = m1.suma(m2);       // Resultado test
        assertTrue(mSumaOK.equals(mSumaTest);
} }
```

Aspectos sobre la clase de prueba

- Se suelen definir constantes (como *MATRIZ1, MATRIZ2 o SUMA* en el ejemplo) con algunos valores de prueba conocidos
- Después compara lo que devuelven los métodos con los valores que ya se conocen
- Se tienen métodos *assertXXX(...)* en *TestCase* para verificar ciertas comparaciones. Podemos poner tantos de ellos como haga falta en cada método.
  - El método fallará con que falle alguno de los *assertXXX(...)*
- Se aconseja colocar las clases de prueba en el mismo paquete que las originales, y con el mismo nombre, terminado en *...Test*.

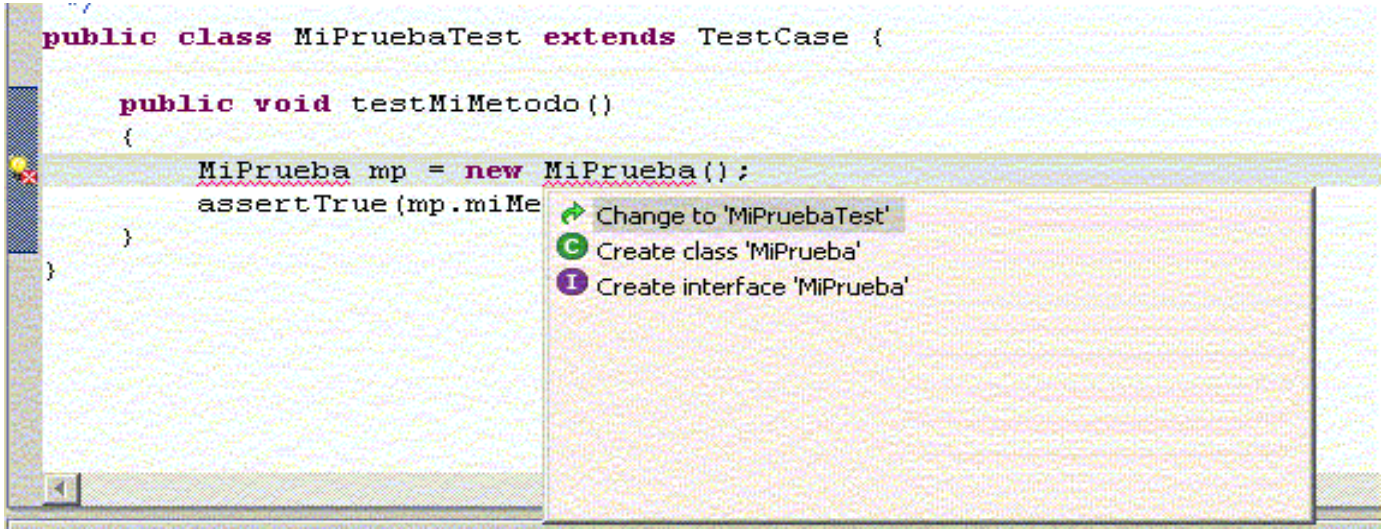
Crear clases de prueba desde Eclipse

- Desde *File – New – JUnit – TestCase* podemos crear directamente clases de prueba, eligiendo el nombre de la clase de prueba, el nombre de la clase a probar, e incluso los métodos a probar



Crear clases de prueba sin las originales

- También podemos *definir la clase de prueba antes que la clase a probar*. Cuando lo hagamos, en las líneas donde dé error por no tener definida la clase a probar, pinchamos con el botón derecho y elegimos *Create class 'nombreclase'*
- A esta metodología se le llama **Desarrollo Dirigido por las Pruebas** (*Test-Driven Development*)



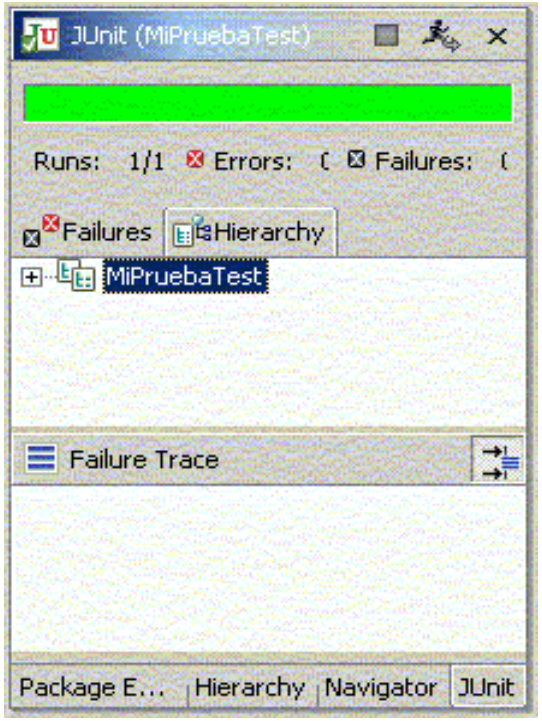
Pruebas con excepciones

- Tenemos el método *fail(...)* para provocar el error de la prueba cuando ésta debe lanzar una excepción, y no la lanza:

