**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <http://maven.apache.org/xsd/maven-4.0.0.xsd>">

<modelVersion>4.0.0</modelVersion>

<groupId>com.firstProject</groupId>

<artifactId>JUnitDemo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>JUnitDemo</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**Exercise 2: Writing Basic JUnit Tests**

**Calculator.java**

**package** com.firstProject.JUnitDemo;

**public** **class** Calculator {

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

**int** result = a + b;

System.***out***.println("Add called: " + result);

**return** result;

}

**public** **int** subtract(**int** a, **int** b) {

**int** result = a - b;

System.***out***.println("Sub called: " + result);

**return** result;

}

}

**CalculatorTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** CalculatorTest {

@Test

**public** **void** testAdd() {

Calculator calc = **new** Calculator();

*assertEquals*(5, calc.add(2, 3));

}

@Test

**public** **void** testSubtract() {

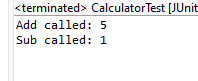
Calculator calc = **new** Calculator();

*assertEquals*(1, calc.subtract(3, 2));

}

}

**Output**



**Exercise 3: Assertions in JUnit**

**AssertionsTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** AssertionsTest {

@Test

**public** **void** testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 2);

*assertFalse*(2 > 5);

Object obj1 = **null**;

*assertNull*(obj1);

Object obj2 = **new** Object();

*assertNotNull*(obj2);

}

}

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

**AdvancedTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**public** **class** AdvancedTest {

**private** Calculator calculator;

// Setup: runs before each @Test

@Before

**public** **void** setUp() {

calculator = **new** Calculator();

System.***out***.println("Setup complete");

}

// Teardown: runs after each @Test

@After

**public** **void** tearDown() {

calculator = **null**;

System.***out***.println("Teardown complete");

}

@Test

**public** **void** testAdd() {

**int** result = calculator.add(4, 6);

*assertEquals*(10, result);

}

@Test

**public** **void** testSubtract() {

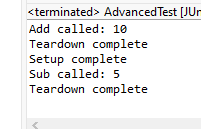
**int** result = calculator.subtract(9, 4);

*assertEquals*(5, result);

}

}

**Output**



**Exercise 1: Parameterized Tests**

**EvenChecker.java**

**package** com.firstProject.JUnitDemo;

**public** **class** EvenChecker {

**public** **static** **boolean** isEven(**int** number) {

**return** number % 2 == 0;

}

}

**EvenCheckerTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Parameterized;

@RunWith(Parameterized.**class**)

**public** **class** EvenCheckerTest {

**private** **int** number;

**private** **boolean** expectedResult;

**public** EvenCheckerTest(**int** number, **boolean** expectedResult) {

**this**.number = number;

**this**.expectedResult = expectedResult;

}

@Parameterized.Parameters

**public** **static** Collection<Object[]> data() {

**return** Arrays.*asList*(**new** Object[][] {

{2, **true**},

{4, **true**},

{6, **true**},

{8, **true**},

{10, **true**},

{1, **false**},

{3, **false**},

{5, **false**},

{7, **false**},

{9, **false**}

});

}

@Test

**public** **void** testIsEven() {

*assertEquals*(expectedResult, EvenChecker.*isEven*(number));

}

}

**Exercise 2: Test Suites and Categories**

**MathTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** MathTest {

@Test

**public** **void** testAddition() {

*assertEquals*(4, 2 + 2);

}

}

**StringTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**public** **class** StringTest {

@Test

**public** **void** testConcatenation() {

*assertEquals*("HelloWorld", "Hello" + "World");

}

}

**AllTests.java**

**package** com.firstProject.JUnitDemo;

**import** org.junit.runner.RunWith;

**import** org.junit.runners.Suite;

@RunWith(Suite.**class**)

@Suite.SuiteClasses({

MathTest.**class**,

StringTest.**class**,

EvenCheckerTest.**class**

})

**public** **class** AllTests {

// This class remains empty, holder for the above annotations

}

**Exercise 3: Test Execution Order**

**OrderedTest.java**

**package** com.firstProject.JUnitDemo;

**import** org.junit.FixMethodOrder;

**import** org.junit.Test;

**import** org.junit.runners.MethodSorters;

@FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)

**public** **class** OrderedTests {

@Test

**public** **void** testA\_First() {

System.***out***.println("First test");

}

@Test

**public** **void** testB\_Second() {

System.***out***.println("Second test");

}

@Test

**public** **void** testC\_Third() {

System.***out***.println("Third test");

