**Trabajo**

**De**

**JUnit**

**Alumno: Ramón Fernando Pérez**

**Curso: 1ro DAW FP Superior**

**Asignatura: Entorno de Desarrollo**

**Profesor: Jaume**

**Indice**

**Que es un test Unitario**

**Que se espera con las pruebas unitarias**

**Que es JUnit**

**Aplicando JUnit en Eclipse**

**@Before y @After, @BeforClass y @AfterClass**

**@TestTimeO**

**@Parameters y @RunWith**

**@SuiteClasses**

**Metodos:**

**Que es un test Unitario**

**JUnit** es un conjunto de bibliotecas creadas por [Erich Gamma](https://es.wikipedia.org/wiki/Erich_Gamma) y [Kent Beck](https://es.wikipedia.org/wiki/Kent_Beck) que son utilizadas en [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n) para hacer [pruebas unitarias](https://es.wikipedia.org/wiki/Prueba_unitaria) de aplicaciones [Java](https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java).

Los test unitarios son para probar el correcto funcionamiento de unidades o módulos de código, esto nos sirve para asegurarnos de que cada una de nuestras funciones o clases o módulos funcionan correctamente de forma independiente. El valor de este tipo de test es que son automáticos e instantáneos de manera que en cualquier momento nos permite saber si nuestra aplicación está cumpliendo con el funcionamiento esperado. Por ejemplo esto nos permite limpiar modificar o extender nuestro código con la seguridad de que el funcionamiento va a seguir siendo correcto después de los cambios, es por tanto una manera de mejorar la calidad del software.

**Que se espera con las pruebas unitarias:**

El testing de software es el arte de medir y mantener la calidad del software para asegurar que las expectativas y requerimientos del usuario, el valor de negocio, los requisitos no funcionales como la seguridad, confiabilidad y tolerancia a fallos, y las políticas operacionales se cumplan. El testing es un esfuerzo del equipo para conseguir un mínimo de calidad y tener una “definición de hecho” entendida y aceptada por todos.

**Que es JUnit:**

**JUnit** es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.

**Aplicando JUnit en Eclipse:**

Esta es la clase principal a la cual se le hacen los test.

En la clase principal hay contenida una subclase UsuarioAutenticado (es la que describe el usuario autenticado en el sistema) de esta se genera la instancia cuando se ejecuta el método login, y va a contener el token identificador del usuario autenticado.

Nota: La autenticación basada en token no es más que una variable que contiene valores identificativos del usuario encriptados.

**package** es.proyecto.autenticacion;

/\*

\* Clase Sistema => contenedor para el login de usuario

\* \*/

**public** **class** Sistema {

**public** String getUsername() {

**return** username;

}

**public** String getPassword() {

**return** password;

}

//parametros de usuario

**private** String username;

**private** String password;

//Instancia de usuario autenticado

**private** UsuarioAutenticado logged;

**public** UsuarioAutenticado getLogged() {

**return** logged;

}

**public** **void** setLogged(UsuarioAutenticado logged) {

**this**.logged = logged;

}

**public** Sistema(String username, String password) {

**super**();

//definir juego de datos estáticos sencillos

//usuario y password

**this**.username = username;

**this**.password = password;

}

**public** UsuarioAutenticado Login(String username, String password) **throws** Exception{

//formar la instancia de usuario autenticado

**if**(username.isEmpty() || password.isEmpty())

**throw** **new** NullPointerException("Algunos de los datos son vacios");

**if**(**this**.username.equals(username) && **this**.password.equals(password)){

//crear la instancia de usuario autenticado por token

logged = **new** UsuarioAutenticado(encriptarDatos());

**return** logged;

}

**throw** **new** Exception("Datos proporcionados incorrectos!");

}

//metodo sencillo q simula encriptacion pero q no lo hace

**private** **int** encriptarDatos(){

**return** String.*format*("{0}{1}",username,password).getBytes().hashCode();

}

**public** **void** Logout(){

**this**.logged = **null**;

}

**public** **class** UsuarioAutenticado{

**private** **int** token;

**public** **int** getToken() {

**return** token;

}

**public** UsuarioAutenticado(**int** token) {

**super**();

**this**.token = token;

}

}

}

**Esta es una de las clases Test:**

Esta clase contiene pruebas unitarias básicas a la clase Sistema y sus funcionalidades.

**package** es.proyecto.autentitacion.tests;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.junit.Assert.*assertNotNull*;

**import** **static** org.junit.Assert.*assertNull*;

**import** **static** org.junit.Assert.*assertTrue*;

**import** org.junit.Before;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**import** es.proyecto.autenticacion.Sistema;

**public** **class** SistemaTests {

**private** Sistema sistema;

@Before

**public** **void** setUp() **throws** Exception {

System.***out***.println("Iniciando instancia sistema");

sistema = **new** Sistema("Ramon", "123456");

}

//en este test se verifica q el usuario sea ramon

@Test

**public** **void** test() {

String user = sistema.getUsername();

String pass = sistema.getPassword();

*assertEquals*("Ramon", user);//assert => verificacion de resultados, hay distintas funcionalidades de assert