```
public void testMetodo()
{
   MiClase m = new MiClase();
   try
   {
        m.unMetodo();
        fail("Debería haber lanzado excepcion");
   } catch (Exception ex) {
        // Funcionamiento correcto, no hay que hacer nada
   }
}
```

Ejecución de pruebas desde Eclipse

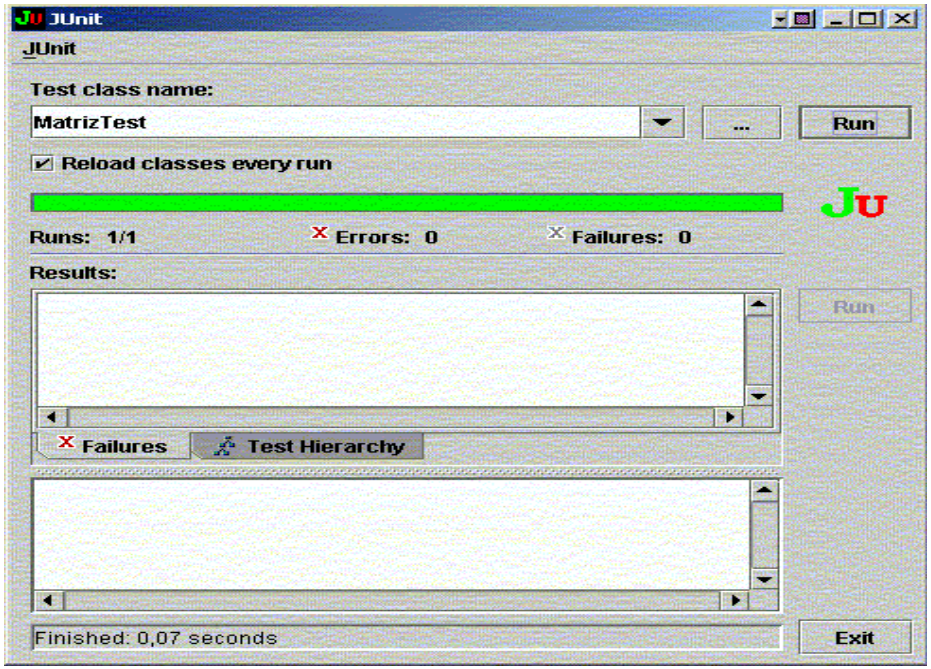
- *Probar las clases*: desde la clase de prueba que queramos ejecutar, vamos a *Run – Run As – JUnit Test*



Ejecución de pruebas fuera de Eclipse

- Podemos ejecutar las clases *junit.textui.TestRunner* (para modo texto) o *junit.swingui.TestRunner* (para modo gráfico), del fichero *junit.jar*:

