**Mockito Hands-On 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);

}

}

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.testing;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock of ExternalApi

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

// Step 2: Stub getData() to return "Mock Data"

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

// Step 3: Inject mock into service

MyService service = new MyService(mockApi);

// Step 4: Call fetchData and verify

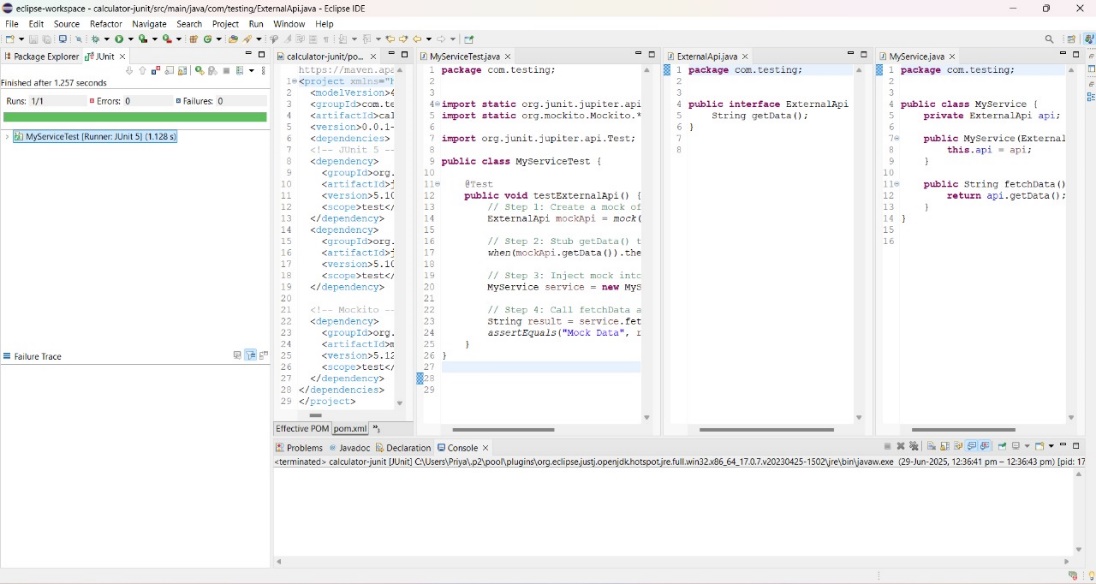
String result = service.fetchData();

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

}

}

**Output:**



**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(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Program:**

<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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

void getData();

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

**MyServiceTest.java**

package com.testing;

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); // Step 1: create mock

MyService service = new MyService(mockApi); // Inject mock

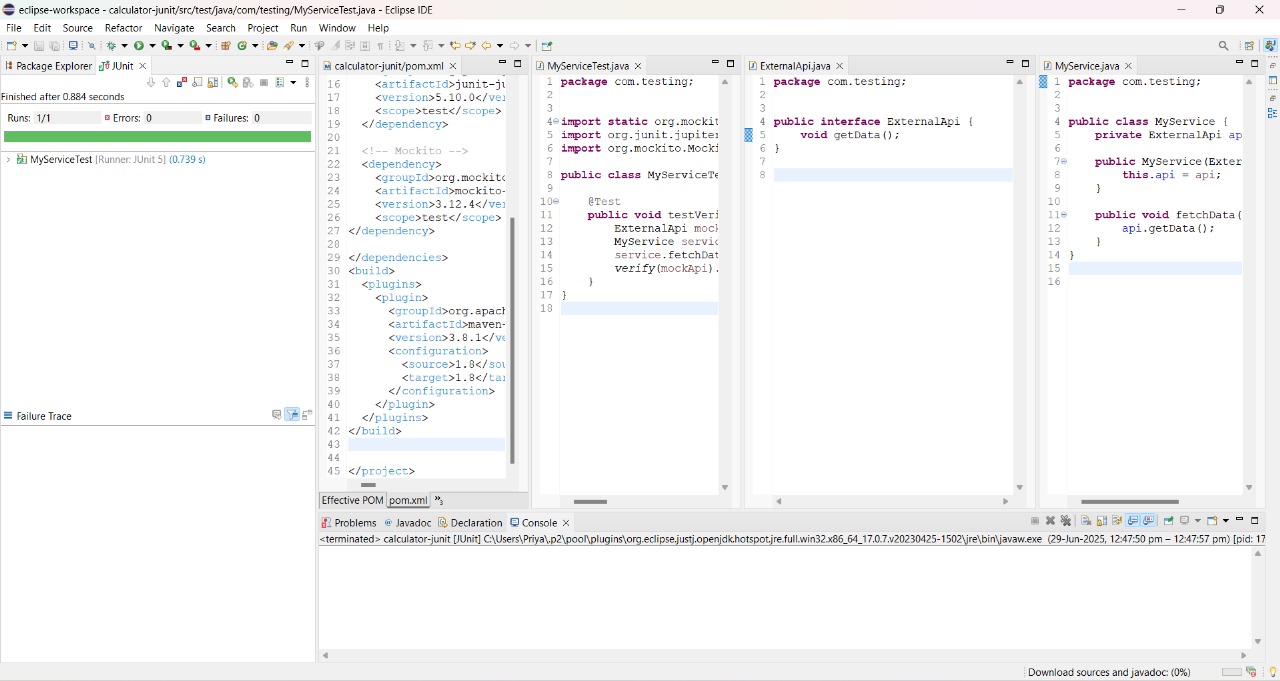
service.fetchData(); // Step 2: call method