*assertEquals*("123456", pass);

*assertTrue*(!String.*valueOf*(pass).equalsIgnoreCase(""));

}

@Test(expected=Exception.**class**)

**public** **void** testPasswordErroneo() **throws** Exception {

sistema.Login("Ramon", "1234567");

}

@Test

**public** **void** testLoginOk() **throws** Exception {

*assertNotNull*(sistema.Login("Ramon", "123456"));

}

@Test(expected=NullPointerException.**class**)

**public** **void** testLoginException() **throws** Exception {

sistema.Login("", "123456");

}

@Test

**public** **void** testInstanciaUsuarioAutneticado() {

sistema.Logout();

*assertNull*(sistema.getLogged());

}

}

**package** es.proyecto.autentitacion.tests;

**import** **static** org.junit.Assert.\*;

**import** java.lang.reflect.Array;

**import** java.util.Arrays;

**import** org.junit.After;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

**import** org.junit.runners.Parameterized.Parameters;

**import** org.junit.runners.Suite.SuiteClasses;

**import** es.proyecto.autenticacion.Sistema;

@RunWith(value=Parameterized.**class**)

**public** **class** SistemaParametrosTests {

Sistema sistema;

**public** SistemaParametrosTests(String username, String password) {

**super**();

//

sistema = **new** Sistema(username, password);

}

@Test

**public** **void** ComprobarInstanciasistema() {

*assertNotNull*(sistema);

}

@Test

**public** **void** testLoginOk() **throws** Exception {

*assertNotNull*(sistema.Login("Fernando", "q112312").getToken());

}

@Parameters

**public** **static** Iterable<Object[]> Parametros(){

**return** Arrays.*asList*(**new** Object[][]{

{"Fernando", "q112312"}});

}

@After

**public** **void** testfinalizado() **throws** Exception {

System.***out***.println("test finalizado");

}

}

**@Before y @After, @BeforClass y @AfterClass**

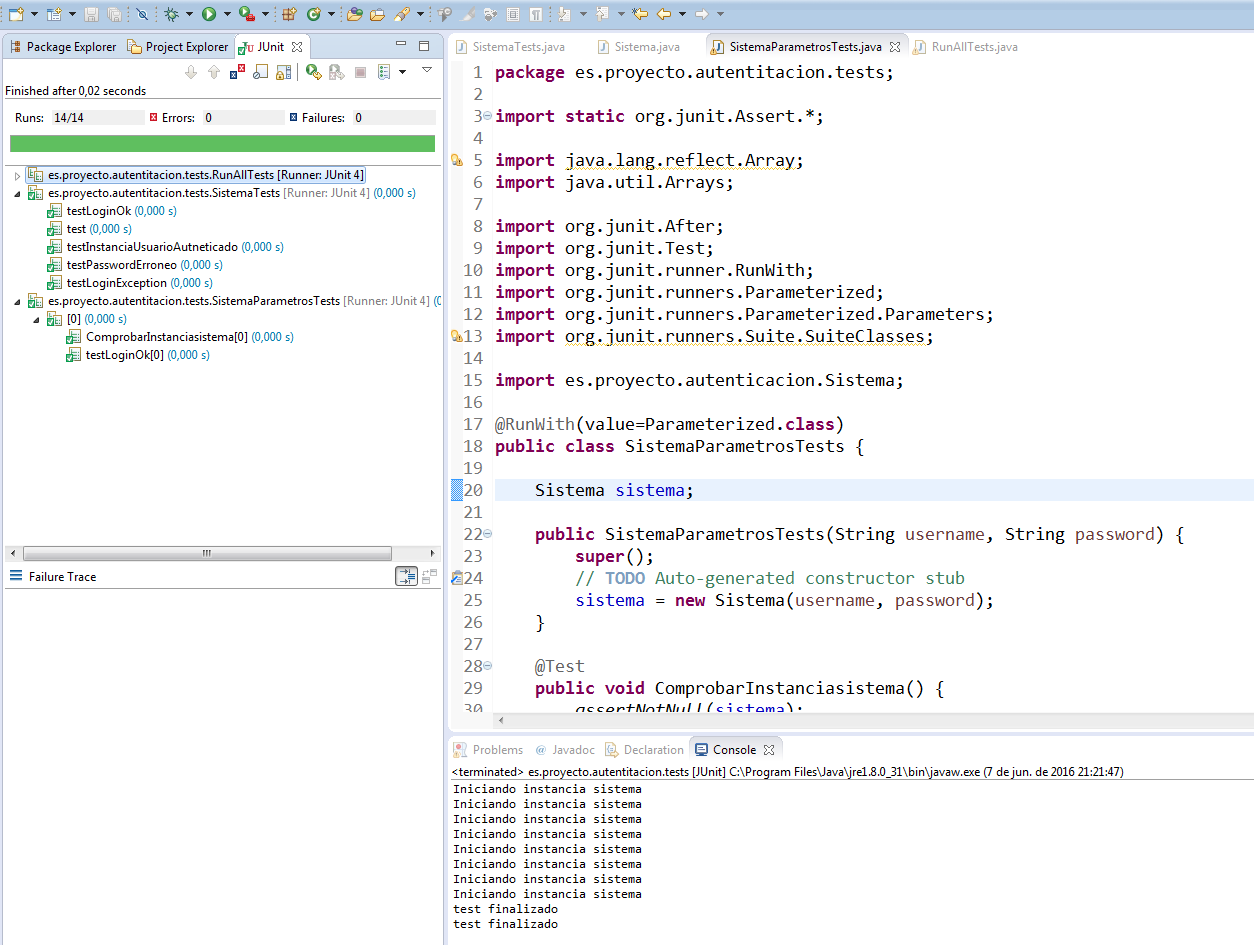
En Junit4 @Before se utiliza para ejecutar conjunto de condiciones previas antes de ejecutar una prueba. Por ejemplo, si hay una necesidad de abrir alguna aplicación y crear un usuario antes de ejecutar una prueba, a continuación, esta anotación se puede utilizar para ese método. Método que está marcado con @Before será ejecutado antes de ejecutar todas las pruebas en la clase.

