***WEEK 2***

***3. Mockito Hands-On Exercises***

**Exercise 1: Mocking and Stubbing**

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**MyServiceTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

// 1. Create mock object

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

// 2. Stub the method

*when*(mockApi.getData()).thenReturn("Mock Data");

// 3. Inject mock into service

MyService service = new MyService(mockApi);

// 4. Call method and assert

String result = service.fetchData();

*assertEquals*("Mock Data", result);

}

}

**ExternalApi.java**

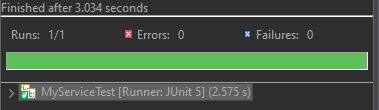
package com.example;

public interface ExternalApi {

String getData();

}

**OUTPUT**



**Exercise 2: Verifying Interactions**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**InteractionTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class InteractionTest {

*@Test*

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

MyService service = new MyService(mockApi);

// Act

service.fetchData();

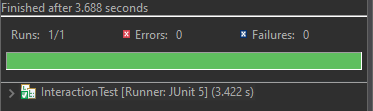
// Assert

*verify*(mockApi).getData(); // verifies that getData() was called once

}

}

**OUTPUT**

****

**Exercise 3: Argument Matching**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

void sendMessage(String recipient, String message); // <-- new method

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

public void sendGreeting(String name) {

externalApi.sendMessage(name, "Hello!");

}

}

**ArgumentMatchingTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class ArgumentMatchingTest {

*@Test*

public void testArgumentMatching() {

// 1. Create mock

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

// 2. Create service with mock

MyService service = new MyService(mockApi);

// 3. Call method with specific arguments

service.sendGreeting("Indhumathi");

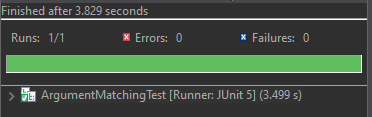
// 4. Verify the arguments

*verify*(mockApi).sendMessage(*eq*("Indhumathi"), *eq*("Hello!"));

}

}

**OUTPUT**

****

**Exercise 4: Handling Void Methods**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

void sendMessage(String recipient, String message);

void logMessage(String message); // <--- New void method

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

public void sendGreeting(String name) {

externalApi.sendMessage(name, "Hello!");

}

public void processLog(String msg) {

externalApi.logMessage(msg); // <--- calls the void method

}

}

**VoidMethodTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class VoidMethodTest {

*@Test*

public void testVoidMethod() {

// 1. Create mock

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

// 2. Stub the void method (optional unless exception needed)

// doNothing().when(mockApi).logMessage(anyString()); // optional

// 3. Inject mock and call method

MyService service = new MyService(mockApi);

service.processLog("Void method called");

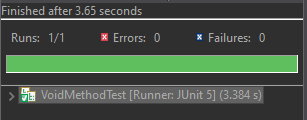
// 4. Verify interaction

*verify*(mockApi).logMessage("Void method called");

}

}

**OUTPUT**

****

**Exercise 5: Mocking and Stubbing with Multiple Returns**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

void sendMessage(String recipient, String message);

void logMessage(String message);

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String[] fetchMultipleData() {

return new String[] {

externalApi.getData(),

externalApi.getData(),

externalApi.getData()

};

}

}

**MultipleReturnsTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MultipleReturnsTest {

*@Test*

public void testMultipleReturns() {

// 1. Create mock

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

// 2. Stub getData() with multiple return values

*when*(mockApi.getData())

.thenReturn("First")

.thenReturn("Second")

.thenReturn("Third");

// 3. Inject mock into service

MyService service = new MyService(mockApi);

// 4. Call method and assert results

String[] results = service.fetchMultipleData();

*assertEquals*("First", results[0]);

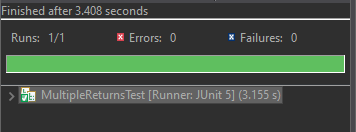
*assertEquals*("Second", results[1]);

*assertEquals*("Third", results[2]);

}

}

**OUTPUT**

****

**Exercise 6: Verifying Interaction Order**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

void connect();

void sendMessage(String recipient, String message);

void disconnect();

String getData();

void logMessage(String message);

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void performOperation() {

externalApi.connect();

externalApi.sendMessage("Indhumathi", "Hello!");

externalApi.disconnect();

}

}

**InteractionOrderTest.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

public class InteractionOrderTest {

*@Test*

public void testInteractionOrder() {

// 1. Create mock

ExternalApi mockApi = *mock*(ExternalApi.class);

// 2. Inject into service

MyService service = new MyService(mockApi);

// 3. Perform the operation

service.performOperation();

// 4. Create InOrder verifier

InOrder inOrder = *inOrder*(mockApi);

// 5. Verify call order

inOrder.verify(mockApi).connect();

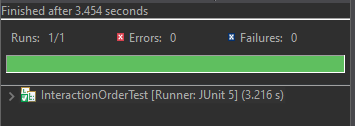
inOrder.verify(mockApi).sendMessage("Indhumathi", "Hello!");

inOrder.verify(mockApi).disconnect();

}

}

**OUTPUT**

****

**Exercise 7: Handling Void Methods with Exceptions**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

void logMessage(String message);

}

**MyService.java**

package com.example;

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public void safeLog(String message) {

try {

externalApi.logMessage(message);

} catch (RuntimeException e) {

System.***out***.println("Logging failed: " + e.getMessage());

}

}

}

**VoidException.java**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class VoidExceptionTest {

*@Test*

public void testVoidMethodThrowsException() {

// Create mock

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

// Stub the void method to throw exception

*doThrow*(new RuntimeException("Simulated failure"))

.when(mockApi).logMessage("Important log");

// Create service and call the method

MyService service = new MyService(mockApi);

service.safeLog("Important log");

// Verify interaction

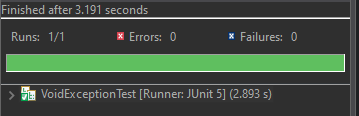
*verify*(mockApi).logMessage("Important log");

}

}

**OUTPUT**

****

****