*verify*(mockApi).getData(); // Step 3: verify interaction

}

}

**Output:**



**Exercise 3: Argument Matching**

**Scenario:**

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

**Steps:**

1. Create a mock object.

2. Call the method with specific arguments.

3. Use argument matchers to verify the interaction.

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

void sendData(String data);

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void process() {

api.sendData("HelloWorld"); // Hardcoded value to match in test

}

}

**MyServiceTest.java**

package com.testing;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testArgumentMatching() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class); // Step 1

MyService service = new MyService(mockApi);

service.process(); // Step 2

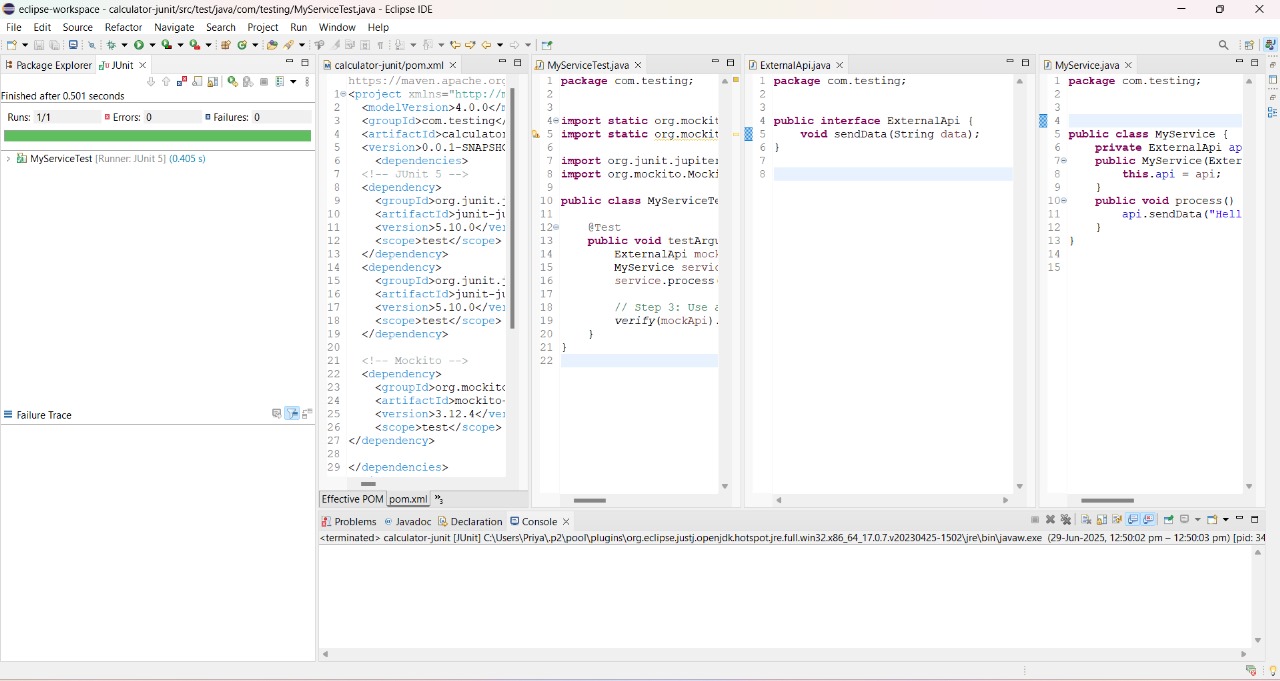
// Step 3: Use argument matcher to verify the argument passed