Método que está marcado con @After es ejecutado después de la ejecución de cada prueba. Si tenemos que restablecer alguna variable después de la ejecución de todas las pruebas a continuación, esta anotación se puede utilizar con un método que tiene el código necesario.

Si una clase de casos de prueba JUnit contiene gran cantidad de pruebas que en su conjunto necesitan un método que establece una condición previa y que necesita ser ejecutado antes de ejecutar la clase de casos de prueba a continuación, podemos utilizar "@BeforeClass" anotación.

De la misma manera "@AfterClass" anotación se puede utilizar para ejecutar un método que necesita ser ejecutado después de ejecutar todas las pruebas en una clase de caso de prueba JUnit.

En el ejemplo anterior vimos el @Before y el @After, que mostraron por consola un mensaje antes y después de cada test.



**@TestTimeO:** Nos da la posibilidad de indicar la duración máxima de un test, si el tiempo dura más del indicado se para y se considera fallido.

Esto se implementa mediante la ejecución del método de ensayo en un hilo separado. Si la prueba que se ejecuta es más larga que el tiempo de espera asignado, la prueba fallará y JUnit interrumpirá el hilo conductor de la prueba. Si la duración de la prueba a caba durante la ejecución de una operación interrumpible, el hilo que ejecutara la prueba saldrá (si la prueba está en un bucle infinito, el hilo conductor de la prueba se ejecutará siempre, mientras que otras pruebas se sigan ejecutando).

@Test(timeout=1000) public void testForever(){

while(true) {

}

}

**@Parameters y @RunWith:** En algunos casos puede ser útil utilizar diferentes conjuntos de valores de entrada para el mismo test. Ésto nos permite testear la misma funcionalidad con diferentes datos sin tener que reescribir el test varias veces. JUnit nos ofrece esta posibilidad con la combinación de los decoradores @RunWith y @Parameters. En el anterior ejercicio hay un ejemplo de ambos.

@RunWith(value=Parameterized.**class**)

**public** **class** SistemaParametrosTests{

Sistema sistema;

**public** SistemaParametrosTests(String username, String password) {

**super**();

//

sistema = **new** Sistema(username, password);

}

@Parameters

**public** **static** Iterable<Object[]> Parametros(){

**return** Arrays.*asList*(**new** Object[][]{

{"Fernando", "q112312"}});

}

**@SuiteClasses:** Permite agrupar diferentes clases de JUnit para ejecutarlas todas. Para crearla: New -> Other -> Java -> JUnit -> JUnit Test Suite -> Next -> (Nombre de la clase) -> Finish Y nos crearía el siguiente código. import org.junit.runner.RunWith; import org.junit.runners.Suite; import org.junit.runners.Suite.SuiteClasses; @RunWith(Suite.class) @SuiteClasses({ CalculadoraParametersTest.class, CalculadoraTest.class }) public class AllTests { }

**package** es.proyecto.autentitacion.tests;

**import** org.junit.After;

**import** org.junit.AfterClass;

**import** org.junit.Before;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Suite;

**import** org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.**class**)

@SuiteClasses({ SistemaParametrosTests.**class**, SistemaTests.**class** })

**public** **class** RunAllTests {

}

**Metodos:**

JUnit dispone de los siguientes test para hacer comprobaciones:

|  |  |
| --- | --- |
| **Método assertxxx() de JUnit** | **Qué comprueba** |
| assertTrue(expresión) | comprueba que expresión evalúe a true |
| assertFalse(expresión) | comprueba que expresión evalúe a false |
| assertEquals(esperado,real) | comprueba que esperado sea igual a real |
| assertNull(objeto) | comprueba que objeto sea null |
| assertNotNull(objeto) | comprueba que objeto no sea null |
| assertSame(objeto\_esperado,objeto\_real) | comprueba que objeto\_esperado y objeto\_real sean el mismo objeto |
| assertNotSame(objeto\_esperado,objeto\_real) | comprueba que objeto\_esperado no sea el mismo objeto que objeto\_real |
| fail() | hace que el test termine con fallo |