JUnit & Mockito

# Descripción:

JUnit es un framework de pruebas unitarias de Java. Una prueba unitaria es el testeo mediante software (automatizado) de una pequeña parte de una aplicación, por ejemplo una clase o un método.

Idealmente un caso de prueba es independiente de otros, para lo cual se puede usar código para simular a otro objeto o método.

Mockito es un framework que simplifica la simulación de objetos, permitiendo hacer casos de prueba independientes.

# Dependencias:

JUnit 5:

<dependency>

    <groupId>org.junit.jupiter</groupId>

    <artifactId>junit-jupiter-api</artifactId>

    <version>5.1.0</version>

    <scope>test</scope>

</dependency>

Si estamos usando un IDE que tiene integrada una versión más vieja de JUnit, hay que agregar las siguientes dependencias para poder correr los tests:

    <dependency>

    <groupId>org.junit.jupiter</groupId>

    <artifactId>junit-jupiter-engine</artifactId>

    <version>5.1.0</version>

    <scope>test</scope>

</dependency>

<dependency>

    <groupId>org.junit.vintage</groupId>

    <artifactId>junit-vintage-engine</artifactId>

    <version>5.1.0</version>

    <scope>test</scope>

    </dependency>

<dependency>

    <groupId>org.junit.platform</groupId>

    <artifactId>junit-platform-launcher</artifactId>

    <version>1.1.0</version>

    <scope>test</scope>

</dependency>

<dependency>

    <groupId>org.junit.platform</groupId>

    <artifactId>junit-platform-runner</artifactId>

    <version>1.1.0</version>

    <scope>test</scope>

</dependency>

Mockito:

<!-- https://mvnrepository.com/artifact/org.mockito/mockito-core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>2.21.0</version>

<scope>test</scope>

</dependency>

# Anotaciones y ejemplos:

## JUnit:

**@Test:**

Se usa para marcar los métodos a testear.

@Test

Public void suma\_test(){

AssertEquals(2 , metodosuma(1,1)); // En este caso el resultado esperado es 2, y el actual es lo que devuelve el método suma.

}

**@RepeatedTest:**

Se usa para hacer test que se repitan X cantidad de veces. Cada iteración se comportara como un @Test independiente, esto significa que las anotaciones que se ejecuten antes o después de cada test se ejecutaran en cada iteración.

@RepeatedTest(10) // Se repetirá 10 veces

Public void suma\_test(){

AssertEquals(2 , metodosuma(1,1));

}

**@BeforeEach:**

Lo que este debajo de esta anotacion se ejecuta una vez antes de empezar cada test (en JUnit 4 se usaba @Before), se puede usar para reiniciar un objeto necesario para varios tests.

Public class Test {

List listaVacia;

@BeforeEach

Public void initialize(){

listaVacia = new ArrayList(); }}

**@BeforeAll:**

Esta anotación indica que lo que esta abajo se ejecutara una única vez antes de comenzar los tests, sirve para inicializar algo. Este método debe ser estatico

Public class Test {

List listaVacia;

@BeforeAll

Public static void initialize(){

listaVacia = new ArrayList();

listaVacia.add(1);

}}

**@AfterEach:**

Igual que @BeforeEach pero luego de cada test. Uno de sus usos por ejemplo es hace rollback a un cambio en la base de datos.

@AfterEach

void tearDown(TestInfo testInfo) {

System.out.println("Finished..." + testInfo.getDisplayName()); // getDisplayName() muestra el nombre que le pusimos al test mediante @DisplayName(“Ejemplo Test”)

}

**@AfterAll:**

Igual que @BeforeAll pero luego de ejecutar todos los tests.

@AfterAll

void fin(){

System.out.println(“Tests finished”);}

**@DisplayName:**

Sirve para ponerle un nombre a un caso de prueba.

@Test

@DisplayName(“Test Prueba 1”)

Test method(){}

## Mockito:

**@Mock:**

Se usa para escribir menos codigo a la hora de mockear objetos y hacer mas fácil de leer el código.

**Sin @Mock:**

List mockedList = mock(List.class);

**Con @Mock:**

@Mock List mockedList;

**@MockBean:**

Esta anotacion añade mocks al SpringApplicationContext y ademas inyecta estos mocks en los campos correspondientes. Se puede utilizar para mockear servicios.

# Métodos más usados:

## JUnit:

**Assert:**

Este método se usa para realizar las comprobaciones que queremos en nuestros tests, existen distintas variantes de assert como por ej assertNotNull, AssertThat, AssertEquals, etc.

assertEquals(60d,suma(30d,30d));

assertNotNull( traerInformacion() );

assertThrows(ArithmeticException.class,

()->{

calculatorService.operate(mock);

}); // En este caso mock contiene 5 / 0, lo cual arroja una arithmetic exception.

## Mockito:

**When:**

Este método se usa para mockear resultados de otros métodos (ajenos a lo que queramos testear)

when(calculatorService.operate(myQueue)).thenReturn(3d);

// En este caso decimos que cuando calculatorService ejecute el método operate recibiendo myQueue como parámetro este método devuelva un 3 double, sin importar el contenido de myQueue. Este método fue testeado con valores reales en su clase correspondiente.

# Links útiles:

<http://static.javadoc.io/org.mockito/mockito-core/2.21.0/org/mockito/Mockito.html#1>

<https://junit.org/junit5/>