**PAKHI SHARMA**

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);

}

}

SOLUTION

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

class ExternalApi {

public String getData() { return "Real Data"; }

}

class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) { this.api = api; }

public String fetchData() { return api.getData(); }

}

public class MyServiceTest1 {

@Test

void testExternalApiMock() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

# 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();

}

}

# 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.

class Calculator {

public int add(int a, int b) { return a + b; }

}

public class CalculatorTest {

@Test

void testArgumentMatching() {

Calculator calc = mock(Calculator.class);

when(calc.add(anyInt(), eq(5))).thenReturn(10);

int result = calc.add(3, 5);

assertEquals(10, result);

verify(calc).add(anyInt(), eq(5));

}

}

# 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.

class Notifier {

public void sendNotification(String msg) { System.out.println(msg); }

}

public class NotifierTest {

@Test

void testVoidMethod() {

Notifier notifier = mock(Notifier.class);

doNothing().when(notifier).sendNotification(anyString());

notifier.sendNotification("Hello");

verify(notifier).sendNotification("Hello");

}

}

# 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.

public class MyServiceTest5 {

@Test

void testMultipleReturns() {

ExternalApi mockApi = mock(ExternalApi.class);

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call");

assertEquals("First Call", mockApi.getData());

assertEquals("Second Call", mockApi.getData());

}

}

# 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.

class Processor {

public void stepOne() {}

public void stepTwo() {}

}

public class ProcessorTest {

@Test

void testOrderOfCalls() {

Processor processor = mock(Processor.class);

processor.stepOne();

processor.stepTwo();

InOrder inOrder = inOrder(processor);

inOrder.verify(processor).stepOne();

inOrder.verify(processor).stepTwo();

}

}

# 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.

public class NotifierExceptionTest {

@Test

void testVoidMethodThrowsException() {

Notifier notifier = mock(Notifier.class);

doThrow(new RuntimeException("Error"))

.when(notifier).sendNotification("Fail");

assertThrows(RuntimeException.class, () -> notifier.sendNotification("Fail"));

verify(notifier).sendNotification("Fail");

}

}