



Tema 3

Pruebas unitarias con JUnit

Proyecto de Análisis y Diseño de Software
2º Ingeniería Informática
Universidad Autónoma de Madrid

Pruebas unitarias

- Una prueba unitaria permite comprobar el funcionamiento de un módulo dentro un sistema (en nuestro caso, comprueba una clase y sus métodos).
- Las pruebas unitarias deben ser:
 - Automatizables: para ello usaremos JUnit 4.
 - Completas
 - Independientes: la ejecución de una prueba no debe afectar a otras

JUnit

- Framework para la automatización de pruebas unitarias de programas Java.
- Por cada clase del sistema se crea una clase de prueba con métodos (*tests*) destinados a probar la clase del sistema.
- Cada test está anotado con `@Test`, y contiene:
 - Código que crea objetos necesarios para la prueba
 - Aserciones que comprueban la corrección del resultado
- JUnit ejecuta todos los tests, y devuelve un informe con el resultado de la ejecución.

JUnit – Ejemplo 1

Clase Java:

```
public class Complejo {  
    private float real;  
    private float imaginario;  
  
    public Complejo (float real, float imaginario) {  
        this.real = real;  
        this.imaginario = imaginario;  
    }  
  
    public float getReal(){  
        return real;  
    }  
  
    public float getImaginario() {  
        return imaginario;  
    }  
  
    public Complejo sumar(Complejo c) {  
        return new Complejo(  
            real + c.getReal(),  
            imaginario + c.getImaginario());  
    }  
}
```

Clase de prueba:

```
import static org.junit.Assert.*;  
import org.junit.Test;  
  
public class ComplejoTest {  
  
    @Test  
    public void testSumar() {  
        // Crear los objetos necesarios para la prueba  
        Complejo c1 = new Complejo(3, 5);  
        Complejo c2 = new Complejo(1, -1);  
  
        // Ejecutar la operacion a probar  
        Complejo resultado = c1.sumar(c2);  
  
        // Comprobar que el resultado es correcto  
        // utilizando aserciones  
        assertEquals(4f, resultado.getReal(), 0);  
        assertEquals(4f, resultado.getImaginario(), 0);  
    }  
}
```

JUnit

■ Aserciones más habituales:

- ☐ assertTrue(<expr_booleana>)
- ☐ assertFalse(<expr_booleana>)
- ☐ assertEquals(...)
- ☐ assertSame(<objeto1>, <objeto2>)
- ☐ assertNotSame(<objeto1>, <objeto2>)
- ☐ assertNull(<objeto>)
- ☐ assertNotNull(<objeto>)
- ☐ fail(<mensaje>): falla un test

Listado completo en:

<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

JUnit

- Los métodos de una clase de prueba se ejecutan en el siguiente orden:
 - Método anotado con `@BeforeClass`
 - Por cada método anotado con `@Test`:
 - Constructor de la clase de prueba
 - Método anotado con `@Before`
 - Método anotado con `@Test`
 - Método anotado con `@After`
 - Método anotado con `@AfterClass`

JUnit – Ejemplo 2

Clase Java:

```
public class Complejo {
    private float real;
    private float imaginario;

    public Complejo (float real, float imaginario) {
        this.real = real;
        this.imaginario = imaginario;
    }

    public float getReal() {
        return real;
    }

    public float getImaginario() {
        return imaginario;
    }

    public Complejo sumar(Complejo c) {
        return new Complejo(
            real + c.getReal(),
            imaginario + c.getImaginario());
    }

    public Complejo dividir(Complejo c) {
        ...
    }
}
```

Clase de prueba:

```
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.Before;
```

```
public class ComplejoTest {

    private Complejo c1;
    private Complejo c2;
```

@Before

```
public void setUp() throws Exception {
    c1 = new Complejo(3, 5);
    c2 = new Complejo(1, -1);
}
```

*se ejecuta antes
de cada test*

@Test

```
public void testSumar() {
    Complejo resultado = c1.sumar(c2);
    assertEquals(4f, resultado.getReal(), 0);
    assertEquals(4f, resultado.getImaginario(), 0);
}
```

