# Guía rápida: Test unitarios en Java (Junit con Eclipse)

## Introducción

La prueba de programas es una aproximación dinámica a la verificación de los mismos, en la cual se ejecuta el software con unos datos o casos de prueba para analizar el buen funcionamiento de los requisitos esperados del programa. Podemos decir que la prueba es un tipo de verificación que se basa en la ejecución del código.

La prueba de unidad es la prueba de un módulo concreto dentro de un software que incluirá muchos otros módulos. Un módulo es una unidad de código que sirve como bloque de construcción para la estructura física de un sistema. El concepto de "módulo" se debe interpretar en términos del lenguaje de programación que estemos utilizando. Por ejemplo, podemos considerar que una clase Java o C++ es un módulo, pero también lo es un módulo del lenguaje Modula-2 o una unidad en TurboPascal.

Una prueba de unidad es un programa que prueba a otro programa. En términos de Java, es una clase con el objetivo de probar a otra clase. Es decir, es un trozo de código que escribimos para determinar si otro trozo de código se comporta como esperábamos o no.

¿Cómo se hace esto?

Para esto usamos aserciones. Una aserción, *assertion*, es un simple método que verifica si algo es verdadero. Por ejemplo, el método *assertTrue*, comprueba si una condición booleana es verdadera y falla en caso contrario. Su implementación podría ser:

```
public void assertTrue(boolean condition) {
  if (!condition) {
    abort();
  }
}
```

En los últimos años, ha aparecido una metodología de desarrollo de programas denominada UExtreme Programming (XP) U(Beck 1999), que hace un énfasis especial en las prácticas y técnicas de prueba unitaria. Como resultado de ese énfasis, se han creado una serie de frameworks para ayudar a realizar pruebas unitarias en diferentes lenguajes. Al conjunto de esos frameworks se les denomina xUnit.

Nos centramos en JUNIT, que es "el XUNIT para Java", los conceptos son muy similares a los que nos encontramos en los frameworks XUNIT para otros lenguajes (incluyendo a C++, Delphi, Smalltalk y Visual Basic).

**Assert** junit.framework

Object
└Assert

| | |
|---|---|
| protected . . . . . | **Assert** () |
| public static . . . . . void | **assertEquals** (String message, Object expected, Object actual) |
| public static . . . . . void | **assertEquals** (Object expected, Object actual) |
| public static . . . . . void | **assertEquals** (String message, *[double\|float]* expected, *[double\|float]* actual, *[double\|float]* delta) |
| public static . . . . . void | **assertEquals** (*[double\|float]* expected, *[double\|float]* actual, *[double\|float]* delta) |
| public static . . . . . void | **assertEquals** (String message, *[boolean\|byte\|char\|int\|long\|short]* expected, *[boolean\|byte\|char\|int\|long\|short]* actual) |
| public static . . . . . void | **assertEquals** (*[boolean\|byte\|char\|int\|long\|short]* expected, *[boolean\|byte\|char\|int\|long\|short]* actual) |
| public static . . . . . void | **assertNotNull** (String message, Object object) |
| public static . . . . . void | **assertNotNull** (Object object) |
| public static . . . . . void | **assertNull** (String message, Object object) |
| public static . . . . . void | **assertNull** (Object object) |
| public static . . . . . void | **assertSame** (String message, Object expected, Object actual) |
| public static . . . . . void | **assertSame** (Object expected, Object actual) |
| public static . . . . . void | **assertTrue** (String message, boolean condition) |
| public static . . . . . void | **assertTrue** (boolean condition) |
| public static . . . . . void | **fail** (String message) |
| public static . . . . . void | **fail** () |

| Método | Qué hace |
|---|---|
| `assertTrue(boolean condicion)` | Falla si la condición es false; pasa en otro caso. |
| `assertEquals(Object esperado, Object actual)` | Falla si esperado y actual no son iguales, según el método equals(); pasa en otro caso. |
| `assertEquals(int esperado, int actual)` | Falla si esperado y actual no son iguales, según el operador ==; pasa en otro caso. Este método está sobrecargado para cada tipo primitivo: int, float, double, char, byte, long, short, y boolean. |
| `assertSame(Object esperado, Object actual)` | Falla si esperado y actual se refieren a diferentes objetos en memoria; pasa si se refieren al mismo objeto en memoria. |
| `assertNull(Object object)` | Pasa si es null; En otro caso falla. |

Además están los métodos assertFalse(), assertNotSame() y assertNotNull().

Para aritmética con punto flotante :

assertEquals([String message], expected, actual**, tolerance**)
Ejemplo: assertEquals (expectedDouble, actualDouble, 0.0001d).

