

Programación Frontend y Backend

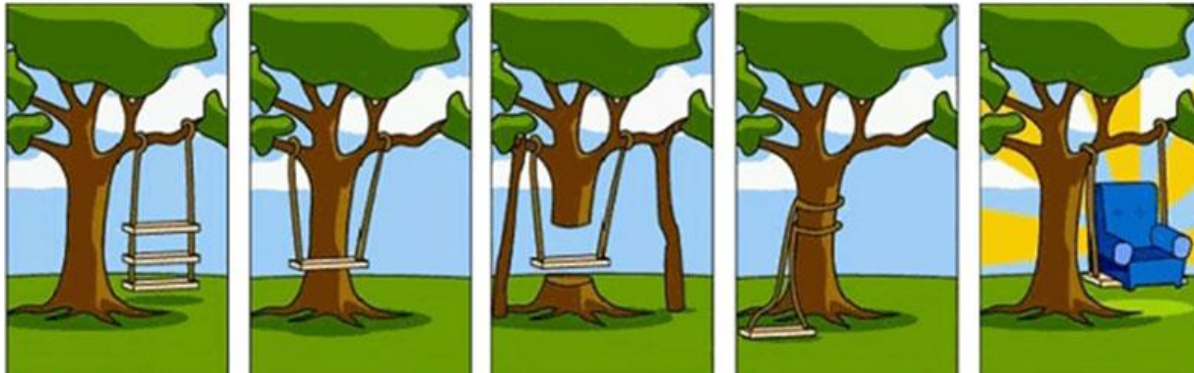
BLOQUE JAVA

Java Testing

Testing

Cuando hablamos de Testing, hablamos de un proceso que nos permite evaluar la funcionalidad de una aplicación con la intención de determinar si el software desarrollado cumple con los requisitos especificados e identificar los defectos para garantizar que el producto esté libre de defecto, este proceso aporta calidad al trabajo realizado.

Uno de los objetivos del testing consiste en identificar las diferencias entre la funcionalidad creada y la funcionalidad requerida o deseada.



Tipos de Pruebas

- Test Unitarios Unit Test
- Test Integrados Integrated Test
- Test de Regresión Regression Test

Test Unitarios

La misión principal de este tipo de prueba consiste en ejecutar cada módulo (o unidad mínima a ser probada) lo que provee un mejor modo de manejar la integración de las unidades en componentes mayores.



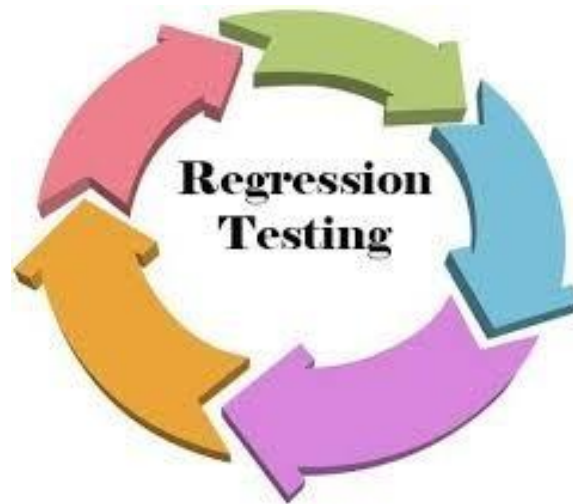
Test Integrados

Una vez hemos superado las pruebas individuales de cada módulo (Test Unitarios) ahora el objetivo consiste en identificar errores introducidos por la combinación de programas probados unitariamente.



Test Regresión

Determinar si los cambios recientes en una parte de la aplicación tienen efecto adverso en otras partes. La prueba de viejas funcionalidades es más importante que la de nuevas funcionalidades.



Test Unitarios

- Unitarios (Unidad)
- Automáticos, Repetibles y Reutilizables
- Cobertura de código
- Profesional (Importante)
- Contexto Independiente

Estrategias

Caja Blanca

Se basa en la estructura interna del código de las aplicaciones. En las pruebas de caja blanca, se utiliza una perspectiva interna del sistema, así como las habilidades de programación, para diseñar casos de prueba. Esta prueba se realiza generalmente en el nivel de la unidad.

