**WEEK-2**

**Mockito Exercises**

**Exercise 1: Mocking and Stubbing**

**Scenario:**

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Code:**

**Steps:**

1. Maven project is setup with group id: com.example, artifact id: MockExample
2. Add required dependency in **pom.xml**

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**3.ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**4.MyService.java**

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**5.MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

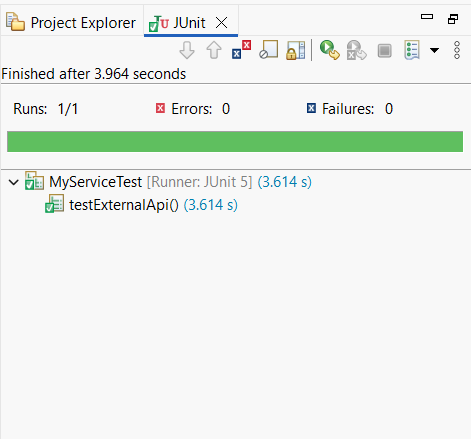
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**



**Explanation:**

* A tool called Mockito is used in Java testing to generate simulated copies of actual objects, particularly ones that rely on external systems like databases or APIs.
* For this work, I used mock() to create a mock of ExternalApi and when(...).thenReturn(...) to specify what its method should return.
* I developed a MyService class that makes use of ExternalApi, and during testing, it passed the mock version.
* In this manner, the test is independent of the actual external API and concentrates just on the service logic.
* Mocking and stubbing make it easier to safely test each component of the application independently.

**Exercise 2: Verifying Interactions**

**Scenario:**

You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

Solution Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApí);

service.fetchData();

verify(mockApi).getData();

}

}

**Code:**

**Steps:**

1. Maven Project Setup with group id: com.example artifact id: MockExample
2. Add required dependencies in **pom.xml**

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**3.ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**4.MyServices.java**

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**5.ServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class ServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

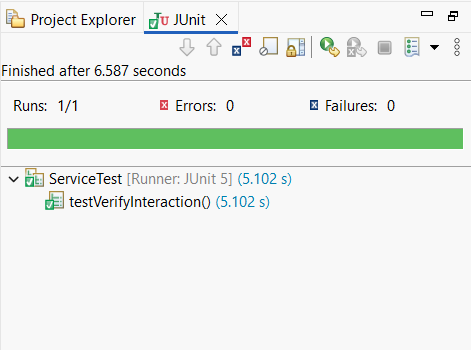
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Output:**



**Explanation:**

• A tool called Mockito is used in Java testing to generate simulated copies of actual objects, particularly ones that rely on external systems like databases or APIs.   
• For this work, I used mock() to create a mock of ExternalApi and when(...).thenReturn(...) to specify what its method should return.   
• I developed a MyService class that makes use of ExternalApi, and during testing, it passed the mock version.   
• In this manner, the test is independent of the actual external API and concentrates just on the service logic.   
• Mocking and stubbing make it easier to safely test each component of the application independently.