**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing in Java with Mockito**

### **Introduction**

In software development, unit testing is a crucial practice to ensure the reliability of code. When testing classes that depend on external systems, such as APIs or databases, directly invoking those systems can make tests slow, brittle, and difficult to maintain. To address this, we use mocking frameworks like Mockito, which allow us to simulate the behavior of external dependencies.

This document provides a detailed walkthrough of a simple unit testing exercise using **Mockito** and **JUnit 5**, focusing on mocking and stubbing an external API.

### **Objective**

* Create a mock object for an external API.
* Stub the method of the API to return a predefined value.
* Write a unit test that verifies the service logic using the mocked API.

### **Technologies Used**

* **Java** (JDK 8+)
* **JUnit 5** for testing framework
* **Mockito** for mocking and stubbing dependencies

### **Step-by-Step Implementation**

#### **Step 1: Define the External API Interface**

package com.example;  
  
public interface ExternalApi {  
 String getData();  
}

This interface simulates an external system that our service depends on.

#### **Step 2: Implement the Service Class**

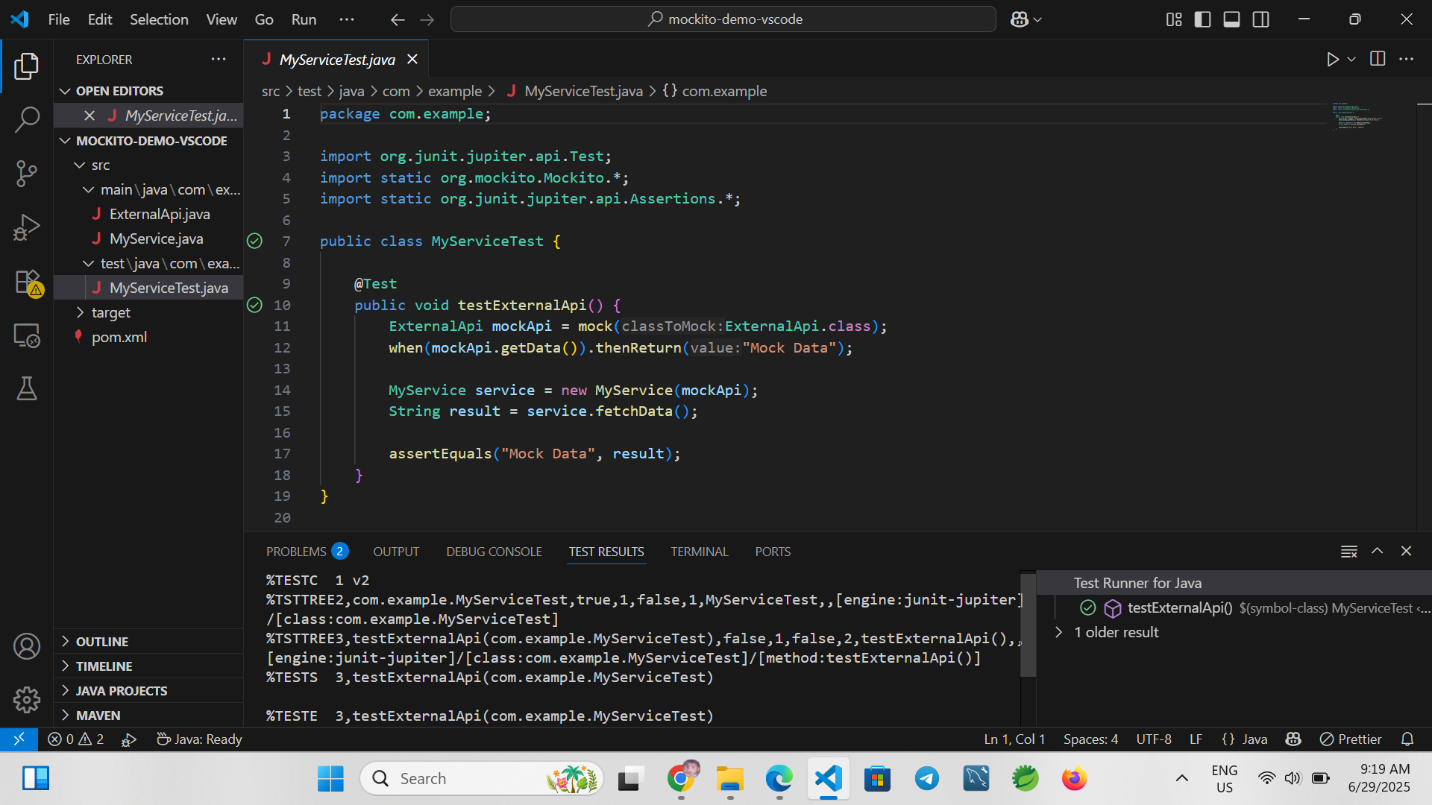
package com.example;  
  