Caja Negra

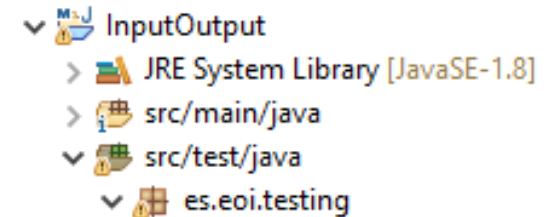
Es un método de prueba de software en el que los evaluadores evalúan la funcionalidad del software bajo prueba sin mirar la estructura del código interno.

JUnit

Crearemos en src/test/java el paquete es.eoi.testing

Una vez creado el paquete creamos el archivo test:

New > Other > Junit Test Case > Junit 4.0



```
3+ import static org.junit.Assert.assertEquals;
9
10 public class TestEjercicio1 {
11
12     @Test
13     public void test() {
14         try {
15             int total = Ejercicio1.suma(1, 1);
16             assertEquals(total, 2);
17         } catch (Exception e) {
18             fail("Not yet implemented");
19         }
20     }
21
22 }
23
```

JUnit

Assertions (Afirmaciones)

<https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html>

assertEquals

assertArrayEquals

assertNotNull

assertNull

assertNotSame

assertSame

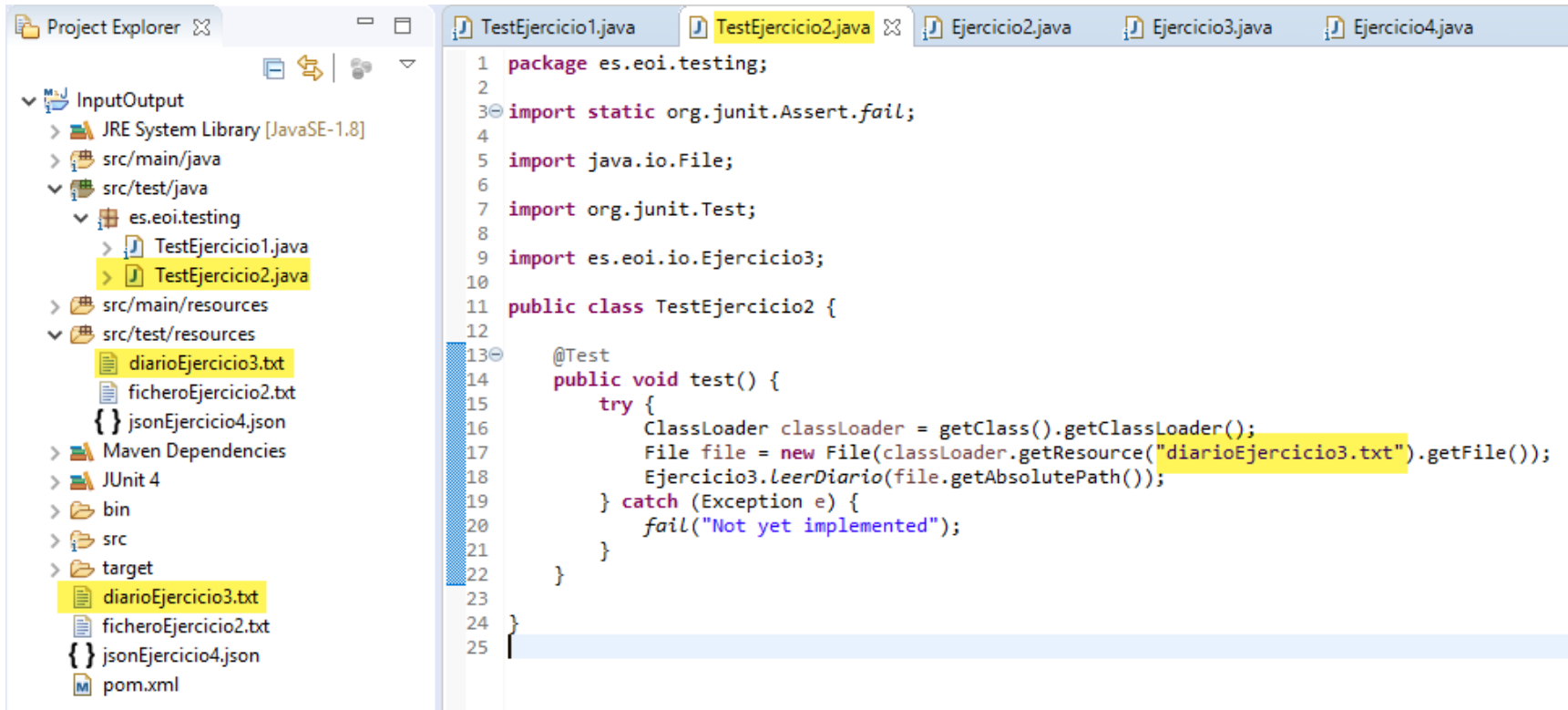
assertTrue

assertTrue

...

JUnit

Si queremos utilizar recursos, utilizaremos los que hayan en src/test/resources