*verify*(mockApi).sendData(*eq*("HelloWorld")); // Use eq() to match the specific argument

}

}

**Output:**



**Exercise 4: Handling Void Methods**

**Scenario:**

You need to test a void method that performs some action.

**Steps:**

1. Create a mock object.

2. Stub the void method.

3. Verify the interaction.

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

void logMessage(String message);

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void doSomething() {

api.logMessage("Action completed");

}

}

**MyServiceTest.java**

package com.testing;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVoidMethodInteraction() {

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

// Step 2: Stub the void method (optional, only if it throws or needs custom behavior)

*doNothing*().when(mockApi).logMessage(*anyString*());

MyService service = new MyService(mockApi);

service.doSomething();

// Step 3: Verify the void method was called with correct argument

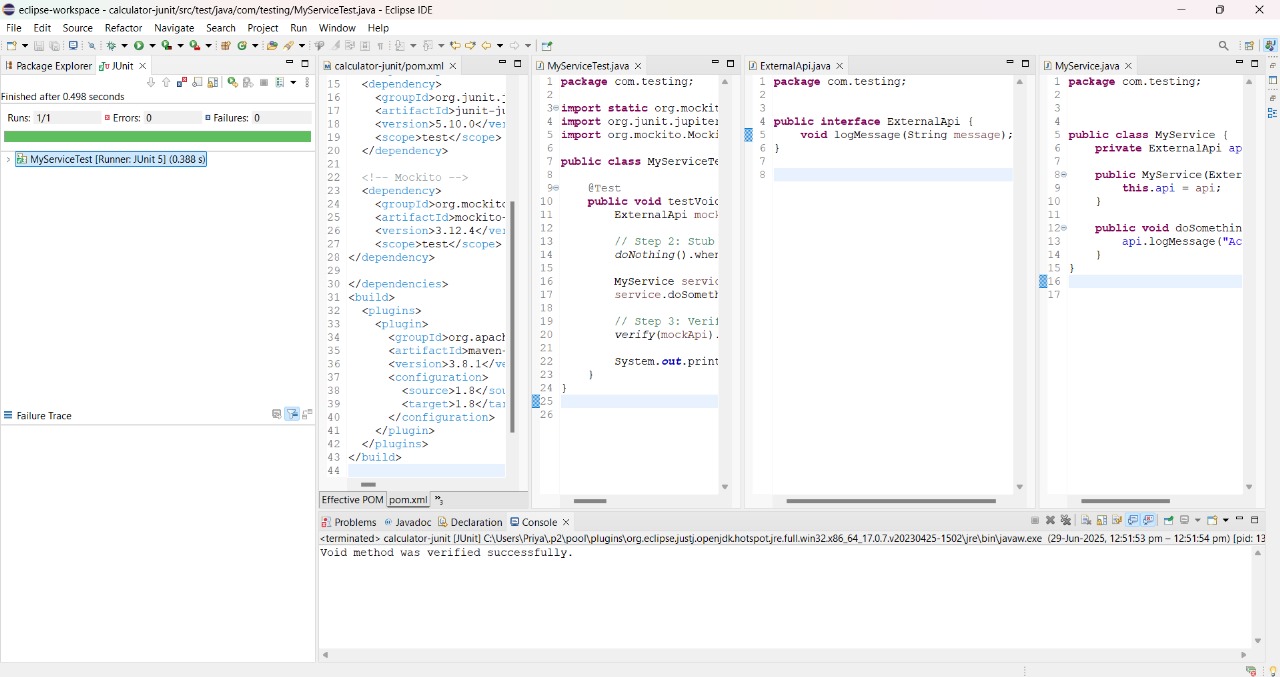
*verify*(mockApi).logMessage("Action completed");

System.*out*.println("Void method was verified successfully.");

}

}

**Output:**



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

**Scenario:**

You need to test a service that depends on an external API with multiple return values.