}

}

**Exercise 4: Exception Testing**

**ExceptionThrower.java**

**package** com.firstProject.JUnitDemo;

**public** **class** ExceptionThrower {

**public** **void** throwException() {

**throw** **new** IllegalArgumentException("Intentional exception");

}

}

**ExceptionThrowerTest.java**

**package** com.firstProject.JUnitDemo;

**import** org.junit.Test;

**public** **class** ExceptionThrowerTest {

@Test(expected = IllegalArgumentException.**class**)

**public** **void** testExceptionThrown() {

ExceptionThrower thrower = **new** ExceptionThrower();

thrower.throwException();

}

}

**Exercise 5: Timeout and Performance Testing**

**PerformanceTester.java**

**package** com.firstProject.JUnitDemo;

**public** **class** PerformanceTester {

**public** **void** performTask() **throws** InterruptedException {

// Simulate work taking time

Thread.*sleep*(500);

}

}

**PerformanceTesterTest.java**

**package** com.firstProject.JUnitDemo;

**import** org.junit.Test;

**public** **class** PerformanceTesterTest {

@Test(timeout = 1000)

**public** **void** testPerformance() **throws** InterruptedException {

PerformanceTester tester = **new** PerformanceTester();

tester.performTask();

}

}

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

**ExternalApi.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** ExternalApi {

String getData();

}

**MyService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**public** **class** MyServiceTest {

@Test

**public** **void** testExternalApi() {

// Create mock object

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

// Stub method

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

// mock in service

MyService service = **new** MyService(mockApi);

String result = service.fetchData();

// Assert result

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

}

}

**Exercise 2: Verifying Interactions**

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**public** **class** MyServiceTest {

@Test

**public** **void** testExternalApi() {

// Create mock object

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

// Stub method

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

// mock in service

MyService service = **new** MyService(mockApi);

String result = service.fetchData();

// Assert result

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

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

// Verify that getData() was called exactly once

Mockito.*verify*(mockApi).getData();

}

}

**Exercise 3: Argument Matching**

**ExternalApi.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** ExternalApi {

String getData();

**void** sendData(String data);

}

**MyService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

**public** **void** processData(String input) {

api.sendData(input);

}

}

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**public** **class** MyServiceTest {

@Test

**public** **void** testExternalApi() {

// Create mock object

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

// Stub method

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

// mock in service

MyService service = **new** MyService(mockApi);

String result = service.fetchData();

// Assert result

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

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

// Verify that getData() was called exactly once

Mockito.*verify*(mockApi).getData();

}

@Test

**public** **void** testArgumentMatching() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

// Verify that sendData() was called with "Test Data"

Mockito.*verify*(mockApi).sendData(Mockito.*eq*("Test Data"));

}

}

**Exercise 4: Handling Void Methods**

**ExternalApi.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** ExternalApi {

String getData();

**void** sendData(String data);

}

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.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);

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

Mockito.*verify*(mockApi).getData();

}

@Test

**public** **void** testArgumentMatching() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

Mockito.*verify*(mockApi).sendData(Mockito.*eq*("Test Data"));

}

@Test

**public** **void** testHandleVoidMethod() {

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

MyService service = **new** MyService(mockApi);

*doNothing*().when(mockApi).sendData("Test Data");

service.processData("Test Data");

*verify*(mockApi).sendData("Test Data");

}

}

**Exercise 5: Mocking and Stubbing with Multiple Returns**

**ExternalApi.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** ExternalApi {

String getData();

**void** sendData(String data);

}

**MyServicesTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.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);

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

Mockito.*verify*(mockApi).getData();

}

@Test

**public** **void** testArgumentMatching() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

Mockito.*verify*(mockApi).sendData(Mockito.*eq*("Test Data"));

}

@Test

**public** **void** testHandleVoidMethod() {

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

MyService service = **new** MyService(mockApi);

*doNothing*().when(mockApi).sendData("Test Data");

service.processData("Test Data");

*verify*(mockApi).sendData("Test Data");

}

@Test

**public** **void** testMultipleReturns() {

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

MyService service = **new** MyService(mockApi);

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

System.***out***.println(service.fetchData());

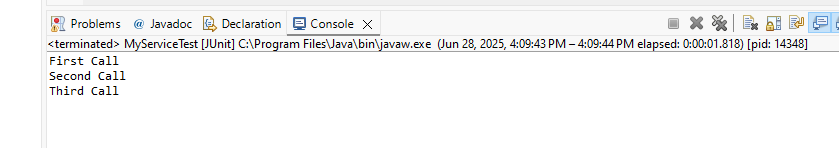
System.***out***.println(service.fetchData());

System.***out***.println(service.fetchData());

}

}

**Output**



**Exercise 6: Verifying Interaction Order**

**ExternalApi.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** ExternalApi {

String getData();

**void** sendData(String data);

**void** log(String message);

}

**MyService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

**public** **void** processData(String input) {

api.sendData(input);

api.log("Data sent: " + input);