The screenshot shows an IDE with a Project Explorer on the left and a code editor on the right. The Project Explorer displays the project structure, including the 'src/test/resources' directory which contains files like 'diarioEjercicio3.txt', 'ficheroEjercicio2.txt', and 'jsonEjercicio4.json'. The code editor shows the 'TestEjercicio2.java' file, which is a JUnit test class. The code imports 'org.junit.Assert.fail' and 'java.io.File', and uses 'org.junit.Test' to annotate a test method. The test method 'test()' attempts to load a resource file 'diarioEjercicio3.txt' from the classpath and calls 'Ejercicio3.leerDiario()' on it. If an exception occurs, it fails the test with the message 'Not yet implemented'.

```
1 package es.eoi.testing;
2
3 import static org.junit.Assert.fail;
4
5 import java.io.File;
6
7 import org.junit.Test;
8
9 import es.eoi.io.Ejercicio3;
10
11 public class TestEjercicio2 {
12
13     @Test
14     public void test() {
15         try {
16             ClassLoader classLoader = getClass().getClassLoader();
17             File file = new File(classLoader.getResource("diarioEjercicio3.txt").getFile());
18             Ejercicio3.leerDiario(file.getAbsolutePath());
19         } catch (Exception e) {
20             fail("Not yet implemented");
21         }
22     }
23 }
24
25 }
```