```
java –cp ./junit.jar junit.swingui.TestRunner
```



Ejecución mediante *main(...)*

- También podemos definir un *main(...)* en nuestro *TestCase* que ejecute dichas clases, y entonces ejecutar nuestra *TestCase* directamente, como una aplicación Java normal:

```
public MatrizTest extends TestCase
{
   ...
   public static void main(String[] args)
   {
        String[] nombresTest = {MatrizTest.class.getName()};
        junit.swingui.TestRunner.main(nombresTest);
   }
}
```

Múltiples pruebas: añadir nuevos métodos

- Imaginemos que añadimos un método *restar(...)* a nuestra clase *Matriz*, para restar matrices:

```
public class Matriz {
   ...

   public Matriz resta(Matriz m)
   {
        // Codigo de la resta de matrices
   }
}
```

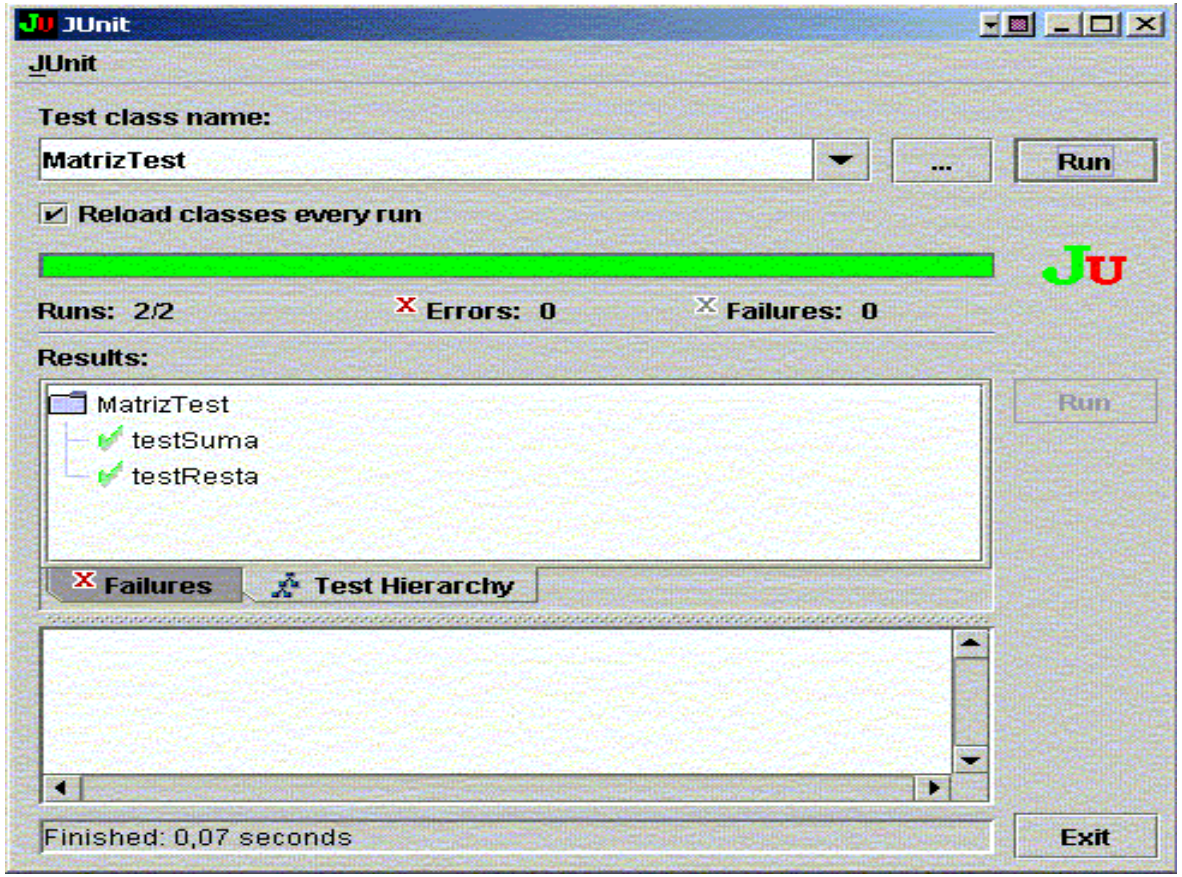
Añadir nuevos métodos de prueba

- Definiríamos nuestro método *testResta( )*en nuestra clase *MatrizTest*, para verificar la resta

