**Student Name:** HARINEE N J

**Registration No:** 22CSR063

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**EXERCISE 1: MOCKING AND STUBBING**

**Introduction:**

This project demonstrates the use of mocking and stubbing in unit testing with Mockito, where a WeatherApi interface is mocked to simulate real weather data responses. The goal is to isolate the WeatherService logic and test its behavior without relying on an actual API.

**Objective:**

* **Mock External Dependencies:** Replace the real WeatherApi with a mock object to avoid external API calls during testing.
* **Stub Return Values:** Define specific return values for different city inputs using when(...).thenReturn(...) to simulate real-world data.
* **Validate Service Logic:** Ensure WeatherService formats and returns correct messages based on mocked API responses.

**Implementation Breakdown:**

**WeatherApi.java:**

public interface WeatherApi {

String getWeather(String city);

}

**WeatherService.java:**

public class WeatherService {

private WeatherApi api;

public WeatherService(WeatherApi api) {

this.api = api;

}

public String fetchTodayWeather(String city) {

return "Today's weather in " + city + ": " + api.getWeather(city);

}

}

**WeatherServiceTest.java:**

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class WeatherServiceTest {

@Test

public void testWeatherMultipleCities() {

WeatherApi mockApi = mock(WeatherApi.class);

when(mockApi.getWeather("Chennai")).thenReturn("Sunny, 32°C");

when(mockApi.getWeather("Delhi")).thenReturn("Cloudy, 28°C");

when(mockApi.getWeather("Mumbai")).thenReturn("Rainy, 29°C");

when(mockApi.getWeather("Kolkata")).thenReturn("Humid, 31°C");

when(mockApi.getWeather("Bengaluru")).thenReturn("Cool, 24°C");

WeatherService service = new WeatherService(mockApi);

assertEquals("Today's weather in Chennai: Sunny, 32°C", service.fetchTodayWeather("Chennai"));

assertEquals("Today's weather in Delhi: Cloudy, 28°C", service.fetchTodayWeather("Delhi"));

assertEquals("Today's weather in Mumbai: Rainy, 29°C", service.fetchTodayWeather("Mumbai"));

assertEquals("Today's weather in Kolkata: Humid, 31°C", service.fetchTodayWeather("Kolkata"));

assertEquals("Today's weather in Bengaluru: Cool, 24°C", service.fetchTodayWeather("Bengaluru"));

System.out.println(service.fetchTodayWeather("Chennai"));

System.out.println(service.fetchTodayWeather("Delhi"));

System.out.println(service.fetchTodayWeather("Mumbai"));

