**Week2\_Mocking and Stubbing**

## 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:**

**Create a mock object for the external API.**

*<?*xml version="1.0" encoding="UTF-8"*?>*<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mocking\_Stubbing</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 *<!-- JUnit 5 (for @Test) -->* <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.8.2</version>  
 <scope>test</scope>  
 </dependency>  
  
 *<!-- Mockito (for mocking and stubbing) -->* <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>4.5.1</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**Stub the methods to return predefined values.**

-- **Create the Interface : ExternalApi**

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

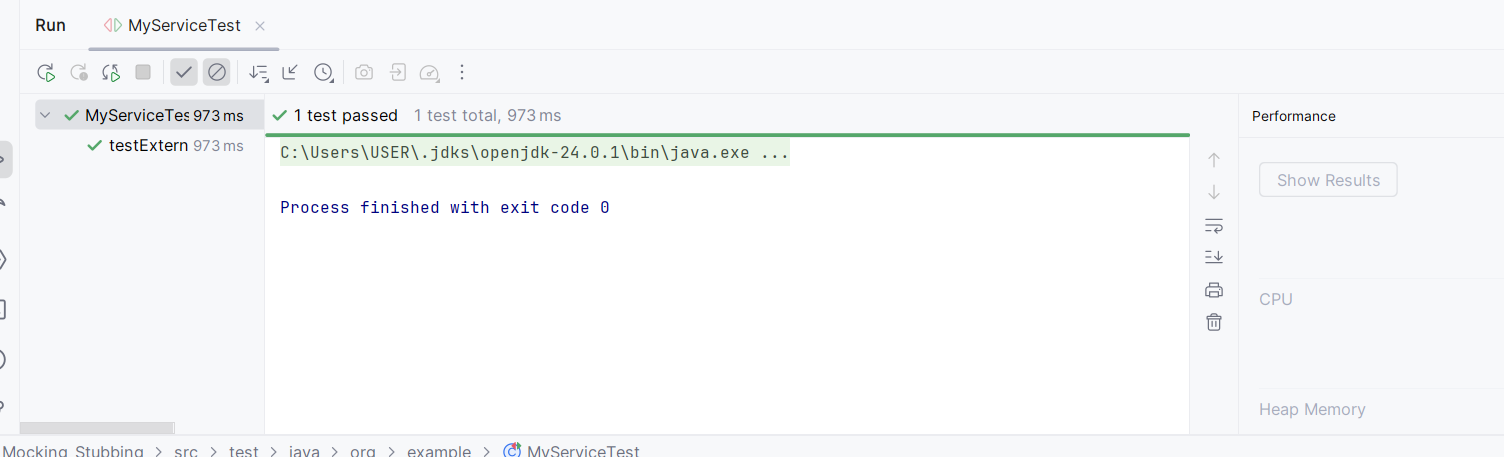
-- **Create the Service Class : MyService**

package org.example;  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String fetchData() {  
 *// This method depends on the external API* return this.externalApi.getData();  
 }  
}

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

package org.example;  
*// Import necessary classes from JUnit 5 and Mockito*import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() { *// We create a fake "stunt double" for the ExternalApi.* ExternalApi mockApi = *mock*(ExternalApi.class);  
 *// We program our fake object: "WHEN getData() is called, THEN RETURN 'Mock Data'".  
 when*(mockApi.getData()).thenReturn("Mock Data");  
 *// Create the service we are testing, but give it our FAKE API object.* MyService service = new MyService(mockApi);  
  
 *// Now, call the method we want to test.* String result = service.fetchData();  
  
 *// Finally, assert that our service returned the fake data we told the mock to produce.  
 assertEquals*("Mock Data", result);  
 }  
}

Output



**Week2\_Verifying Interactions**

## Exercise 2: Verifying Interactions

**Scenario:**

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

**Steps:**

**Create a mock object.**

*<?*xml version="1.0" encoding="UTF-8"*?>*<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 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mocking\_Stubbing</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 *<!-- JUnit 5 (for @Test) -->* <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.8.2</version>  
 <scope>test</scope>  
 </dependency>  
  
 *<!-- Mockito (for mocking and stubbing) -->* <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>4.5.1</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**Call the method with specific arguments.**

-- **Create the Interface : ExternalApi**

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

-- **Create the Service Class : MyService**

package org.example;  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String fetchData() {  
 *// This method depends on the external API* return this.externalApi.getData();  
 }  
}

**Verify the interaction.**

package org.example;  
  
*// Import necessary classes from JUnit 5 and Mockito*import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
   
 *// This new method goes inside your MyServiceTest class* @Test  
 public void testVerifyInteraction() {  
 *// ARRANGE: Create a mock object for the dependency.* ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
  
 *// ACT: Call the method on the service we are testing.* service.fetchData();  
  
 *// VERIFY: Check that an interaction with the mock occurred.  
 // This line asks Mockito: "Verify that the `getData()` method was  
 // called on the mockApi object exactly one time."  
 verify*(mockApi).getData();  
 }  
  
 @Test  
 public void testExternalApi() {  
 *// STEP 1: Create a mock object for the external API.  
 // We create a fake "stunt double" for the ExternalApi.* ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 *// STEP 2: Stub the methods to return predefined values.  
 // We program our fake object: "WHEN getData() is called, THEN RETURN 'Mock Data'".  
 when*(mockApi.getData()).thenReturn("Mock Data");  
  
 *// STEP 3: Write a test case that uses the mock object.  
 // Create the service we are testing, but give it our FAKE API object.* MyService service = new MyService(mockApi);  
  
 *// Now, call the method we want to test.* String result = service.fetchData();  
  
 *// Finally, assert that our service returned the fake data we told the mock to produce.  
 assertEquals*("Mock Data", result);  
 }  
}

Output