@Test

```
public void testDividir() {
    Complejo resultado = c1.dividir(c2);
    assertEquals(-1f, resultado.getReal(), 0);
    assertEquals(4f, resultado.getImaginario(), 0);
}
```

JUnit – Ejemplo 3

Clase Java:

```
public class Complejo {
    private float real;
    private float imaginario;

    public Complejo (float real, float imaginario) {
        this.real = real;
        this.imaginario = imaginario;
    }

    public float getReal() {
        return real;
    }

    public float getImaginario() {
        return imaginario;
    }

    public Complejo sumar(Complejo c) {
        return new Complejo(
            real + c.getReal(),
            imaginario + c.getImaginario());
    }

    public Complejo dividir(Complejo c)
        throws Exception {
        ...
    }
}
```

Clase de prueba:

```
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.Before;

public class ComplejoTest {

    private Complejo c1;
    private Complejo c2;

    @Before
    public void setUp() throws Exception {
        c1 = new Complejo(3, 5);
        c2 = new Complejo(1, -1);
    }

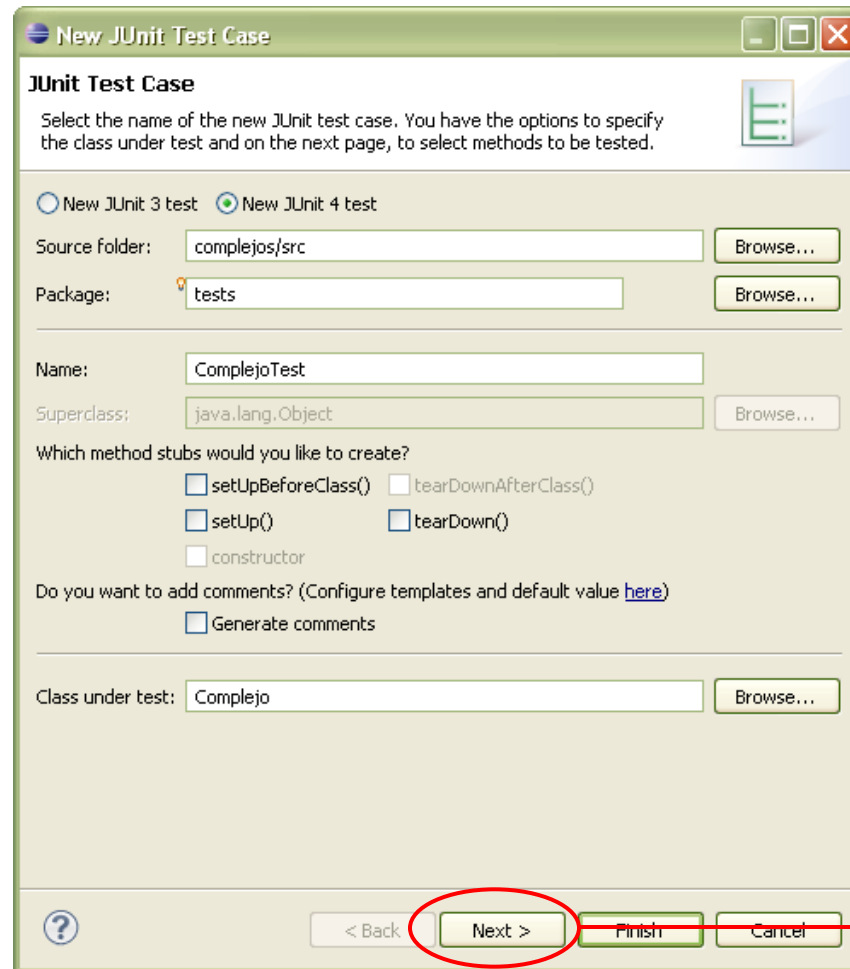
    ...

    @Test(expected = ArithmeticException.class)
    public void testDividirPorCero(){
        Complejo cero = new Complejo(0, 0);
        Complejo resultado = c1.dividir(cero);
    }
}
```