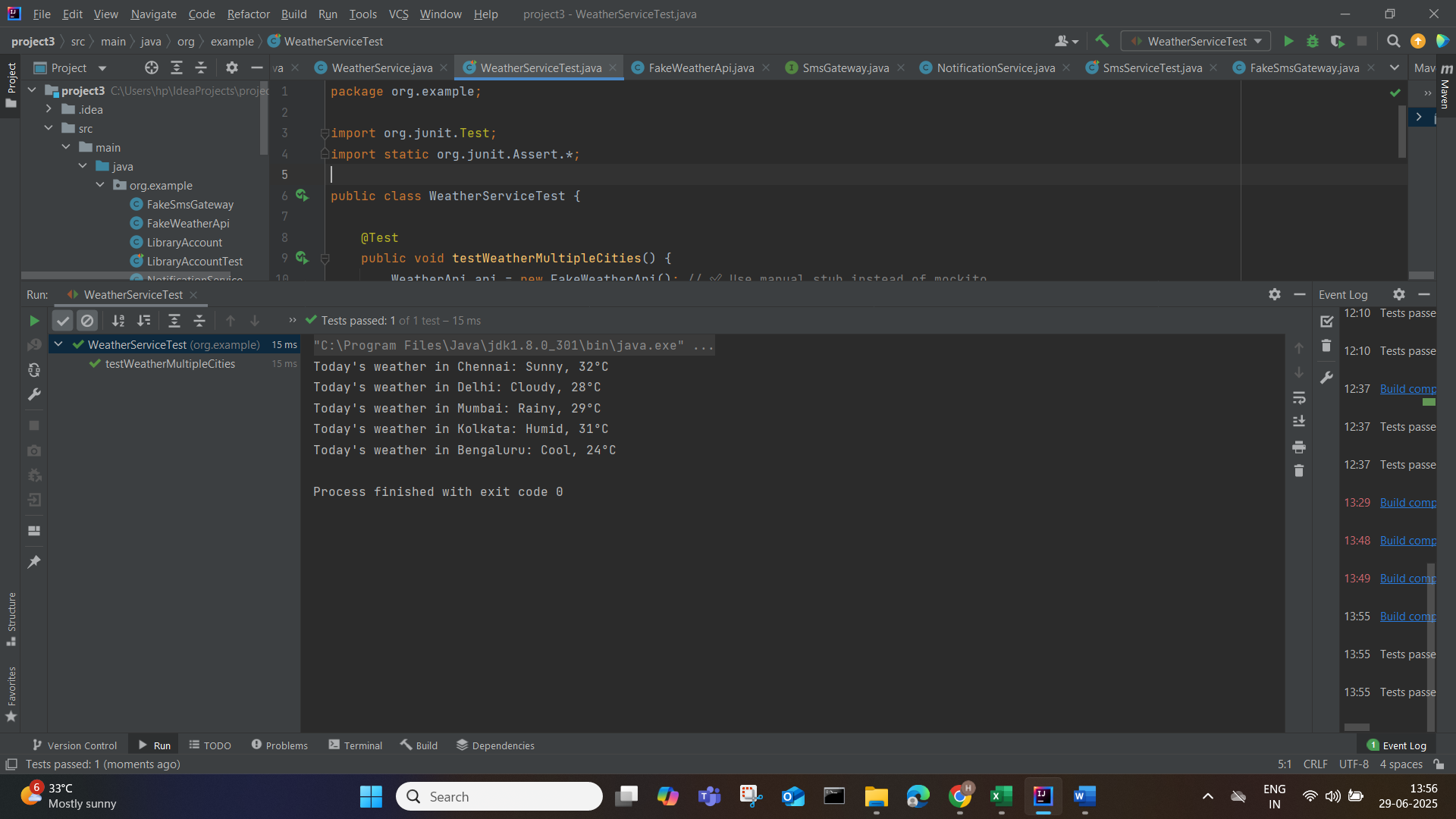
System.out.println(service.fetchTodayWeather("Kolkata"));

System.out.println(service.fetchTodayWeather("Bengaluru"));

}

}

**Output:**



**Conclusion:**

The test successfully verifies WeatherService functionality using mocked data, ensuring correctness without relying on external services. This approach enhances test reliability, speed, and control, making it ideal for isolated unit testing.

**EXERCISE 2: VERIFYING INTERACTIONS**

**Introduction:**

This test illustrates the use of **Mockito’s interaction verification** to ensure that the NotificationService correctly interacts with its dependency, SmsGateway, by sending the right SMS message during user notification.

**Objective:**

* **Mock Dependencies:** Replace the real SmsGateway with a mock to isolate and control the test environment.
* **Trigger Notification:** Call the notifyUser() method to simulate sending an OTP message to a user.
* **Verify Interaction:** Use verify() to confirm that sendSms() was invoked with the expected message.

**Implementation Breakdown:**

**SmsGateway.java:**

interface SmsGateway {

void sendSms(String message);

}

class NotificationService {

private final SmsGateway gateway;

public NotificationService(SmsGateway gateway) {

this.gateway = gateway;

}

public void notifyUser(String username) {

String message = "Hi " + username + ", your OTP is 123456.";

gateway.sendSms(message);

System.out.println("Notification sent: " + message);

}

}

**SmsServiceTest.java:**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.mock;

import static org.mockito.Mockito.verify;

public class SmsServiceTest {

@Test

public void testSmsSentWithCorrectMessage() {

SmsGateway mockGateway = mock(SmsGateway.class);

NotificationService service = new NotificationService(mockGateway);

service.notifyUser("Harinee");

