**3. Mockito Advanced exercises**

**Exercise 1: Mocking Databases and Repositories**

**CODE**

***File name: Repository.java***

package com.example.mock;

public interface Repository {

String getData();

}

***File name: Service.java***

package com.example.mock;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

***File name: ServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class ServiceTest {

@Test

public void testServiceWithMockRepository() {

// Arrange

Repository mockRepository = *mock*(Repository.class);

*when*(mockRepository.getData()).thenReturn("Mock Data");

// Act

Service service = new Service(mockRepository);

String result = service.processData();

// Assert

*assertEquals*("Processed Mock Data", result);

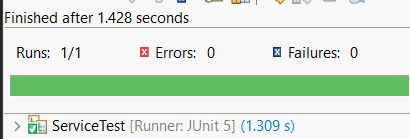
System.*out*.println("Test passed: " + result);

}

}

**OUTPUT**





**Exercise 2: Mocking External Services (RESTful APIs)**

**CODE**

***File name: RestClient.java***

package com.example.mock;

public interface RestClient {

String getUserById(int id); // Simulates an external REST API call

}

***File name: UserService.java***

package com.example.mock;

public class UserService {

private RestClient restClient;

public UserService(RestClient restClient) {

this.restClient = restClient;

}

public String getFormattedUser(int id) {

String userData = restClient.getUserById(id);

return "User Info: " + userData;

}

}

***File name: UserServiceTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class UserServiceTest {

@Test

public void testGetFormattedUser() {

// Arrange

RestClient mockRestClient = *mock*(RestClient.class);

*when*(mockRestClient.getUserById(101)).thenReturn("Name=Alice, Age=30");

UserService userService = new UserService(mockRestClient);

// Act

String result = userService.getFormattedUser(101);

// Assert

*assertEquals*("User Info: Name=Alice, Age=30", result);

// Verify interaction

*verify*(mockRestClient).getUserById(101);

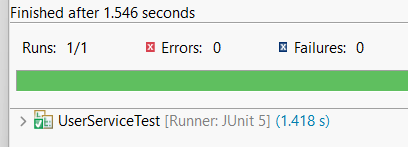
System.*out*.println(" Test passed: " + result);

}

}

**OUTPUT**





**Exercise 3: Mocking File I/O**

**CODE**

***File name: PaymentProcessor.java***

package com.example.mock;

public class PaymentProcessor {

public double getExchangeRate() {

// Simulate a call to an external exchange rate service

return 83.45;

}

public double convertToINR(double usdAmount) {

return usdAmount \* getExchangeRate();

}

}

***File name: PaymentProcessorTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class PaymentProcessorTest {

@Test

public void testConvertToINRWithSpy() {

// Create a real object

PaymentProcessor realProcessor = new PaymentProcessor();

// Spy the real object (partial mock)

PaymentProcessor spyProcessor = *spy*(realProcessor);

// Override just one method

*when*(spyProcessor.getExchangeRate()).thenReturn(80.0);

// Call the real method using the mocked dependency

double result = spyProcessor.convertToINR(10.0);

*assertEquals*(800.0, result);

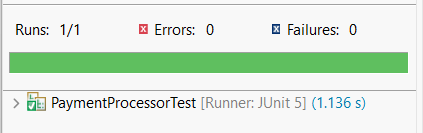
System.*out*.println("INR Conversion (with mocked rate): ₹" + result);

}

}

**OUTPUT**





**Exercise 4: Mocking Network Interactions**

**CODE**

***File name: EmailService.java***

*package com.example.mock;*

*public interface EmailService {*

*void send(String recipient, String message);*

*}*

***File name: NotificationSender.java***

package com.example.mock;

public class NotificationSender {

private EmailService emailService;

public NotificationSender(EmailService emailService) {

this.emailService = emailService;

}

public void notify(String userId) {

String email = userId + "@example.com";

String msg = "Welcome, " + userId + "!";

emailService.send(email, msg);

}

}

***File name: NotificationSenderTest.java***

package com.example.mock;

import org.junit.jupiter.api.Test;

import org.mockito.ArgumentCaptor;

import static org.mockito.Mockito.\*;

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

public class NotificationSenderTest {

@Test

public void testNotificationWithArgumentCaptor() {

// Create mock

EmailService mockEmail = *mock*(EmailService.class);

// Inject into class under test

NotificationSender sender = new NotificationSender(mockEmail);

// Act

sender.notify("john");

// Capture arguments

ArgumentCaptor<String> recipientCaptor = ArgumentCaptor.*forClass*(String.class);

ArgumentCaptor<String> messageCaptor = ArgumentCaptor.*forClass*(String.class);

// Verify and capture

*verify*(mockEmail).send(recipientCaptor.capture(), messageCaptor.capture());

// Assert captured values

*assertEquals*("john@example.com", recipientCaptor.getValue());

*assertEquals*("Welcome, john!", messageCaptor.getValue());

System.*out*.println("Email sent to: " + recipientCaptor.getValue());

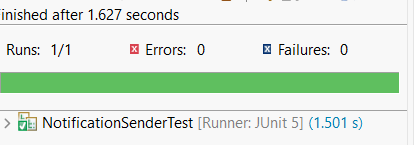
System.*out*.println("Message: " + messageCaptor.getValue());

}

}

**OUTPUT**





**Exercise 5: Mocking Multiple Return Values**

**CODE**

***File name: MathUtils.java***

package com.example.mock;

public class MathUtils {

public static int add(int a, int b) {

return a + b;

}

}

***File name: MathService.java***

package com.example.mock;

public class MathService {

public int doubleSum(int x, int y) {

return 2 \* MathUtils.*add*(x, y);

}

}

**File name: MathServiceTest.java**

package com.example.mock;

import org.junit.jupiter.api.Test;

import org.mockito.MockedStatic;

import static org.mockito.Mockito.\*;

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

public class MathServiceTest {

@Test

public void testDoubleSumWithStaticMock() {

// Mock static method

try (MockedStatic<MathUtils> mocked = *mockStatic*(MathUtils.class)) {

// Stub static method

mocked.when(() -> MathUtils.*add*(2, 3)).thenReturn(10);

// Call method under test

MathService service = new MathService();

int result = service.doubleSum(2, 3);

// Assert result based on mocked value

*assertEquals*(20, result);

// Verify static method was called

mocked.verify(() -> MathUtils.*add*(2, 3));

System.*out*.println("doubleSum returned: " + result + " (with mocked static add)");

}

}

}

**OUTPUT**



