**Testing en Java**

///////////////////////////////////////////

En Maven:

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

PasswordUtil.java

Public class PasswordUtil {

public enum SecurityLevel {

WEAK, MEDIUM, STRONG

}

public static SecurityLevel assessPassword(String password) {

if (password.length() < 8 || password.matches("[a-zA-Z]+")) return SecurityLevel.WEAK;

if (password.matches("[a-zA-Z0-9]+")) return SecurityLevel.MEDIUM;

return SecurityLevel.STRONG;

}

/////////////////////////////////////////////

package com.platzi.javatests.util;

import org.junit.Test;

import static com.platzi.javatests.util.PasswordUtil.SecurityLevel.\*;

import static org.junit.Assert.assertEquals;

public class PasswordUtilTest {

@Test

public void assessPasswordWeak\_whenHasLessThanEightLetters() {

assertEquals(WEAK, PasswordUtil.assessPassword("1234567"));

}

@Test

public void assessPasswordWeak\_whenHasOnlyLetters() {

assertEquals(WEAK, PasswordUtil.assessPassword("abcdefgh"));

}

@Test

public void assessPasswordMedium\_whenHasLettersAndNumbers() {

assertEquals(MEDIUM, PasswordUtil.assessPassword("abcd1234"));

}

@Test

public void assessPasswordStrong\_whenHasLettersNumbersAndSymbols() {

assertEquals(STRONG, PasswordUtil.assessPassword("abcd123#!"));

}

}

///////////////////////////////

**Instalación de IntelliJ IDEA, creación del Proyecto con Maven y Tests Unitarios**

En intelliJ > Create New Project > Maven > Next > en GroupId se pone el dominio de la empresa para la que se trabaja com.platzi >

Los proyectos Maven por defecto usan Java 5

Agregar:

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

En src está pom.xml

En src > main > java se crea un paquete: com.platzi.javatests, que tiene una clase de utilidad: StringUtil

public class StringUtil {

public static String repeat(String str, int times) {

StringBuilder result = new StringBuilder();

for (int i = 0; i < times; i++) {

result.append(str);

}

return result.toString();

}

}

/////////////////////////////////////

StringUtilTest .java

package com.platzi.javatests.util;

public class StringUtilTest {

public static void main(String[] args) {

String result = StringUtil.repeat("hola", 3);

if(result.equals("holaholahola")) {

System.out.println("OK");

} else {

System.out.println("ERROR");

}

String **result2** = StringUtil.repeat("hola", 1);

if(!**result2**.equals("hola")) {

System.out.println("OK");

} else {

System.out.println("ERROR");

}

}

}

////////////////////////////////////////////////

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

import org.junit.Test;

public class StringUtilTest {

@Test

public void testRepeat() {

assertEquals("holaholahola", StringUtil.repeat("hola", 3));

assertEquals("hola", StringUtil.repeat("hola", 1));

}

}

////////////////////////////////////////////////

class StringUtilTest {

@Test

void repeatStringPositiveTimes() {

assertEquals("HelloHelloHelloHelloHello", StringUtil.repeat("Hello", 5));

}

@Test

void repeatStringZeroTimes() {

assertEquals("", StringUtil.repeat("Hello", 0));

}

@Test

void repeatStringNegativeTimes() {

assertThrows(IllegalArgumentException.class, () -> StringUtil.repeat("Hello", -10));

}

}

if(times < 0){throw new IllegalArgumentException(“negative times not allowed”);}

@Test(expected = IllegalArgumentException.class)

////////////////////////////////////////////////////////////

Librería mockito

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.Mockito;

import org.mockito.junit.MockitoJUnitRunner;

import static org.junit.Assert.\*;

@RunWith(MockitoJUnitRunner.class)

public class PaymentProcessorTest {

@Mock

PaymentGateway paymentGateway;

@InjectMocks

private PaymentProcessor **paymentProcessor**;

@Test

public void payment\_is\_correct() {

Mockito.when(paymentGateway.requestPayment(Mockito.any()))

.thenReturn(new PaymentResponse(PaymentResponse.PaymentStatus.OK));

assertTrue(**paymentProcessor**.makePayment(1000));