El test termina correctamente si lanza una excepción de tipo ArithmeticException

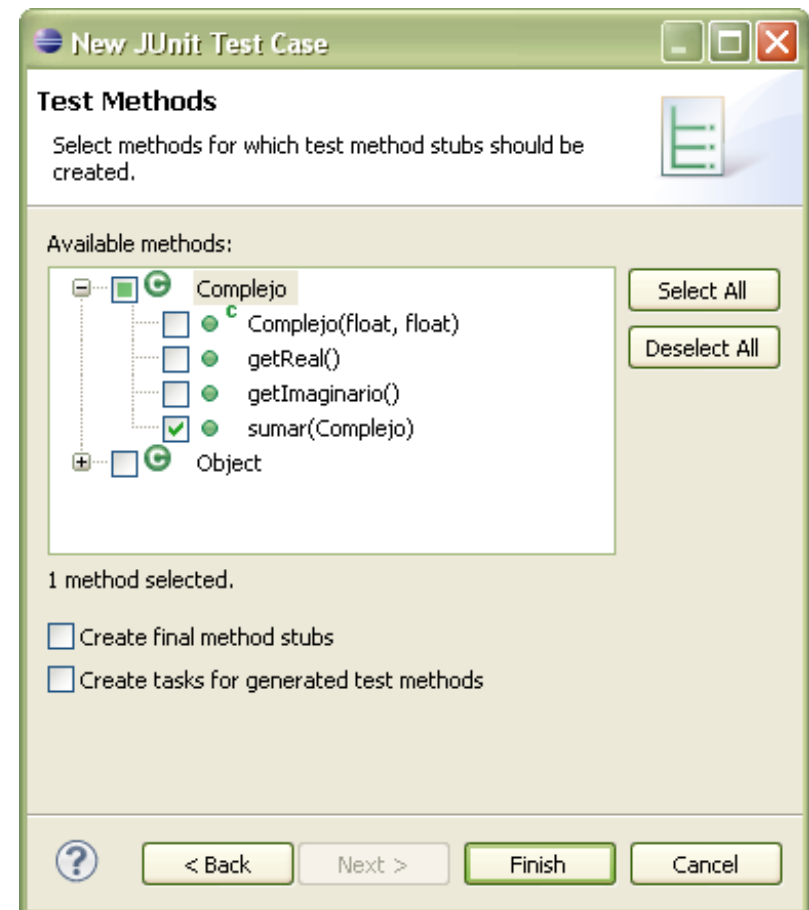
JUnit en Eclipse (i)

Paso 1:
crear clase de prueba
(*new Junit Test Case*)



