**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>mockitodemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>mockitodemo</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.5</version>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package app;

public interface ExternalApi {

String getData();

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // Uses external API

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

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() {

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

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

MyServices service = new MyServices(mockApi);

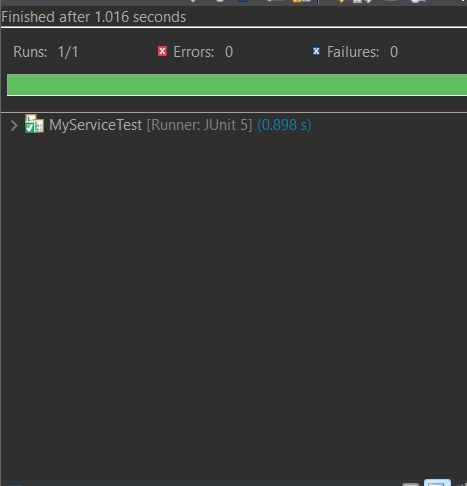
String result = service.fetchData();

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

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

**ExternalApi.java**

package app;

public interface ExternalApi {

String getData();

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // Uses external API

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

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

MyServices service = new MyServices(mockApi);

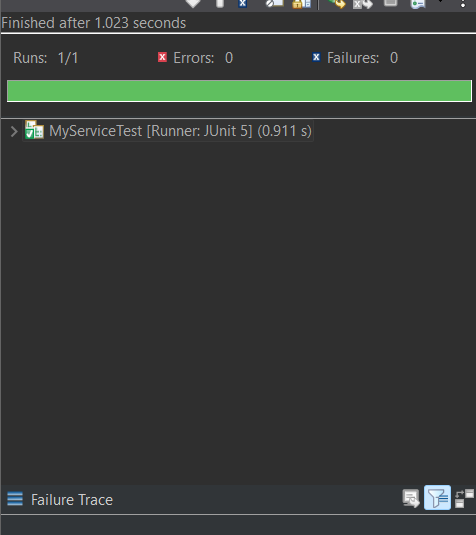
service.fetchData();

*verify*(mockApi).getData(); // checks if getData() was called

}

}

**Output:**

****

**Exercise 3: Argument Matching**

**ExternalApi.java**

package app;

public interface ExternalApi {

String getData(); // existing

void sendData(String payload); // new method

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

public void processAndSend(String input) {

// maybe some logic here, but we just call the API for now

api.sendData(input);

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyWithArgument() {

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

MyServices service = new MyServices(mockApi);

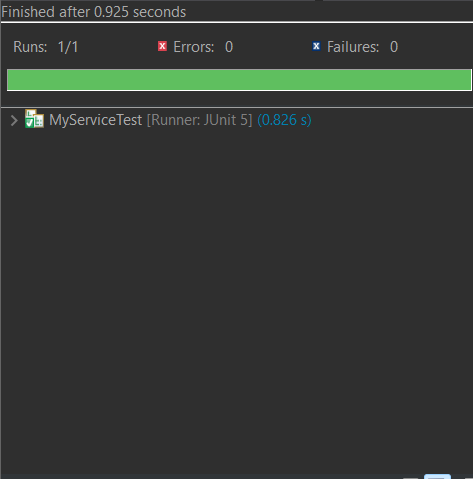
service.processAndSend("Hello World");

*verify*(mockApi).sendData("Hello World");

}

}

**Output:**

****

**Exercise 4: Handling Void Methods**

**ExternalApi.java**

package app;

public interface ExternalApi {

void log(String message); // <- void method

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public void performLogging() {

api.log("Action completed");

}

}

**MyServicesTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVoidMethodInteraction() {

// Step 1: Create mock

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

// Step 2: Optional - stub the void method (not always needed)

// doNothing().when(mockApi).log(anyString());

// Step 3: Use the mock

MyServices service = new MyServices(mockApi);

service.performLogging();

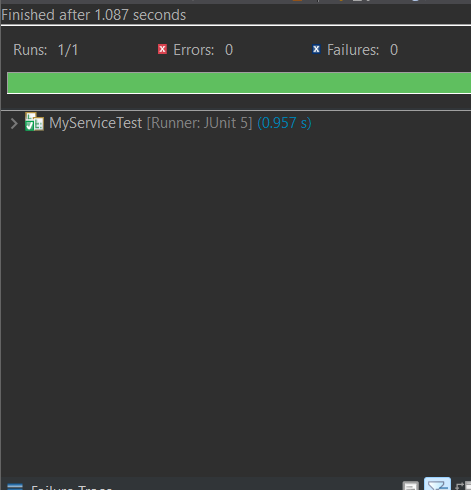
// Step 4: Verify the interaction

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

}

}

**Output:**

****

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

**ExternalApi.java**

package app;

public interface ExternalApi {

String getStatus(); // This method will return different values on each call

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public String[] checkStatusTwice() {

