**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);

}

}

**Code:**

**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.example</groupId>

<artifactId>MockitoMockingExample</artifactId>

<version>1.0</version>

<name>MockitoMockingExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example.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() {

// Step 1: Create a mock object of ExternalApi

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

// Step 2: Stub the method to return mock data

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

// Step 3: Inject the mock into the service and call the method

MyService service = new MyService(mockApi);

String result = service.fetchData();

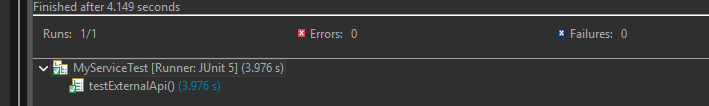
// Step 4: Verify result

*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();

}

}

**Code:**

**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.example</groupId>

<artifactId>MockitoExample</artifactId>

<version>1.0</version>

<name>MockitoMockingExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

// Step 1: Create mock

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

// Step 2: Inject into service and call fetchData()

MyService service = new MyService(mockApi);

service.fetchData();

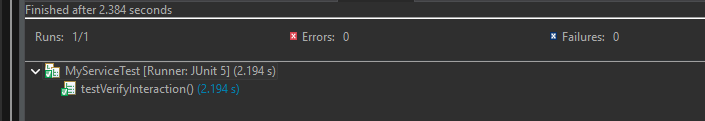
// Step 3: Verify that mockApi.getData() was called exactly once

*verify*(mockApi).getData();

}

}

**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.

**Code:**

**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.example</groupId>

<artifactId>MockitoArgumentMatcherExample</artifactId>

<version>1.0</version>

<name>MockitoArgumentMatcherExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

void sendMessage(String message);

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

public void deliverMessage(String msg) {

api.sendMessage(msg);

}

}

**MyServiceArgumentTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class MyServiceArgumentTest {

*@Test*

public void testSendMessageWithExactArgument() {

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

MyService service = new MyService(mockApi);

service.deliverMessage("Hello Mockito");

// Verify it was called with exact value

*verify*(mockApi).sendMessage("Hello Mockito");

}

*@Test*

public void testSendMessageWithAnyArgument() {

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

MyService service = new MyService(mockApi);

service.deliverMessage("Hello Matchers");

// Use argument matcher: any string

*verify*(mockApi).sendMessage(*anyString*());

}

*@Test*

public void testSendMessageWithMatcherAndCustomCheck() {

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

MyService service = new MyService(mockApi);

service.deliverMessage("Important: Process now");

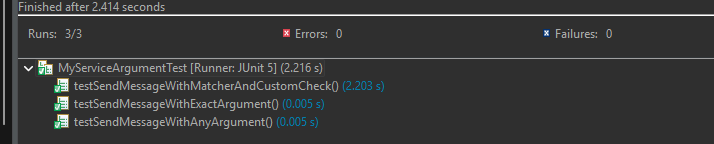
// Use custom matcher: string starts with "Important"

*verify*(mockApi).sendMessage(*argThat*(msg -> msg.startsWith("Important")));

}

}

**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.

**Code:**

**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.example</groupId>

<artifactId>MockitoVoidMethodExample</artifactId>

<version>1.0</version>

<name>MockitoVoidMethodExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getData();

void sendMessage(String message); // void method to test

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void deliverMessage(String message) {

api.sendMessage(message);

}

}

**MyServiceVoidTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.*anyString*;

public class MyServiceVoidTest {

*@Test*

public void testVoidMethodInteraction() {

// Step 1: Create mock

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

// Step 2: (Optional) Stub void method (not always needed)

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

// Step 3: Call method and verify interaction

MyService service = new MyService(mockApi);

service.deliverMessage("Hello Void");

// Step 4: Verify the void method was called

*verify*(mockApi).sendMessage("Hello Void");

}

*@Test*

public void testVoidMethodWithStubbedBehavior() {

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

// Stub the void method to throw an exception

*doThrow*(new RuntimeException("Failed to send")).when(mockApi).sendMessage("fail");

MyService service = new MyService(mockApi);