**Steps:**

1. Create a mock object for the external API.

2. Stub the methods to return different values on consecutive calls.

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

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

String getStatus();

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String checkSystem() {

return api.getStatus();

}

}

**MyServiceTest.java**

package com.testing;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

@Test

public void testMultipleReturns() {

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

// Step 2: Stub getStatus() to return different values each call

*when*(mockApi.getStatus())

.thenReturn("LOADING")

.thenReturn("INITIALIZING")

.thenReturn("READY");

MyService service = new MyService(mockApi);

// Step 3: Call the method multiple times and assert return values

*assertEquals*("LOADING", service.checkSystem());

*assertEquals*("INITIALIZING", service.checkSystem());

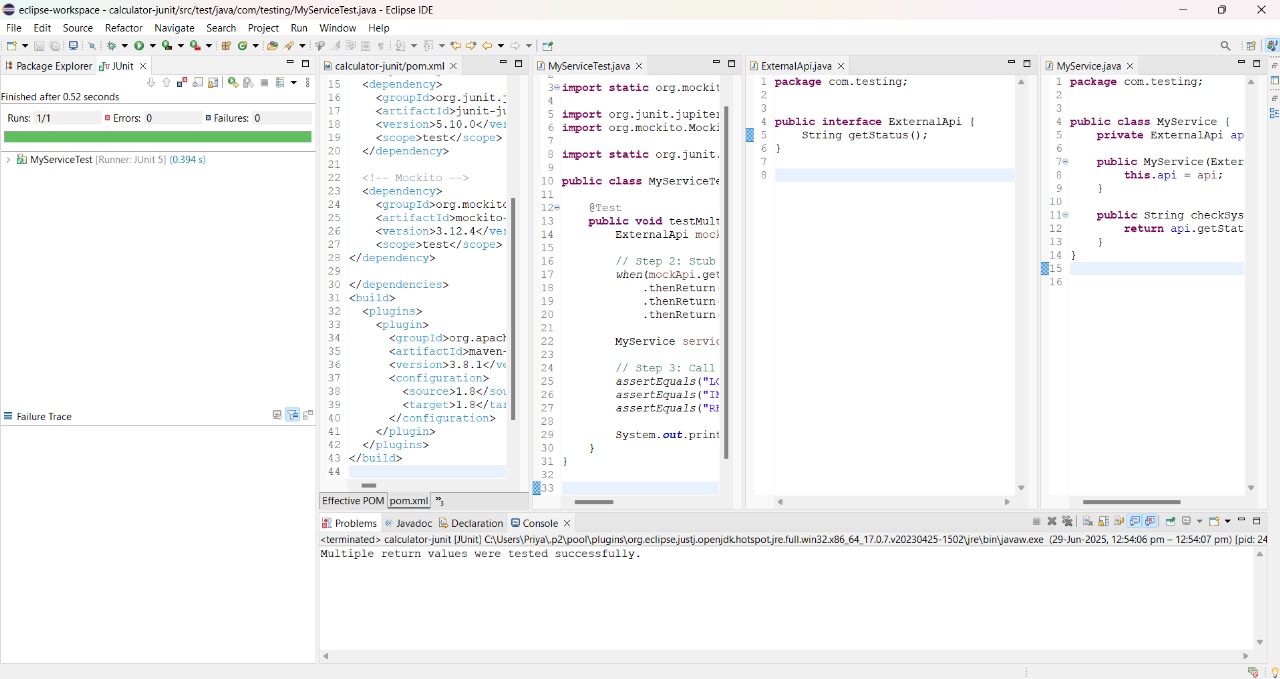
*assertEquals*("READY", service.checkSystem());

System.*out*.println("Multiple return values were tested successfully.");

}

}

**Output:**



**Exercise 6: Verifying Interaction Order**

**Scenario:**

You need to ensure that methods are called in a specific order.

