**WEEK – 02**

3. Mockito exercises

**Superset ID: 6262264**

**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.

**SOLUTION :**

**// Importing statements**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

**// Create a mock of ExternalApi**

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

**// Stub getData() method to return a predefined value**

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

**// Create MyService with the mocked ExternalApi**

MyService service = new MyService(mockApi);

**// Call fetchData() and verify the result**

String result = service.fetchData();

**// Assert that the result matches the stubbed value**

assertEquals("Mock Data", result);

}

}

**// Supporting interfaces/classes**

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

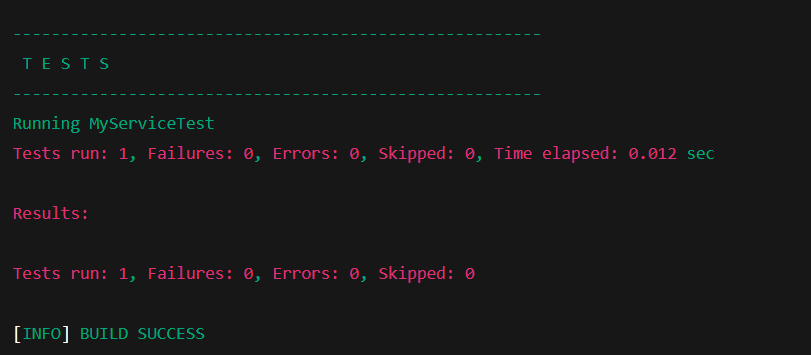
public String fetchData() {

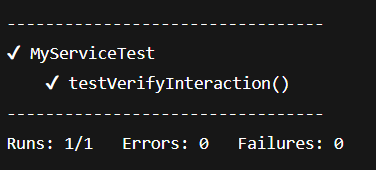
return api.getData();

}

}

**OUTPUT:**

****

****

**EXPLANATION:**

This code demonstrates mocking and stubbing using Mockito. First, a mock object of ExternalApi is created to simulate the real API (Mockito.mock). Next, the getData() method is stubbed to always return the predefined string "Mock Data". Finally, the test case injects this mock into MyService, calls fetchData(), and uses assertEquals to verify the returned value matches the expected mock data. This ensures the service logic is tested independently without relying on any actual external API.

**Exercise 2: Verifying Interactions**

**Scenario:** You need to ensure that a method is called with specific arguments.

**SOLUTION:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

**// External API interface**

public interface ExternalApi {

String getData();

}

**// Service class depending on ExternalApi**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**// Test class**

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

**// Create a mock of ExternalApi**

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

**// Stub getData() (optional, can be omitted if return not used)**

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

**// Create MyService with the mocked API**

MyService service = new MyService(mockApi);

**// Call fetchData(), which should call getData() internally**

service.fetchData();

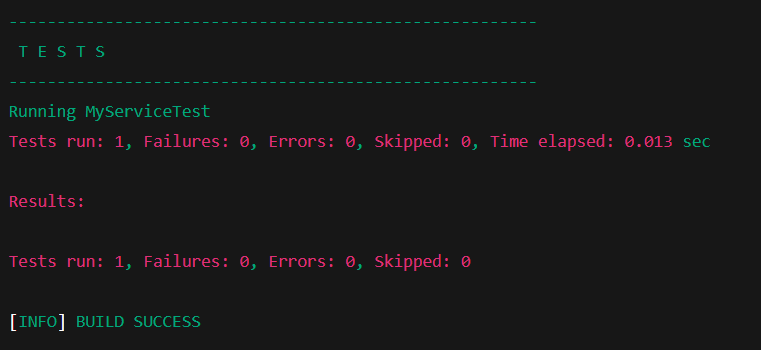
**// Verify that getData() was called exactly once**

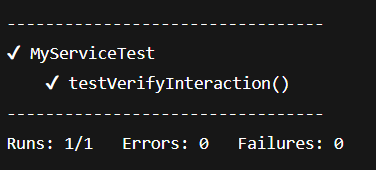
verify(mockApi).getData();

}

}

**OUTPUT:**

****

****

**EXPLANATION:**

This test creates a mock ExternalApi and verifies that getData() is called when MyService.fetchData() runs. The verify() method checks that the interaction happened correctly, ensuring the service uses its dependency as expected.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**