También resulta de mucha utilidad:

**fail**

fail([String message])

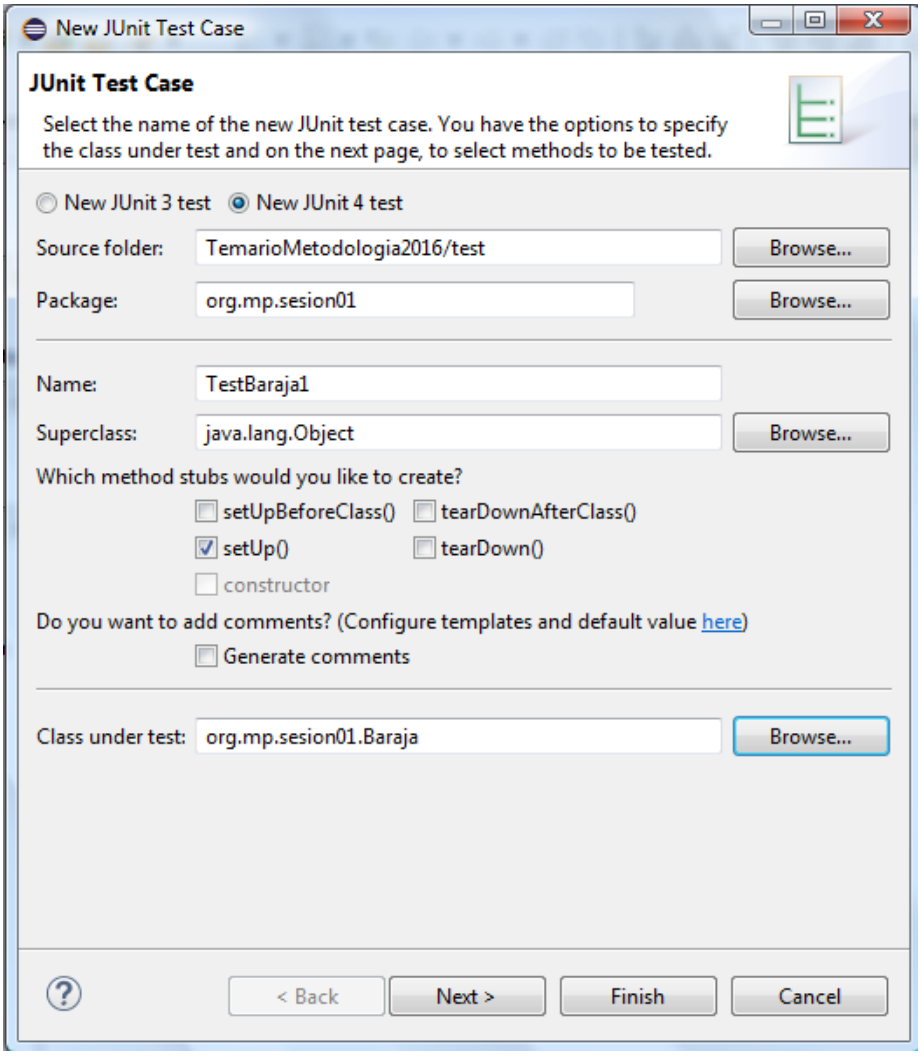
Hace que el test falle inmediatamente con el mensaje que se indique que es opcional. Normalmente se usa para probar código que tiene excepciones y estás no han sido tratadas.

