**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.

**Mocking:**

Mocking is the process of creating fake objects that simulate the behaviour of real ones.  
In unit testing, mocking allows you to isolate the class being tested from its dependencies.

**Stubbing:**

Stubbing is configuring a mock object to return specific data when a method is called.

**Step1: External API (to be mocked) – Interface:**

public interface ExternalApi

{

String getData();

}

**Step2: Service That Uses the API :**

**MyService.java:**

public class MyService

{

private ExternalApi api;

public MyService(ExternalApi api)

{

this.api = api;

}

public String fetchData()

{

return api.getData(); // Call external API

}

}

**Step3: Test Class Using Mockito**

**MyServiceTest.java:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create a mock for ExternalApi

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

// Step 2: Stub the getData() method

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

// Step 3: Inject mock into MyService and call method

MyService service = new MyService(mockApi);

String result = service.fetchData();

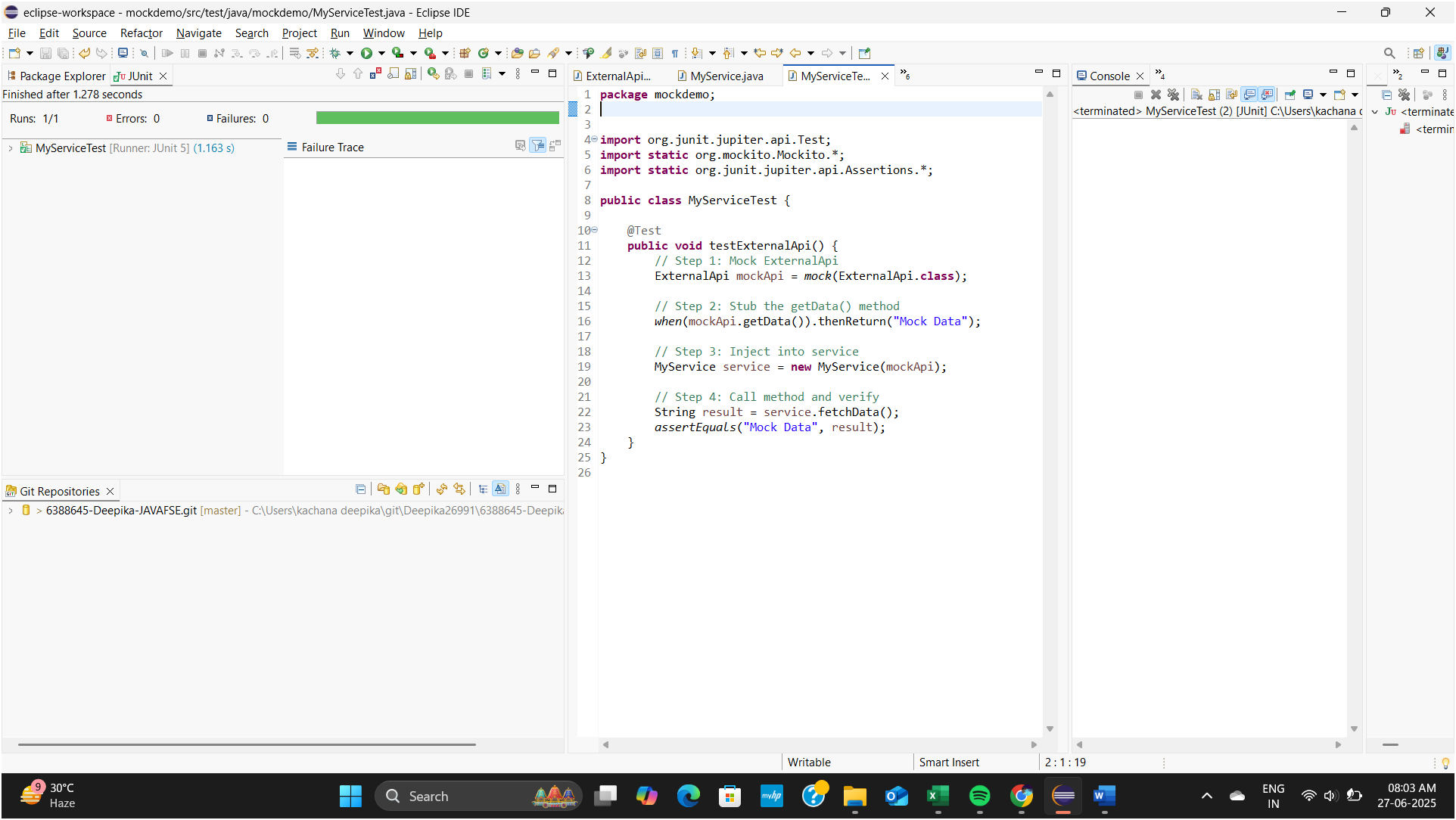
// Step 4: Assert the expected output

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.