}

@Test

public void payment\_is\_wrong() {

Mockito.when(paymentGateway.requestPayment(Mockito.any()))

.thenReturn(new PaymentResponse(PaymentResponse.PaymentStatus.ERROR));

assertFalse(**paymentProcessor**.makePayment(1000));

}

}

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>2.23.4</version>

<scope>test</scope>

</dependency>

public class Dice {

private Dice dice;

private int minNumberToWin;

public Dice(int sides) {

this.sides = sides;

}

public int roll() {

return new Random().nextInt();

}

public Player(Dice dice, int minNumberToWin) {

this.dice = dice;

this.minNumberToWin = minNumberToWin;

}

public boolean play() {

int diceNumber = dice.roll();

return diceNumber > minNumberToWin;

}

}

Crear un test:

Alt Enter sobre el nombre de la clase > create Test

@RunWith(MockitoJUnitRunner.class)

public class PlayerTest {

@Mock

Dice dice;

@InjectMocks

Player player = new Player();

@Test

public void lose\_when\_dice\_number\_is\_too\_low() {

when(dice.roll()).thenReturn(2);

player = new Player(dice, 3);

assertFalse(player.play());

}

@Test

public void wins\_when\_dice\_number\_is\_big() {

when(dice.roll()).thenReturn(4);

player = new Player(dice, 3);

assertTrue(player.play());

}

}

/////////////////////////////////////////////////////

Nuevo paquete payments

package com.platzi.javatests.payments;

public interface PaymentGateway {

PaymentResponse requestPayment(PaymentRequest request);

}

////////////////////////////////////////

PaymentRequest.java

package com.platzi.javatests.payments;

public class PaymentRequest {

private double amount;

public PaymentRequest(double amount) {

this.amount = amount;

}

public double getAmount() {

return amount;

}

}

///////////////////////////////////////////////////////

PaymentProcessor.java

package com.platzi.javatests.payments;

import static com.platzi.javatests.payments.PaymentResponse.PaymentStatus.OK;

public class PaymentProcessor {

private PaymentGateway paymentGateway;

public PaymentProcessor(PaymentGateway paymentGateway) {

this.paymentGateway = paymentGateway;

}

public boolean makePayment(double amount) {

PaymentResponse response = paymentGateway.requestPayment(new PaymentRequest(amount));

return response.getStatus() == OK;

}

}

/////////////////////////////////////

PaymentResponse.java

package com.platzi.javatests.payments;

public class PaymentResponse {

enum PaymentStatus {

OK, ERROR

}

private PaymentStatus status;

public PaymentResponse(PaymentStatus status) {

this.status = status;

}

public PaymentStatus getStatus() {

return status;

}

}

//////////////////////////////////

PaymentProcessorTest.java

package com.platzi.javatests.payments;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.junit.MockitoJUnitRunner;

import static com.platzi.javatests.payments.PaymentResponse.PaymentStatus.ERROR;

import static com.platzi.javatests.payments.PaymentResponse.PaymentStatus.OK;

import static org.junit.Assert.assertFalse;

import static org.junit.Assert.assertTrue;

import static org.mockito.Mockito.any;

import static org.mockito.Mockito.when;

@RunWith(MockitoJUnitRunner.class)

public class PaymentProcessorTest {

@InjectMocks

PaymentProcessor paymentProcessor;

@Mock

PaymentGateway paymentGateway;

@Test

public void makePayment\_whenPaymentIsCorrect() {

paymentProcessor = new PaymentProcessor(paymentGateway);

// any(): Simulate with mockito classes that can not be tested directly (PaymentRequest).

when(paymentGateway.requestPayment(any())).thenReturn(new PaymentResponse(OK));

assertTrue(paymentProcessor.makePayment(1000));

}

@Test

public void makePayment\_whenPaymentIsWrong() {

paymentProcessor = new PaymentProcessor(paymentGateway);

// any(): Simulate with mockito classes that can not be tested directly (PaymentRequest).

when(paymentGateway.requestPayment(any())).thenReturn(new PaymentResponse(ERROR));

assertFalse(paymentProcessor.makePayment(1000));

}

}

///////////////////////////////