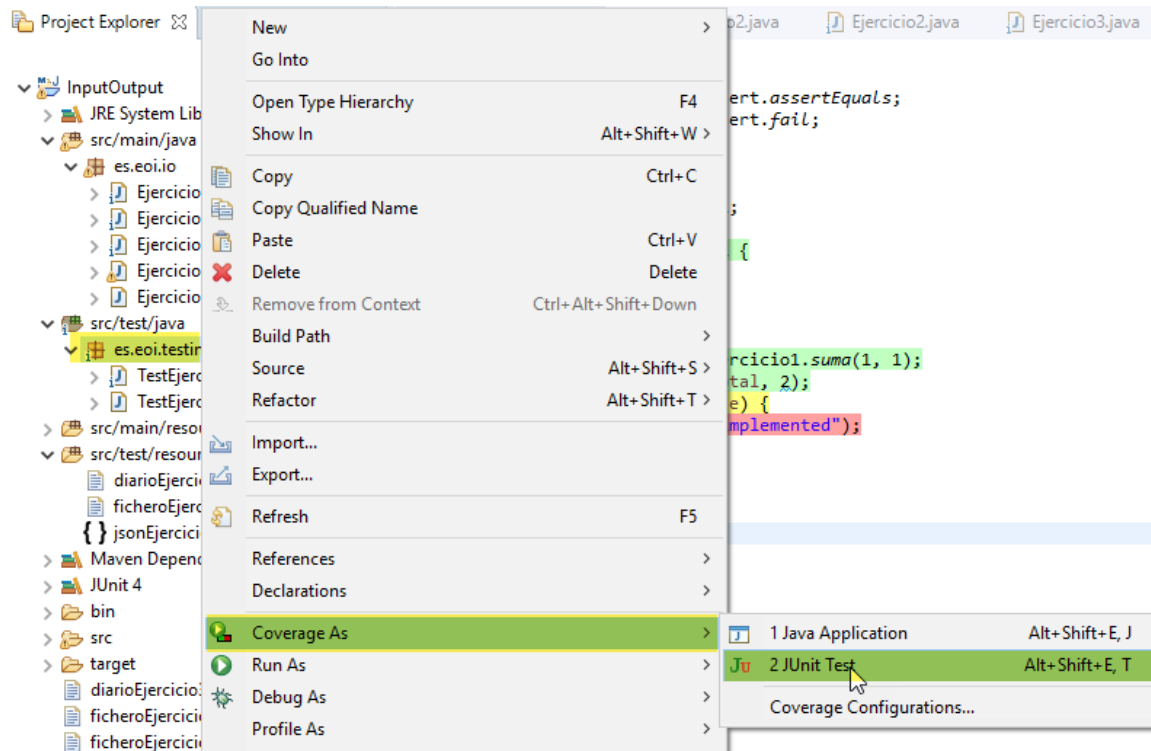
JUnit

Cubrir el mayor porcentaje de código es muy importante:

```
switch(a) {  
    case 1:  
        if(b == 0) {  
            System.out.println(1);  
        }  
        break;  
    case 2:  
        if(b == 1) {  
            System.out.println(2);  
        }  
        break;  
    default:  
        if(b == 1) {  
            System.out.println(3);  
        } else {  
            System.out.println(4);  
        }  
        break;  
}
```

EclEmma

Instalaremos el plugin si no lo tenemos desde Help > Eclipse Marketplace



Testing

The screenshot displays an IDE interface with two main panels. The left panel shows the JUnit test runner results, indicating that the tests passed successfully. The right panel shows a coverage report for the project.

JUnit Test Results:

Finished after 0,054 seconds

Runs: 2/2 Errors: 0 Failures: 0

- > es.eoi.testing.TestEjercicio2 [Runner: JUnit 4] (0,001 s)
- > es.eoi.testing.TestEjercicio1 [Runner: JUnit 4] (0,000 s)

Coverage Report:

Element	Coverage	Covered Instructio...	M
InputOutput	9,3 %	66	
src/main/java	4,9 %	33	
es.eoi.io	4,9 %	33	
Ejercicio4.java	0,0 %	0	
Ejercicio3.java	24,4 %	29	
Ejercicio2.java	0,0 %	0	
EjercicioA.java	0,0 %	0	
Ejercicio1.java	10,5 %	4	
src/test/java	84,6 %	33	
es.eoi.testing	84,6 %	33	
TestEjercicio1.java	81,2 %	13	
TestEjercicio2.java	87,0 %	20	

Ejercicios

- Vamos a modificar el ejercicio de la calculadora (Ejercicio 1 – Calculadora) para habilitar la suma, resta, multiplicación y división de dos números, para ello crearemos las funciones necesarias que nos permitirán cubrir el mayor porcentaje de código. Crearemos los métodos: suma(int a, int b), resta(int a, int b), multiplica(int a, int b) y divide (int a, int b). Por último crearemos una clase de test que cubra mediante test unitarios el mayor porcentaje de código.
- Vamos a modificar el resto de ejercicios para cubrir el mayor porcentaje de código posible, para ello debemos separar el código correctamente en las funciones que sean necesarias
- Por último, en ocasiones no solo haremos o modificaremos los test unitarios de nuestro código por lo que se os va a proporcionar una serie de procedimientos que debéis intentar cubrir en su totalidad.