Tests de unidad
Junit

Tests de unidad

- Los tests de unidad son pruebas que se realizan sobre una unidad de un programa.
- Una unidad puede ser un método, una clase o un módulo entero.
- En aplicaciones en entornos complejos además se realizan tests de integración, que prueban la integración de los diferentes módulos, test de sistema, que prueban un sistema entero, y tests de integración de sistemas, que prueban la correcta interconexión de los diferentes sistemas.

JUnit

- Junit es el framework más utilizado en Java para automatizar los tests de unidad.
- Permite programar tests de unidad utilizando una interfaz sencilla.
- Está integrado en Eclipse.
- Eclipse Luna tiene JUnit 4.
- Podemos descargar JUnit independientemente de Eclipse e integrarlo con ant o Maven.

JUnit

- Un test de unidad debe poder ejecutarse de forma automática.
- Las pruebas automáticas reducen riesgos y ahorran costes respecto a hacerlas manualmente.

TestCase

```
public class TestString {

@Test

public void testConcat() {

   String ies = "IES";

   String saladillo="Saladillo";

   String concatenada=ies.concat(saladillo);

   assertEquals("IESSaladillo", concatenada);

}


}
```

TestCase

- Los métodos de pruebas en Junit 3, debían de contener la palabra “test”, era así como JUnit los reconocía, en la versión 4 se utiliza la anotación @test antes del método.
- Los métodos de test devolverán void y no contendrán argumentos.
- En Eclipse aparecerá un árbol con los tests de unidad y los resultados de los diferentes tests de unidad.

TestCase

En cada método ponemos aserciones que se deban de cumplir, podemos tener una o varias.

```
Assert.assertEquals("IESSaladillo", concatenada);
```

Para superar un test se deben de cumplir todas las aserciones.

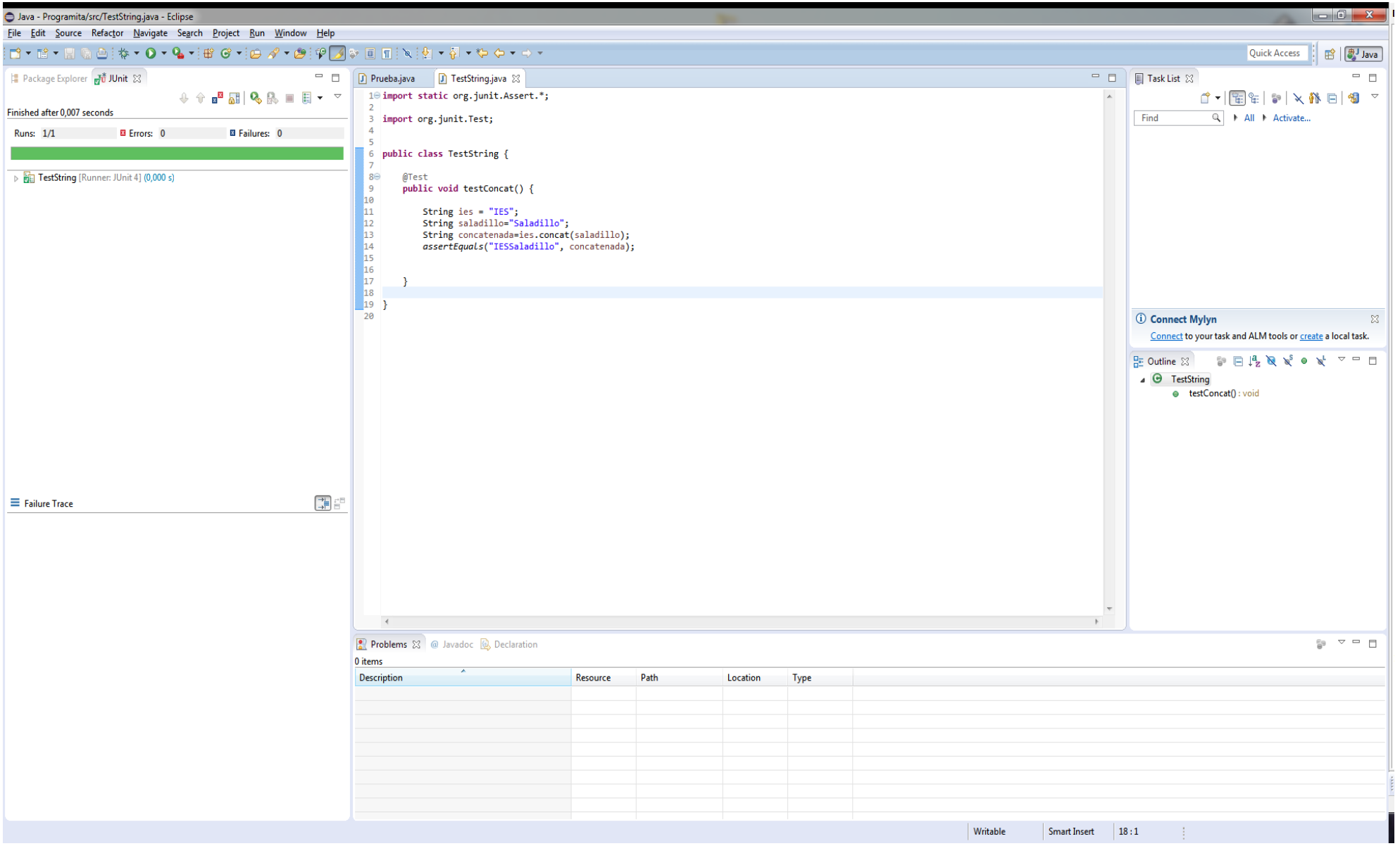
Asserts

- Existen múltiples métodos con diferentes firmas en Assert, algunos de ellos son:

- assertEquals(expected, actual)
- assertEquals(message, expected, actual)
- assertEquals(expected, actual, delta)
- assertFalse(condition)
- assertFalse(message, condition)
- Assert(Not)Null(object)
- Assert(Not)Null(message, object)
- Assert(Not)Same(expected, actual)
- Assert(Not)Same(message, expected, actual)
- assertTrue(condition)
- assertTrue(message, condition)

Asserts

- Las más utilizadas son assertTrue y assertEquals.
- Para lanzar un test con “Run as” nos aparecerá la opción de Junit en Eclipse.


Java - Programita/src/TestString.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Package Explorer
JUnit
Finished after 0,007 seconds
Runs: 1/1
Errors: 0
Failures: 0
TestString [Runner: JUnit 4] (0,000 s)
Failure Trace
Prueba.java
TestString.java
1 import static org.junit.Assert.*;
2
3 import org.junit.Test;
4
5
6 public class TestString {
7
8 @Test
9 public void testConcat() {
10
11 String ies = "IES";
12 String saladillo="Saladillo";
13 String concatenada=ies.concat(saladillo);
14 assertEquals("IESSaladillo", concatenada);
15
16
17 }
18
19 }
20
Task List
Find
All
Activate...
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
TestString
testConcat() : void
Problems
Javadoc
Declaration
0 items
Description
Resource
Path
Location
Type
Writable
Smart Insert
18 : 1

SetUp y tearDown

- En ocasiones puede ser interesante ejecutar cierto código antes o después del test de unidad. Los métodos setUp() y tearDown() tienen este propósito.Por ejemplo para testar una clase que acceda a una base de datos podríamos establecer la conexión en setUp() y finalizarla en tearDown().
- Junit 4 soporta anotaciones para setUp y tearDown, se puede usar @Before y @After.