}

}

**MyService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

**public** **void** processData(String input) {

api.sendData(input);

api.log("Data sent: " + input);

}

}

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**import** org.mockito.InOrder;

**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);

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

Mockito.*verify*(mockApi).getData();

}

@Test

**public** **void** testArgumentMatching() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

Mockito.*verify*(mockApi).sendData(Mockito.*eq*("Test Data"));

}

@Test

**public** **void** testHandleVoidMethod() {

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

MyService service = **new** MyService(mockApi);

*doNothing*().when(mockApi).sendData("Test Data");

service.processData("Test Data");

*verify*(mockApi).sendData("Test Data");

}

@Test

**public** **void** testMultipleReturns() {

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

MyService service = **new** MyService(mockApi);

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

System.***out***.println(service.fetchData());

System.***out***.println(service.fetchData());

System.***out***.println(service.fetchData());

}

@Test

**public** **void** testInteractionOrder() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

InOrder inOrder = Mockito.*inOrder*(mockApi);

inOrder.verify(mockApi).sendData("Test Data");

inOrder.verify(mockApi).log("Data sent: Test Data");

}

}

**Exercise 7: Handling Void Methods with Exceptions**

**MyServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.junit.Assert.*assertEquals*;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.Test;

**import** org.mockito.Mockito;

**import** org.mockito.InOrder;

**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);

}

@Test

**public** **void** testVerifyInteraction() {

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

MyService service = **new** MyService(mockApi);

service.fetchData();

Mockito.*verify*(mockApi).getData();

}

@Test

**public** **void** testArgumentMatching() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

Mockito.*verify*(mockApi).sendData(Mockito.*eq*("Test Data"));

}

@Test

**public** **void** testHandleVoidMethod() {

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

MyService service = **new** MyService(mockApi);

*doNothing*().when(mockApi).sendData("Test Data");

service.processData("Test Data");

*verify*(mockApi).sendData("Test Data");

}

@Test

**public** **void** testMultipleReturns() {

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

MyService service = **new** MyService(mockApi);

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call")

.thenReturn("Third Call");

System.***out***.println(service.fetchData());

System.***out***.println(service.fetchData());

System.***out***.println(service.fetchData());

}

@Test

**public** **void** testInteractionOrder() {

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

MyService service = **new** MyService(mockApi);

service.processData("Test Data");

InOrder inOrder = Mockito.*inOrder*(mockApi);

inOrder.verify(mockApi).sendData("Test Data");

inOrder.verify(mockApi).log("Data sent: Test Data");

}

@Test(expected = RuntimeException.**class**)

**public** **void** testVoidMethodThrowsException() {

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

MyService service = **new** MyService(mockApi);

*doThrow*(**new** RuntimeException("Simulated Exception")).when(mockApi).sendData("Test Data");

service.processData("Test Data");

}

}

**Advanced Mockito Hands-On Exercises**

**Exercise 1: Mocking Databases and Repositories**

**Respository.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** Repository {

String getData();

}

**Service.java**

**package** com.firstProject.JUnitDemo;

**public** **class** Service {

**private** **final** Repository repository;

**public** Service(Repository repository) {

**this**.repository = repository;

}

**public** String processData() {

String data = repository.getData();

**return** "Processed " + data;

}

}

**ServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.jupiter.api.Test;

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

**public** **class** ServiceTest {

@Test

**public** **void** testServiceWithMockRepository() {

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

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

Service service = **new** Service(mockRepository);

String result = service.processData();

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

}

}

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

**RestClient.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** RestClient {

String getResponse();

}

**ApiService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** ApiService {

**private** **final** RestClient restClient;

**public** ApiService(RestClient restClient) {

**this**.restClient = restClient;

}

**public** String fetchData() {

**return** "Fetched " + restClient.getResponse();

}

}

**ApiServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.jupiter.api.Test;

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

**public** **class** ApiServiceTest {

@Test

**public** **void** testServiceWithMockRestClient() {

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

*when*(mockRestClient.getResponse()).thenReturn("Mock Response");

ApiService apiService = **new** ApiService(mockRestClient);

String result = apiService.fetchData();

*assertEquals*("Fetched Mock Response", result);

}

}

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

**FileReader.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** FileReader {

String read();

}

**FileWriter.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** FileWriter {

**void** write(String content);

}

**FileService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** FileService {

**private** **final** FileReader fileReader;

**private** **final** FileWriter fileWriter;

**public** FileService(FileReader fileReader, FileWriter fileWriter) {

**this**.fileReader = fileReader;

**this**.fileWriter = fileWriter;

}

**public** String processFile() {

String content = fileReader.read();

fileWriter.write("Processed " + content);