String first = api.getStatus();

String second = api.getStatus();

return new String[]{first, second};

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

*@Test*

public void testMultipleReturns() {

// Step 1: Create mock

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

// Step 2: Stub to return multiple values

*when*(mockApi.getStatus())

.thenReturn("Pending")

.thenReturn("Completed");

// Step 3: Use the mock in the service

MyServices service = new MyServices(mockApi);

String[] results = service.checkStatusTwice();

// Step 4: Assert both return values

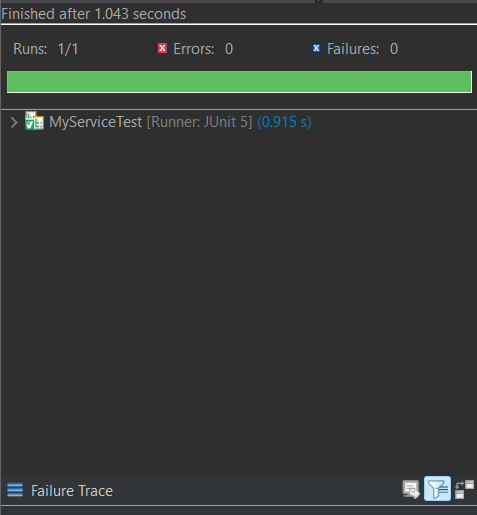
*assertEquals*("Pending", results[0]);

*assertEquals*("Completed", results[1]);

}

}

**Output:**



**Exercise 6: Verifying Interaction Order**

**ExternalApi.java**

package app;

public interface ExternalApi {

void connect();

void fetchData();

void disconnect();

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public void process() {

api.connect();

api.fetchData();

api.disconnect();

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import org.mockito.InOrder;

public class MyServiceTest {

*@Test*

public void testVerifyInteractionOrder() {

// Step 1: Create mock

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

// Step 2: Use the mock in the service

MyServices service = new MyServices(mockApi);

service.process();

// Step 3: Verify interaction order

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).connect();

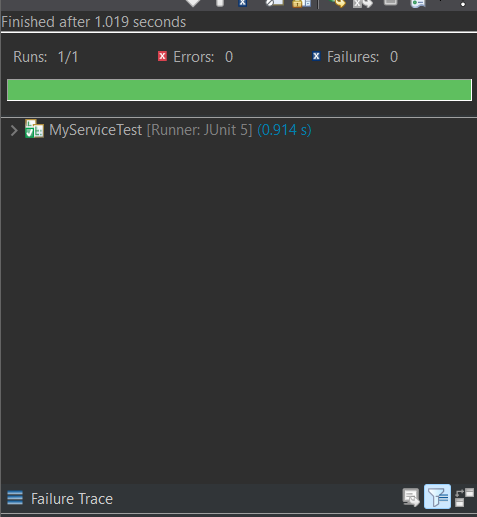
inOrder.verify(mockApi).fetchData();

inOrder.verify(mockApi).disconnect();

}

}

**Output:**

****

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

**ExternalApi.java**

package app;

public interface ExternalApi {

void deleteData();

}

**MyServices.java**

package app;

public class MyServices {

private final ExternalApi api;

public MyServices(ExternalApi api) {

this.api = api;

}

public void delete() {

api.deleteData(); // Might throw exception

}

}

**MyServiceTest.java**

package tests;

import app.ExternalApi;

import app.MyServices;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

*@Test*

public void testVoidMethodThrowsException() {

// 1. Create mock

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

// 2. Stub the void method to throw an exception

*doThrow*(new RuntimeException("Delete failed")).when(mockApi).deleteData();

// 3. Use the mock in the service

MyServices service = new MyServices(mockApi);

// 4. Verify exception is thrown

*assertThrows*(RuntimeException.class, service::delete);

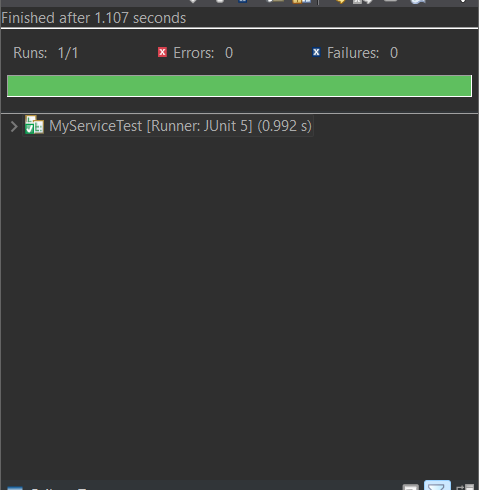
// 5. Verify interaction

*verify*(mockApi).deleteData();

}

}

**Output:**

****