# Cognizant Digital Nurture 4.0 Deep Skilling

## Mockito 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 :**

You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

Step 1: Create a Mock Object for the External API Use Mockito to create a mock of the external dependency

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

**Step 2: Stub Methods to Return Predefined Values**

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

**Step 3: Write a Test Case that Uses the Mock Object**

import static org.mockito.Mockito.\*;

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

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);

}

}

Explanation:

Mockito.mock(Class) creates a fake instance of the given class.

when(...).thenReturn(...) stubs a method to return a specific value.

This is useful when testing services that rely on APIs, databases, or external services.

Conclusion: Mocking and stubbing are powerful techniques in unit testing to isolate the component under test. They help simulate behavior of complex or unavailable dependencies to ensure reliable and fast tests.

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();

}

}

**Solution :**

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.

**Solution :**

import static org.mockito.Mockito.*; import static org.mockito.ArgumentMatchers.*; import org.junit.jupiter.api.Test; import org.mockito.Mockito;

public class ArgumentMatcherTest {

@Test

public void testArgumentMatching() {

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

mockApi.sendMessage("Hello World");

verify(mockApi).sendMessage(eq("Hello World"));

}

}

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.

**Solution :**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class VoidMethodTest {

@Test

public void testVoidMethod() {

Logger mockLogger = Mockito.mock(Logger.class);

doNothing().when(mockLogger).logMessage(anyString());

mockLogger.logMessage("Test message");

verify(mockLogger).logMessage("Test message");

}

}

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.

**Solution :**

import static org.mockito.Mockito.; import org.junit.jupiter.api.Test; import org.mockito.Mockito; import static org.junit.jupiter.api.Assertions.;

public class MultipleReturnsTest {

@Test

public void testMultipleReturns() {

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

when(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

MyService service = new MyService(mockApi);

assertEquals("First Call", service.fetchData());

assertEquals("Second Call", service.fetchData());

assertEquals("Third Call", service.fetchData());

}

}

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.

**Solution :**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

import org.mockito.Mockito;

public class InteractionOrderTest {

@Test

public void testInteractionOrder() {

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

mockApi.initialize();

mockApi.getData();

mockApi.cleanup();

InOrder inOrder = Mockito.inOrder(mockApi);

inOrder.verify(mockApi).initialize();

inOrder.verify(mockApi).getData();

inOrder.verify(mockApi).cleanup();

}

}

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.

**Solution :**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class VoidMethodExceptionTest {

@Test

public void testVoidMethodThrowsException() {

Notifier mockNotifier = Mockito.mock(Notifier.class);

doThrow(new RuntimeException("Error occurred"))

.when(mockNotifier).sendNotification(anyString());

assertThrows(RuntimeException.class, () -> {

mockNotifier.sendNotification("Test message");

});

verify(mockNotifier).sendNotification("Test message");

}

}