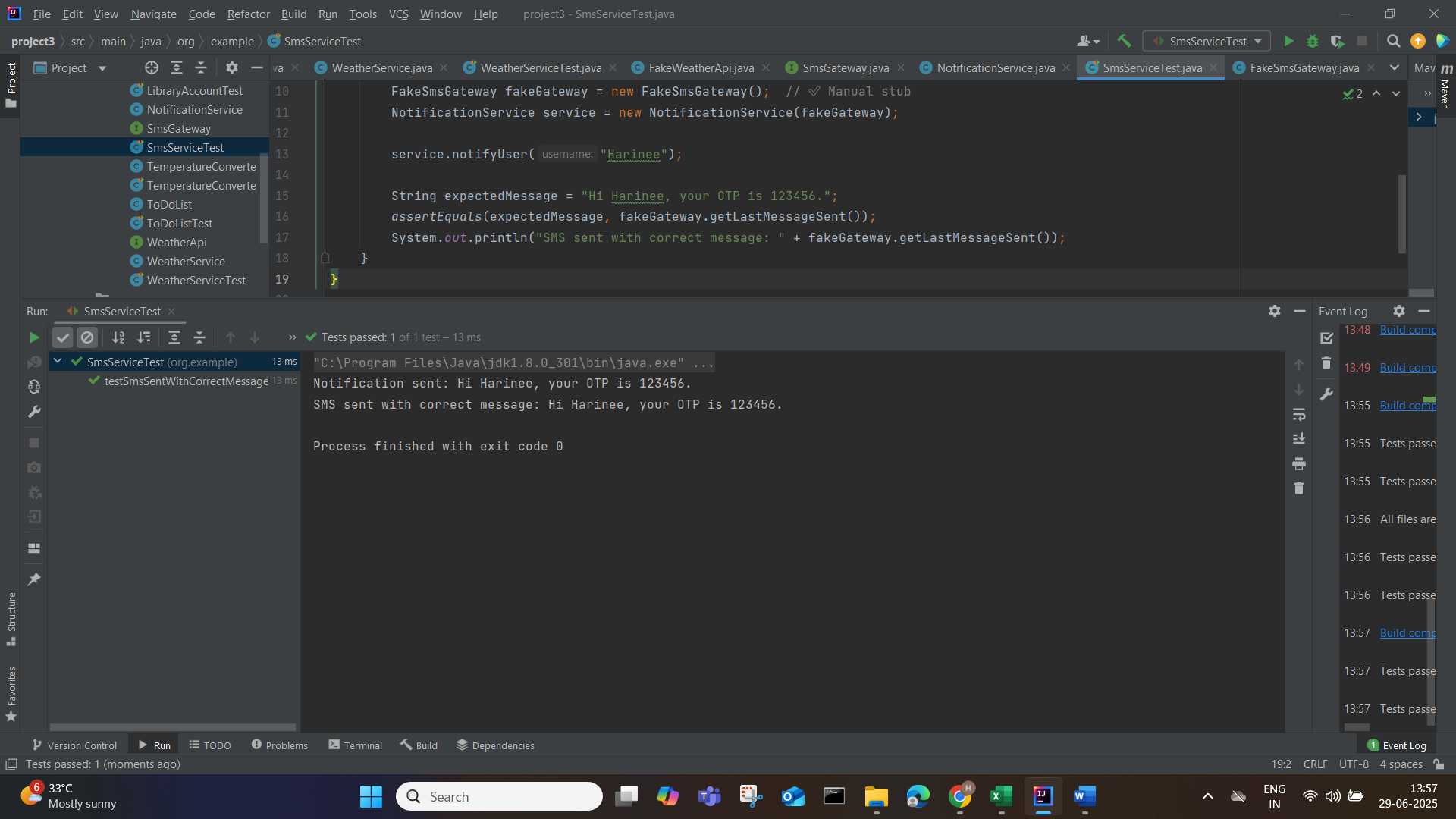
verify(mockGateway).sendSms("Hi Harinee, your OTP is 123456.");

System.out.println("SMS sent with correct message.");

}

}

**Output:**



**Conclusion:**

The test successfully validates that the NotificationService interacts correctly with SmsGateway, ensuring the right SMS content is sent. This interaction-based testing strengthens confidence in the behavior of service-to-dependency communication.