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

This class uses ExternalApi to fetch data. It will be the class under test.

#### **Step 3: Write the Test Case Using Mockito**

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() {  
 // Step 1: Create mock object  
 ExternalApi mockApi = Mockito.mock(ExternalApi.class);  
  
 // Step 2: Stub method to return mock data  
 when(mockApi.getData()).thenReturn("Mock Data");  
  
 // Step 3: Inject mock into service and invoke method  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 // Step 4: Verify result  
 assertEquals("Mock Data", result);  
 }  
}

### **Expected Output**



### **Conclusion**

Mocking and stubbing are foundational techniques in unit testing, especially when dealing with external dependencies. By using Mockito in conjunction with JUnit, developers can write fast, isolated, and reliable tests. This exercise demonstrated how to mock an external API, stub its method, and verify the behavior of a service class in a clean and maintainable way.

**Exercise 2: Verifying Interactions Using Mockito**

**Introduction**

In unit testing, it is often important not just to check the return values of methods, but also to confirm whether certain methods were called—and how they were called. This is especially critical when your code interacts with external services or triggers important side effects.

Mockito provides a simple and powerful way to verify interactions with mock objects. This document explains how to use Mockito to ensure that a method is invoked with the expected arguments.

**Objective**

* Create a mock object.
* Call the method with specific arguments.
* Verify that the interaction with the mock occurred as expected.

**Technologies Used**

* **Java** (JDK 8+)
* **JUnit 5** for the testing framework
* **Mockito** for mocking and verification

**Step-by-Step Implementation**

***Step 1: Define the External API Interface***

**package com**.**example**;  
  
**public** **interface** ExternalApi {  
 String getData();  
}

This is a simple interface that will be mocked during testing.