```
public MatrizTest extends TestCase
{
   ...
   public static int[][] RESTA={{-2,-1,0},{-2,-1,0}, {1,0,2}};

   public void testResta() {
        Matriz m1 = new Matriz(MATRIZ1);      // Matriz ejemplo
        Matriz m2 = new Matriz(MATRIZ2);      // Matriz ejemplo
        Matriz mRestaOK = new Matriz(RESTA); // Resultado OK
        Matriz mRestaTest = m1.resta(m2);     // Resultado test
        assertTrue(mRestaOK.equals(mRestaTest);
   }
}
```

Ejecución de múltiples pruebas

- Y volveríamos a ejecutar las pruebas como antes:



Inicialización: *setUp* y *tearDown*

- Si tenemos código similar en los métodos *testXXX(...)* de nuestro *TestCase*, podemos utilizar el método *setUp( )* para colocarlo allí una sola vez:

```
public MatrizTest extends TestCase {
   public void setUp() {
      // Código de inicialización general, como por ejemplo:
      m1 = new Matriz(MATRIZ1);      // Matriz de prueba 1
      m2 = new Matriz(MATRIZ2);      // Matriz de prueba 2
   }
   ...
}
```

- Se tiene también un método *tearDown( )* si queremos liberar recursos al final de la prueba (por ejemplo, borrar registros de prueba de una base de datos)

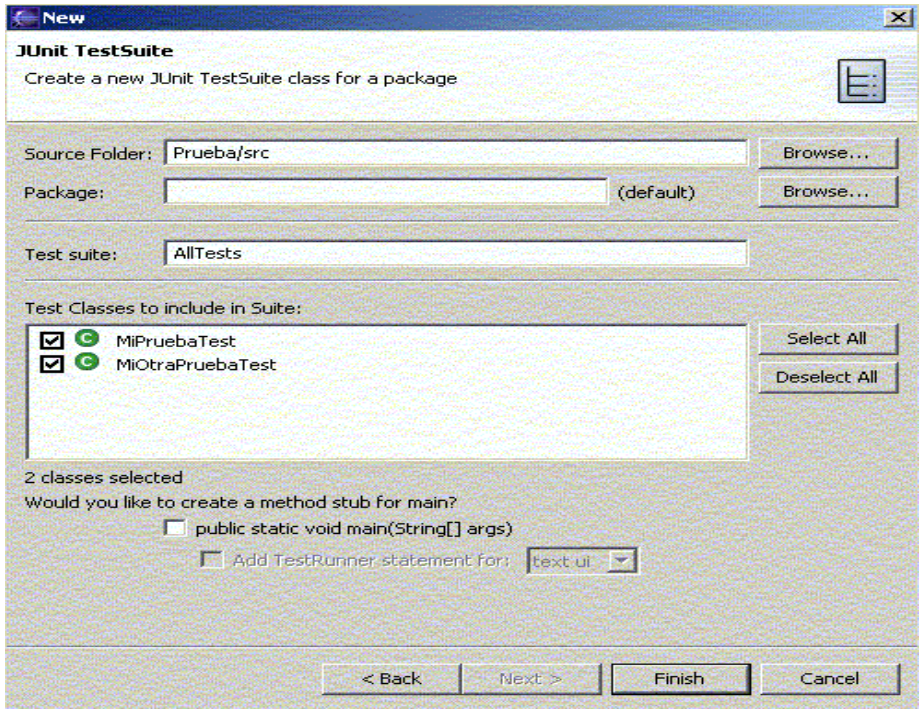
Suites de pruebas

- Con las suites podemos agrupar métodos de prueba
- Hacemos una clase con un método estático *suite( )* que devuelva un objeto *Test*:

