**Cognizant Deep Nurture 4.0 Deep Skilling - Week 2**

**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); } }    
   
   
   
ExternalApi.java – The External Interface

package com.example;

public interface ExternalApi {

String getData();

} 

MyService.java – The Class to Test

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData(); // Relies on external system

}

}

MyServiceTest.java – Mockito Test

package com.example;

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

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Stub method

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

// Step 3: Inject mock into service

MyService service = new MyService(mockApi);

// Act

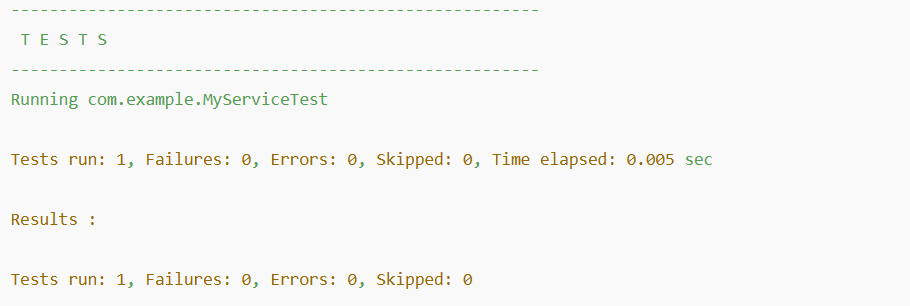
String result = service.fetchData();

// Assert

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();    
}    
}    
   
ExternalApi.java – The External Interface

package com.example;

public interface ExternalApi {

String getData();

}

MyService.java – The Service to Test

package com.example;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}   
 MyServiceTest.java – Verifying Interaction Using Mockito

package com.example;

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

MyService service = new MyService(mockApi);

// Step 3: Call method

service.fetchData();

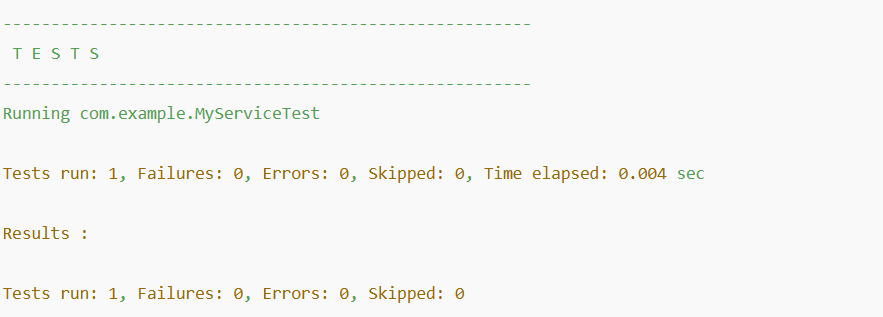
// Step 4: Verify that getData() was called

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.    
   
   
NotificationService.java – Interface to Be Mocked

package com.example;

public interface NotificationService {

void sendEmail(String recipient, String message);

}

UserService.java – Uses the External Dependency

package com.example;

public class UserService {

private final NotificationService notificationService;

public UserService(NotificationService notificationService) {

this.notificationService = notificationService;

}

public void registerUser(String email) {

// Some registration logic...

String welcomeMsg = "Welcome to our platform!";

notificationService.sendEmail(email, welcomeMsg);

}

}

UserServiceTest.java – Verifying Argument Matching

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

public class UserServiceTest {

@Test

public void testSendEmailCalledWithCorrectArguments() {

// Step 1: Create mock

NotificationService mockNotification = mock(NotificationService.class);

// Step 2: Inject mock

UserService userService = new UserService(mockNotification);

// Step 3: Call method with arguments

userService.registerUser("test@example.com");

// Step 4: Use argument matchers to verify

verify(mockNotification).sendEmail(eq("test@example.com"), contains("Welcome"));

}

}   
   
   
   
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.

LoggerService.java – The Void Method Interface

package com.example;

public interface LoggerService {

void log(String message);

}

AppService.java – The Class Using Logger

package com.example;

public class AppService {

private final LoggerService logger;

public AppService(LoggerService logger) {

this.logger = logger;

}

public void process(String input) {

if (input != null && !input.isEmpty()) {

logger.log("Processed input: " + input);

}

}

}

AppServiceTest.java – Testing the Void Method

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class AppServiceTest {

@Test

public void testLogMethodCalled() {

// Step 1: Create mock

LoggerService mockLogger = mock(LoggerService.class);

// Step 2: Stub (optional for void, unless using doNothing, doThrow etc.)

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

// Step 3: Use mock in service

AppService appService = new AppService(mockLogger);

appService.process("Hello!");

// Step 4: Verify void method interaction

verify(mockLogger).log("Processed input: Hello!");

}

}

OUTPUT:

