Skill – TDD using JUnit5 and Mockito

# Mockito Hands-On Exercises

## Exercise 1: Mocking and Stubbing

Purpose:  
  
The purpose of this exercise is to demonstrate how to test a service that depends on an external API using Mockito by mocking the API and stubbing its methods.

### Understanding Mocking and Stubbing:

Mocking allows you to create simulated objects that mimic the behavior of real objects in controlled ways. Stubbing is the process of specifying what the mock should return when certain methods are called.

### Implementation:

## CODE:

## MockitoExample/pom.xml:

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>MockitoExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 4.13.2 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<!-- Mockito Core -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

<!-- Hamcrest for assertions -->

<dependency>

<groupId>org.hamcrest</groupId>

<artifactId>hamcrest</artifactId>

<version>2.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

## ExternalApi.java:

**public** **interface** ExternalApi {

String getData();

}

## MyService.java:

**public** **class** MyService {

**private** **final** ExternalApi externalApi;

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

**public** String fetchData() {

**return** externalApi.getData();

}

}

## MyServiceTest.java:

**import** org.junit.Test;

**import** **static** org.mockito.Mockito.\*;

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

**public** **class** MyServiceTest {

@Test

**public** **void** testExternalApi() {

// 1. Create a mock of ExternalApi

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

// 2. Stub getData() to return "Mock Data"

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

// 3. Pass mock into MyService

MyService service = **new** MyService(mockApi);

// 4. Call fetchData and verify result

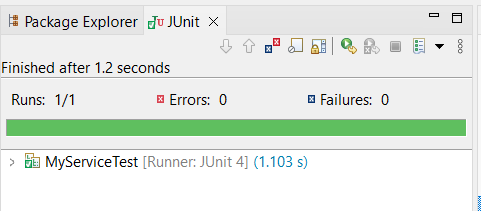
String result = service.fetchData();

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

}

}

### Output:



The test will pass if the fetchData method of MyService correctly returns "Mock Data" from the mocked API.

## Exercise 2: Verifying Interactions

## Purpose: The purpose of this exercise is to verify that certain methods of a mock object are invoked with expected arguments.

## Implementation:

## MyServiceTest.java:

**import** org.junit.Test;

**import** **static** org.mockito.Mockito.\*;

**public** **class** MyServiceTest {

@Test

**public** **void** testVerifyInteraction() {

// 1. Create mock

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

// 2. Inject mock into service

MyService service = **new** MyService(mockApi);

// 3. Call the method

service.fetchData();

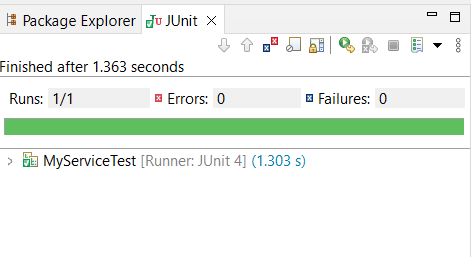
// 4. Verify interaction

*verify*(mockApi).getData();

}

}

### Output:



The test will pass if the getData method of the mock API was called exactly once during the execution of fetchData().