```
public class MiSuite {
   public static Test suite () {
        TestSuite suiteRaiz = new TestSuite("Raiz");
        TestSuite suiteMatriz = new TestSuite("SuiteMatriz");
        TestSuite suiteOtra = new TestSuite("SuiteOtra");
        suiteMatriz.addTest(new MatrizTest("testSuma"));
        suiteRaiz.addTest(suiteMatriz);
        suiteRaiz.addTest(suiteOtra);
        ...
        return suiteRaiz;
   }
}
```

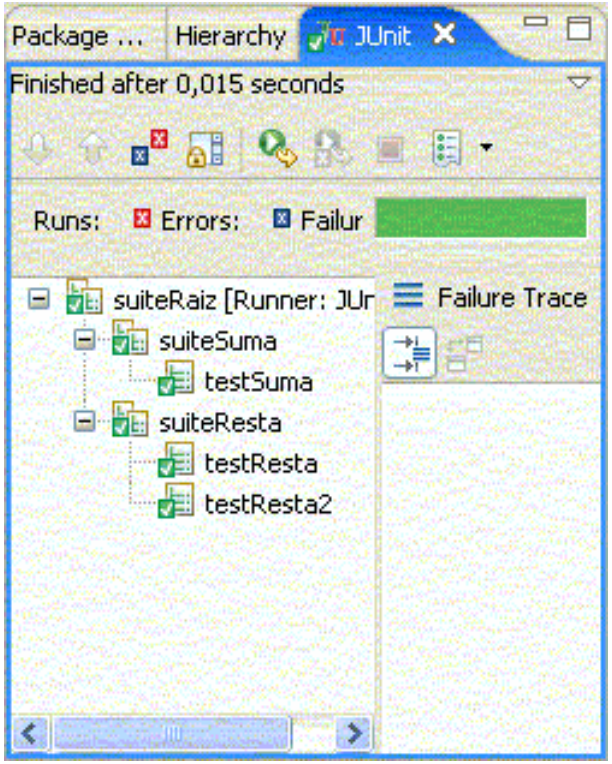
Suites con Eclipse

- Para crear suites de pruebas, vamos a *File – New – JUnit – TestSuite*, e indicamos el nombre de la suite y las clases de prueba a incluir:



Ejecutar suites

- Se ejecutan como una prueba normal, desde *Run – Run As – JUnit Test*



Alternativas en las suites

- Podemos colocar en una suite pruebas de diferentes clases (diferentes *TestCases*)
- Con *addTest(...)* también podemos añadir todas las pruebas de una clase entera con el objeto *class* de la clase:

```
suiteMatriz.addTest(new TestSuite(MatrizTest.class));
```

- Para ejecutarlas sin Eclipse, definimos en la suite un *main(...)* que lance el *run( )* de *TestRunner*, y éste llama automáticamente al método *suite( )*:

```
public static void main(String[] args) {
   junit.swingui.TestRunner.run(MiSuite.class);
}
```

JUnit y Ant

- Podemos ejecutar pruebas de JUnit desde tareas de Ant
  - Añadimos la librería de *junit.jar* al classpath de Ant (*Window – Preferences – Ant – Runtime*)
  - Creamos un fichero *build.xml* como:

```
<project name="XXXXXX" default="ejecutaJUnit" basedir=".">

  <property name="src" value="./src" />
  <property name="classes" value="./bin" />
  <property name="test.class" value="es.ua.jtech.jhd.sesion7.ejemplos.MatrizSuite" />

  <path id="test.classpath">
    <pathelement location="${classes}" />
    <pathelement location="C:/eclipse/plugins/org.junit_3.8.1/junit.jar" />
  </path>

  <target name="ejecutaJUnit">
    <junit fork="yes" haltonfailure="yes">
      <test name="${test.class}" />
      <formatter type="plain" usefile="false" />
      <classpath refid="test.classpath" />
    </junit>
  </target>

</project>
```