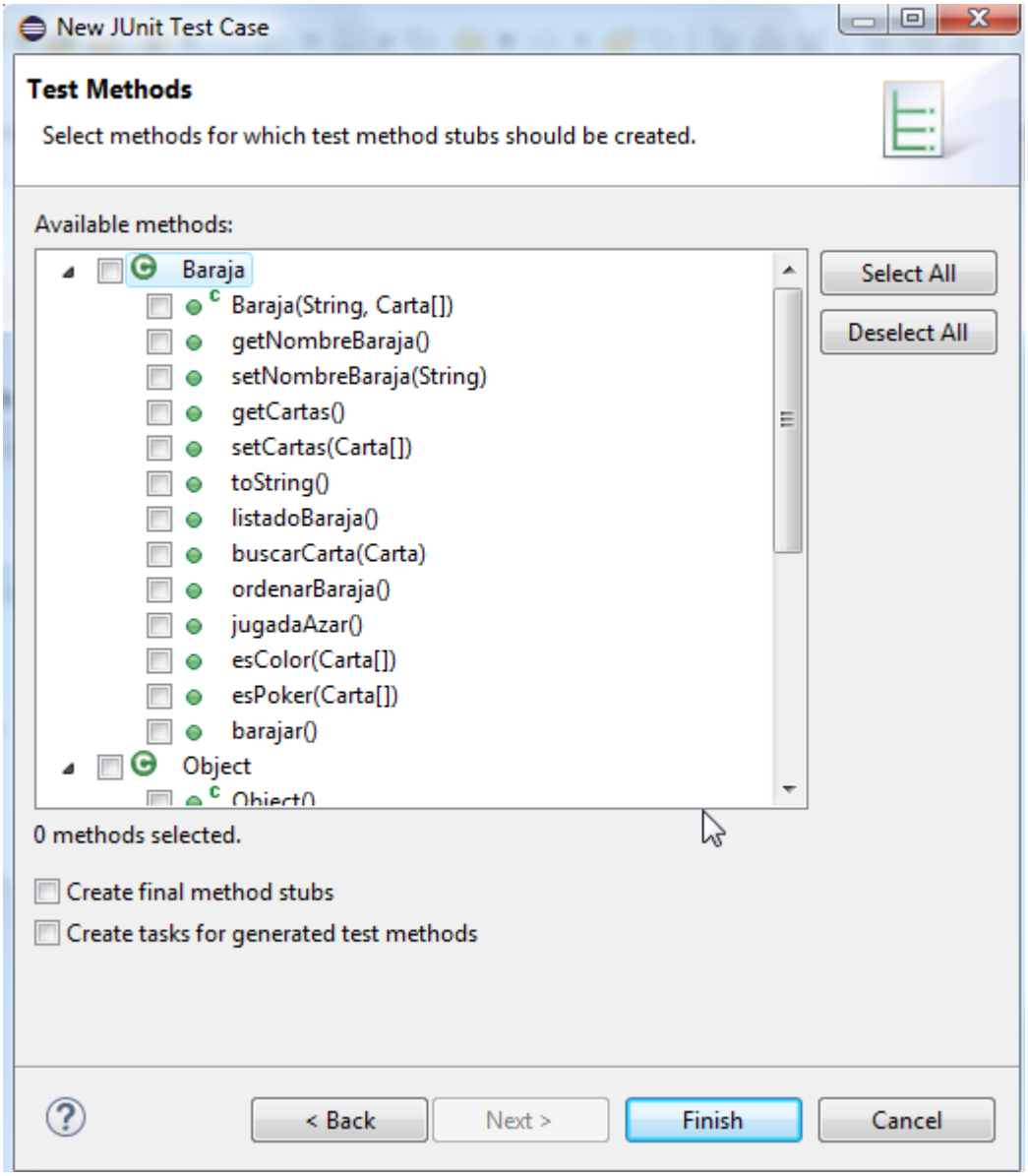
## Pruebas JUnit en Eclipse

Eclipse facilita la tarea de crear y ejecutar pruebas unitarias utilizando el framework JUnit [www.junit.org]. Para poder utilizar el framework JUnit, es necesario colocar la librería junit.jar en el classpath del proyecto.

El plugin JDT incluye un wizard (asistente) para la creación de casos de prueba (JUnit Test Cases) muy similar al propio wizard de creación de clases, explicado en este mismo documento.

Crear un nuevo TestCase, como decía, es muy similar a crear una nueva clase. De igual forma, se puede utilizar el botón de creación de clases (en la barra de herramientas principal, con la Perspectiva Java activa). En el menú desplegable que se muestra, en lugar de seleccionar Class o Interface, como se había explicado, se debe seleccionar la opción TestCase, lo cual mostrará el wizard JUnit.