**Steps:**

1. Create a mock object.

2. Call the methods in a specific order.

3. Verify the interaction order.

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

void connect();

void fetchData();

void disconnect();

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void execute() {

api.connect();

api.fetchData();

api.disconnect();

}

}

**MyServiceTest.java**

package com.testing;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testMethodCallOrder() {

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

MyService service = new MyService(mockApi);

service.execute();

// Step 3: Verify the order of interactions

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).connect();

inOrder.verify(mockApi).fetchData();

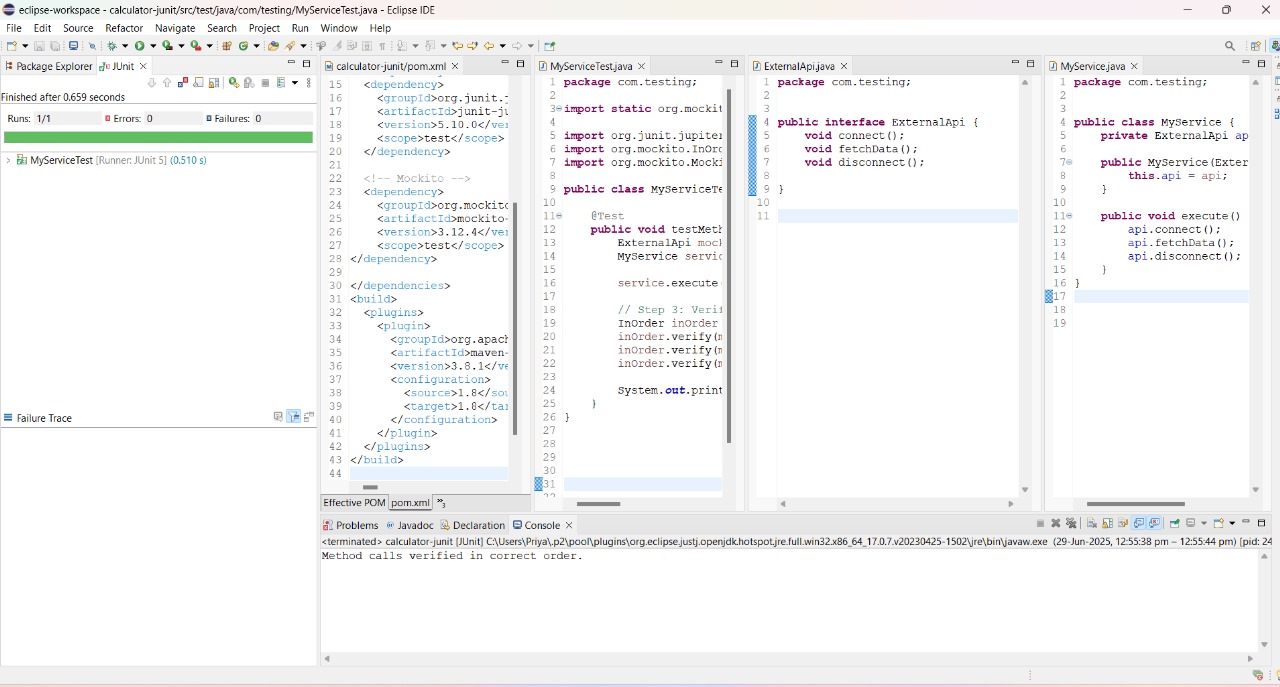
inOrder.verify(mockApi).disconnect();

System.*out*.println("Method calls verified in correct order.");

}

}

**Output:**



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

**Scenario:**

You need to test a void method that throws an exception.

**Steps:**

1. Create a mock object.

2. Stub the void method to throw an exception.

3. Verify the interaction.

**Program:**

**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.testing</groupId>

<artifactId>calculator-junit</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>3.12.4</version> <!-- OK for Java 8 -->

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.testing;

public interface ExternalApi {

void deleteRecord(String recordId) throws RuntimeException;

}

**MyService.java**

package com.testing;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void performDelete(String id) {

try {

api.deleteRecord(id);

System.*out*.println("Delete successful");

} catch (RuntimeException e) {

System.*out*.println("Exception caught: " + e.getMessage());

}

}

}

**MyServiceTest.java**

package com.testing;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVoidMethodThrowsException() {

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

// Step 2: Stub the void method to throw an exception

*doThrow*(new RuntimeException("Delete failed"))

.when(mockApi)

.deleteRecord("123");

MyService service = new MyService(mockApi);

service.performDelete("123");

// Step 3: Verify that the method was called despite the exception

*verify*(mockApi).deleteRecord("123");

System.*out*.println("Exception handling and interaction verified.");

}

}

**Output:**