// Verify exception is thrown when called with "fail"

try {

service.deliverMessage("fail");

} catch (RuntimeException ex) {

System.***out***.println("Caught expected exception: " + ex.getMessage());

}

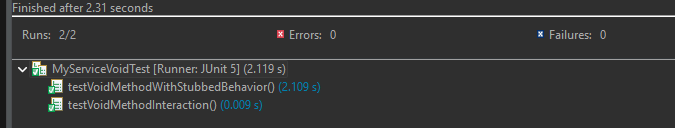
// Verify it was still called

*verify*(mockApi).sendMessage("fail");

}

}

**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.

**Code:**

**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.example</groupId>

<artifactId>MockitoMultipleReturnsExample</artifactId>

<version>1.0</version>

<name>MockitoMultipleReturnsExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

String getStatus();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String[] pollStatusThreeTimes() {

return new String[] {

api.getStatus(),

api.getStatus(),

api.getStatus()

};

}

}

**MyServiceMultiReturnTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceMultiReturnTest {

*@Test*

public void testMultipleReturnsFromMock() {

// Step 1: Create mock

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

// Step 2: Stub to return different values on each call

*when*(mockApi.getStatus())

.thenReturn("PENDING")

.thenReturn("PROCESSING")

.thenReturn("DONE");

// Step 3: Inject and test

MyService service = new MyService(mockApi);

String[] actualStatuses = service.pollStatusThreeTimes();

// Step 4: Assert result

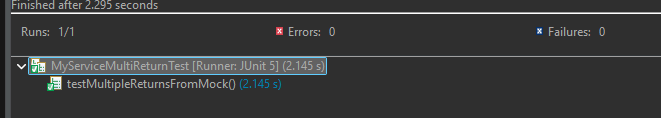
String[] expected = {"PENDING", "PROCESSING", "DONE"};

*assertArrayEquals*(expected, actualStatuses);

}

}

**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.

**Code:**

**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.example</groupId>

<artifactId>MockitoMultipleReturnsExample</artifactId>

<version>1.0</version>

<name>MockitoMultipleReturnsExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

void authenticate();

void fetchData();

void logout();

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void runWorkflow() {

api.authenticate();

api.fetchData();

api.logout();

}

}

**MyServiceInteractionOrderTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import static org.mockito.Mockito.\*;

public class MyServiceInteractionOrderTest {

*@Test*

public void testMethodCallOrder() {

// Step 1: Create mock

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

// Step 2: Inject mock and run workflow

MyService service = new MyService(mockApi);

service.runWorkflow();

// Step 3: Verify order

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).authenticate();

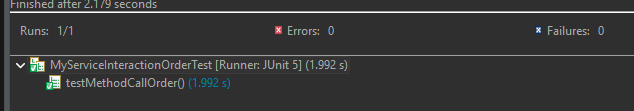
inOrder.verify(mockApi).fetchData();

inOrder.verify(mockApi).logout();

}

}

**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.

**Code:**

**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.example</groupId>

<artifactId>MockitoVoidExceptionExample</artifactId>

<version>1.0</version>

<name>MockitoVoidExceptionExample</name>

<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>5.12.0</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**ExternalApi.java**

package com.example.mockito;

public interface ExternalApi {

void sendMessage(String message); // may throw an exception

}

**MyService.java**

package com.example.mockito;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void deliverMessage(String message) {

api.sendMessage(message);

}

}

**MyServiceVoidExceptionTest.java**

package com.example.mockito;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceVoidExceptionTest {

*@Test*

public void testVoidMethodThrowsException() {

// Step 1: Create mock

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

// Step 2: Stub void method to throw exception when called with "fail"

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

.when(mockApi)

.sendMessage("fail");

// Step 3: Inject and call

MyService service = new MyService(mockApi);

// Step 4: Assert exception is thrown

*assertThrows*(RuntimeException.class, () -> service.deliverMessage("fail"));

// Step 5: Verify the method was called

*verify*(mockApi).sendMessage("fail");

}

}

**Output:**