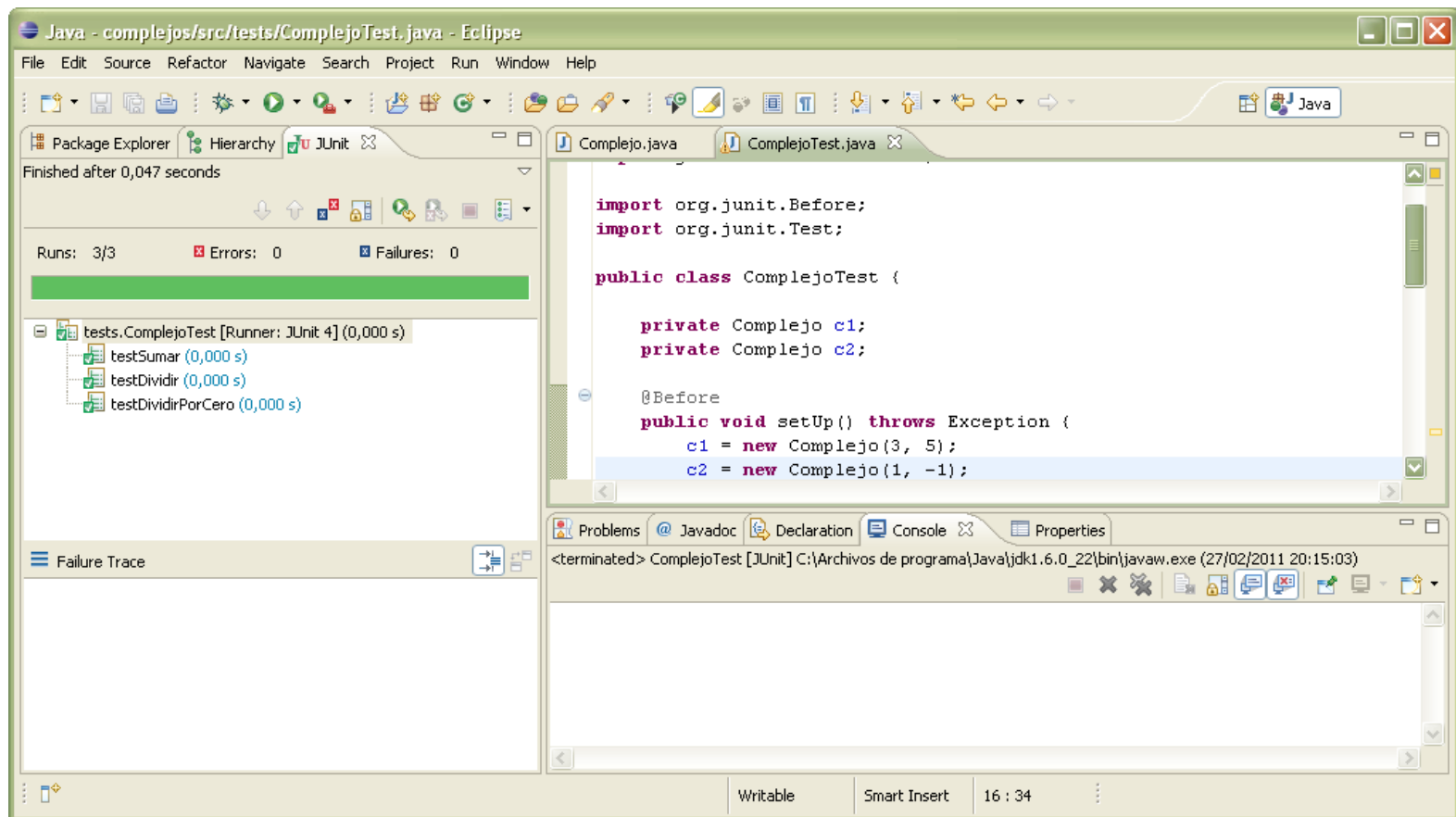
JUnit en Eclipse (ii)

Paso 2:
seleccionar métodos a probar



JUnit en Eclipse (iii)

Resultado de pasar los casos de prueba:



JUnit – Pruebas en bloque

Clase para ejecutar en secuencia todas las clases de pruebas

```
package es.uam.eps.padsof.proyecto.test;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
```

```
@RunWith(Suite.class)
@SuiteClasses({ AlumnoTest.class,
                AsignaturaTest.class,
                ProfesorTest.class,
                MensajeTest.class,
                VisitanteTest.class,
                UsuarioTest.class
                })
```

```
public class AllTests {

}
```

La clase está vacía pero al ejecutarla como un Junit Test Case se ejecutan una a una todas las pruebas de clases indicadas arriba

¿Cuándo?

- ***Pruebas de Unidad:*** para probar el código realizado, antes de la integración con otras clases. Estas pruebas se realizan a la vez que se programa.
- ***Pruebas de Regresión:*** para comprobar que al modificar el código no se han introducido errores.
- ***Desarrollo Dirigido por Pruebas:*** antes de crear el propio código. Las pruebas de unidad ayudan a diseñar el API de la clase que se está creando.

Bibliografía

- <http://junit.sourceforge.net/>
- **Test Driven: TDD and Acceptance TDD for Java Developers.** Koskela, Manning Publications, 2007.
- **Pruebas de software y JUnit: un análisis en profundidad y ejemplos prácticos.** Bolaños, Sierra, Alarcón. Prentice-Hall, 2008.
INF/681.3.06/BOL