**Step 1: WeatherApi.java — Interface (Mock Target)**

package weatherapp;

public interface WeatherApi

{

String getForecast();

}

**Step 2: WeatherService.java — Main Service**

package weatherapp;

public class WeatherService {

private WeatherApi api;

public WeatherService(WeatherApi api) {

this.api = api;

}

public String showForecast()

{

return api.getForecast();

}

}

**Step3:**

**WeatherServiceTest.java — Test with Interaction Verification**

package weatherapp;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class WeatherServiceTest

{

@Test

public void testVerifyForecastCalled()

{

// Step 1: Create a mock of WeatherApi

WeatherApi mockApi = mock(WeatherApi.class);

// Step 2: Inject the mock into WeatherService

WeatherService service = new WeatherService(mockApi);

// Step 3: Call method that internally uses the mock

service.showForecast();

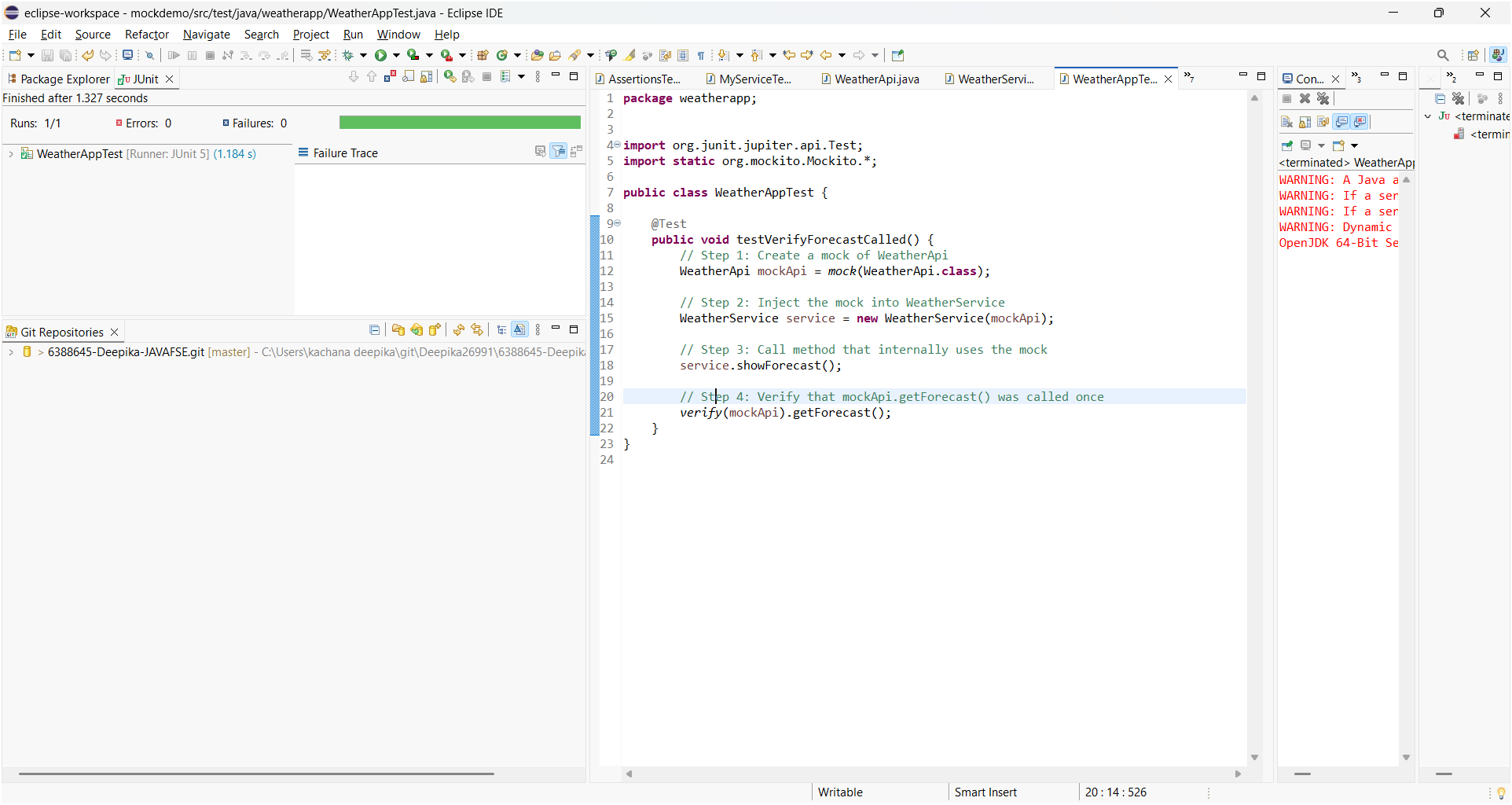
// Step 4: Verify that mockApi.getForecast() was called once

verify(mockApi).getForecast();

}

}

* This ensures your method interacts correctly with its dependencies.
* Useful for testing service-to-API communication, event triggers, or method call contracts.

**OUTPUT:**