***Step 2: Implement the Service Class***

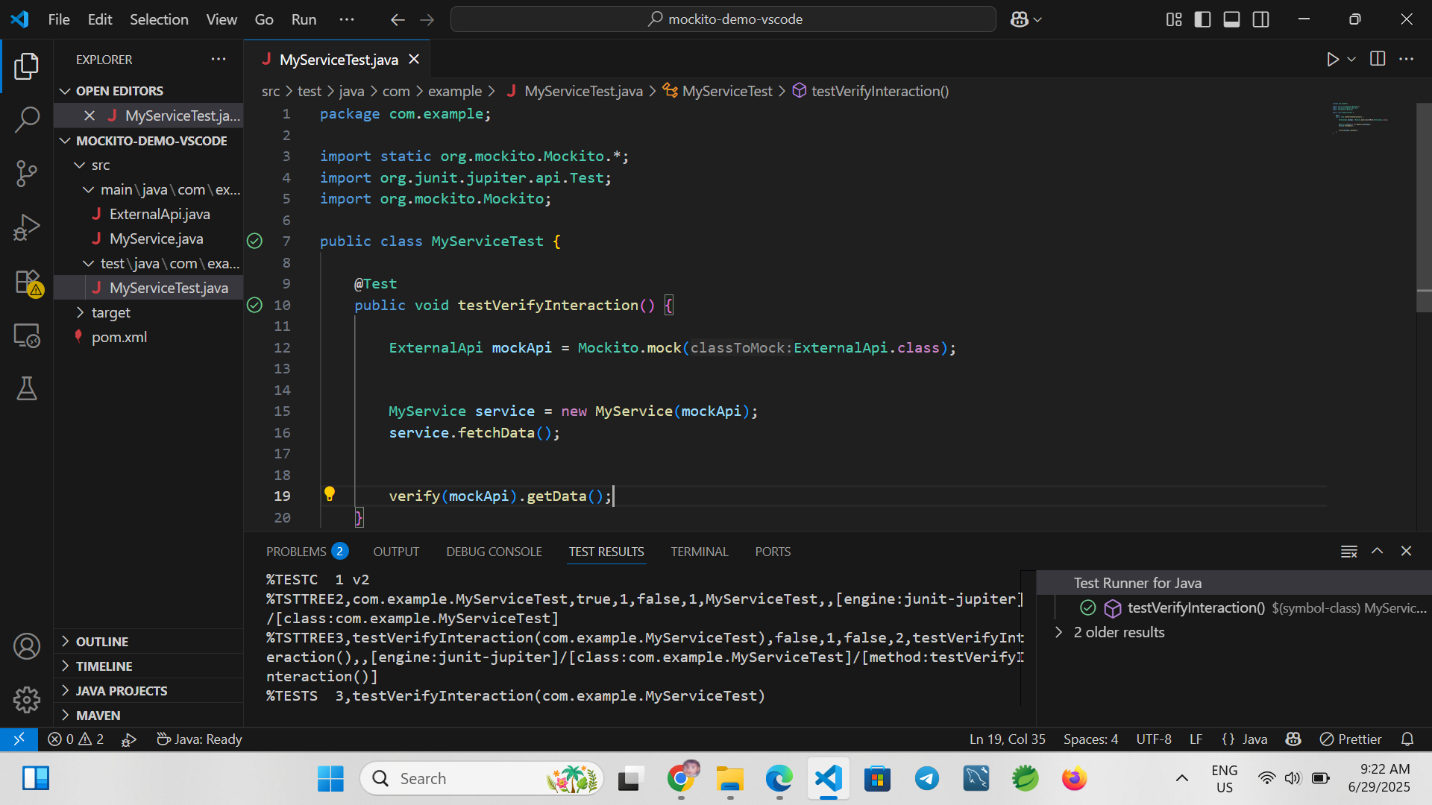
**package com**.**example**;  
  
**public** **class** MyService {  
 **private** ExternalApi api;  
  
 **public** MyService(ExternalApi api) {  
 **this**.api = api;  
 }  
  
 **public** String fetchData() {  
 **return** api.getData();  
 }  
}

The service class uses the ExternalApi interface. In the test, we’ll check if getData() is actually called.

***Step 3: Write the Test Case Using Mockito***

**package com**.**example**;  
  
**import** **static org**.**mockito**.**Mockito**.\*;  
**import** **org**.**junit**.**jupiter**.**api**.**Test**;  
**import** **org**.**mockito**.**Mockito**;  
  
**public** **class** MyServiceTest {  
  
 @Test  
 **public** void testVerifyInteraction() {  
 *// Step 1: Create mock object*  
 ExternalApi mockApi = Mockito.mock(ExternalApi.class);  
  
 *// Step 2: Use the mock in service*  
 MyService service = **new** MyService(mockApi);  
 service.fetchData();  
  
 *// Step 3: Verify interaction with the mock*  
 verify(mockApi).getData();  
 }  
}

**Expected Output**



**Conclusion**

Verifying interactions is a powerful feature in Mockito, useful when testing methods that call other methods but don’t return values or when checking the control flow. This exercise demonstrated how to use verify() to confirm that a mock method was invoked during the execution of a test case.