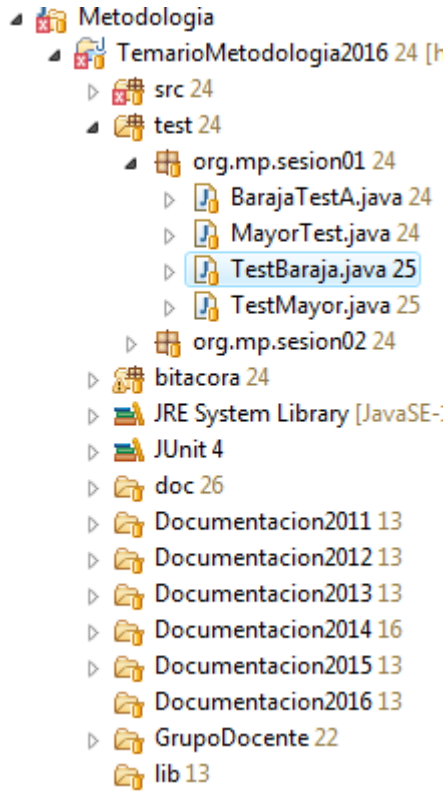




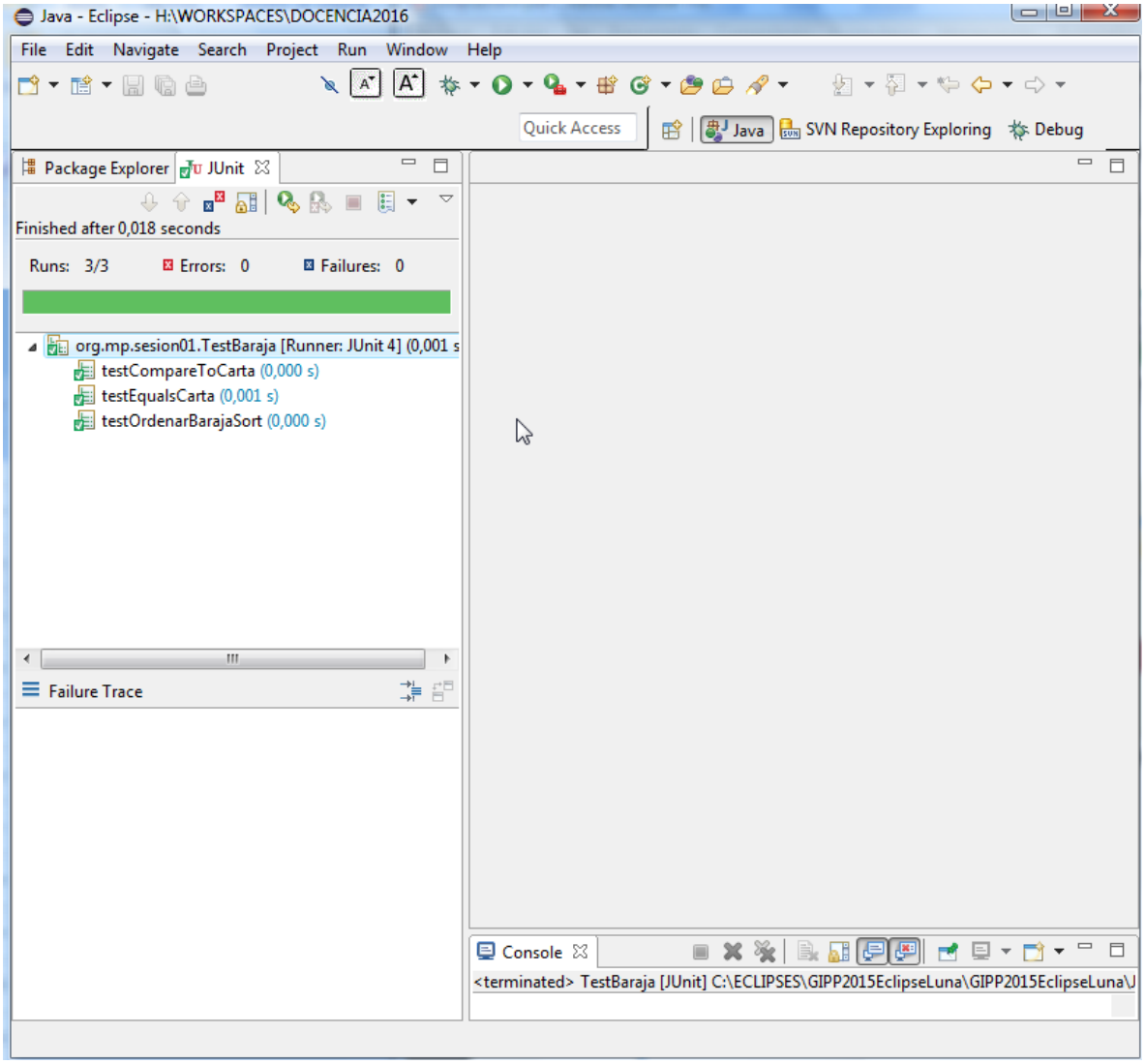

La plantilla más sencilla que genera es:

```
1  package org.mp.sesion01;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  public class TestBaraja1 {
9
10     @Before
11     public void setUp() throws Exception {
12     }
13
14     @Test
15     public void test() {
16         fail("Not yet implemented");
17     }
18
19 }
```

Para ejecutar el test sobre la clase, botón derecho → Run As → Junit Test




Y el resultado:



## Test suites

Hemos visto hasta ahora, una clase de prueba que contiene métodos de prueba, cada método contiene una o más sentencias de aserción. Pero una clase de prueba puede también invocar a otras clases de pruebas, clases individuales, paquetes o incluso todo el sistema.

La mágia se logra mediante la creación de *test suites*, batería de pruebas. Cualquier clase de prueba puede contener un método estático llamado *suite*

```
public static Test suite();
```

Podemos proporcionar un método *suite* que devuelva una colección de test. JUnit ejecuta todos los métodos *test* automáticamente.