**return** "Processed " + content;

}

}

**FileServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.jupiter.api.Test;

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

**public** **class** FileServiceTest {

@Test

**public** **void** testServiceWithMockFileIO() {

FileReader mockFileReader = *mock*(FileReader.**class**);

FileWriter mockFileWriter = *mock*(FileWriter.**class**);

*when*(mockFileReader.read()).thenReturn("Mock File Content");

FileService fileService = **new** FileService(mockFileReader, mockFileWriter);

String result = fileService.processFile();

*assertEquals*("Processed Mock File Content", result);

*verify*(mockFileWriter).write("Processed Mock File Content");

}

}

**Exercise 4: Mocking Network Interactions**

**NetworkClient.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** NetworkClient {

String connect();

}

**NetworkService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** NetworkService {

**private** **final** NetworkClient networkClient;

**public** NetworkService(NetworkClient networkClient) {

**this**.networkClient = networkClient;

}

**public** String connectToServer() {

String connection = networkClient.connect();

**return** "Connected to " + connection;

}

}

**NetworkServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.jupiter.api.Test;

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

**public** **class** NetworkServiceTest {

@Test

**public** **void** testServiceWithMockNetworkClient() {

NetworkClient mockNetworkClient = *mock*(NetworkClient.**class**);

*when*(mockNetworkClient.connect()).thenReturn("Mock Connection");

NetworkService networkService = **new** NetworkService(mockNetworkClient);

String result = networkService.connectToServer();

*assertEquals*("Connected to Mock Connection", result);

}

}

**Exercise 5: Mocking Multiple Return Values**

**Repository.java**

**package** com.firstProject.JUnitDemo;

**public** **interface** Repository {

String getData();

}

**Service.java**

**package** com.firstProject.JUnitDemo;

**public** **class** Service {

**private** **final** Repository repository;

**public** Service(Repository repository) {

**this**.repository = repository;

}

**public** String processData() {

String data = repository.getData();

**return** "Processed " + data;

}

}

**MultiReturnServiceTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** org.junit.jupiter.api.Test;

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

**public** **class** MultiReturnServiceTest {

@Test

**public** **void** testServiceWithMultipleReturnValues() {

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

*when*(mockRepository.getData())

.thenReturn("First Mock Data")

.thenReturn("Second Mock Data");

Service service = **new** Service(mockRepository);

String firstResult = service.processData();

String secondResult = service.processData();

*assertEquals*("Processed First Mock Data", firstResult);

*assertEquals*("Processed Second Mock Data", secondResult);

}

}

**Spring Testing Exercises**

**Exercise 1: Basic Unit Test for a Service Method**

**Calculator.java**

**package** com.firstProject.JUnitDemo;

**public** **class** Calculator {

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

**return** a + b;

}

}

**Calculators.java**

**package** com.firstProject.JUnitDemo;

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

**import** org.junit.jupiter.api.Test;

**public** **class** CalculatorTests {

**private** **final** Calculator calculator = **new** Calculator();

@Test

**public** **void** testAdd() {

*assertEquals*(10, calculator.add(7, 3));

*assertEquals*(0, calculator.add(-2, 2));

*assertEquals*(-5, calculator.add(-3, -2));

}

}

**Exercise 2: Mocking a Repository in a Service Test**

**User.java**

**package** com.firstProject.JUnitDemo;

**public** **class** User {

**private** Long id;

**private** String name;

**public** User() {}

**public** User(Long id, String name) {

**this**.id = id;

**this**.name = name;

}

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**UserRepository.java**

**package** com.firstProject.JUnitDemo;

**import** java.util.Optional;

**public** **interface** UserRepository {

Optional<User> findById(Long id);

**UserService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** UserService {

**private** UserRepository userRepository;

**public** UserService(UserRepository userRepository) {

**this**.userRepository = userRepository;

}

**public** User getUserById(Long id) {

**return** userRepository.findById(id).orElse(**null**);

}

}

**UserServiceTest.java**

**package** com.firstProject.JUnitDemo;

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

**import** **static** org.mockito.Mockito.\*;

**import** java.util.Optional;

**import** org.junit.jupiter.api.Test;

**public** **class** UserServiceTest {

@Test

**public** **void** testGetUserById() {

UserRepository mockRepo = *mock*(UserRepository.**class**);

UserService userService = **new** UserService(mockRepo);

User mockUser = **new** User(1L, "Alice");

*when*(mockRepo.findById(1L)).thenReturn(Optional.*of*(mockUser));

User result = userService.getUserById(1L);

*assertNotNull*(result);

*assertEquals*("Alice", result.getName());

*verify*(mockRepo).findById(1L);

}

**Exercise 3: Testing a REST Controller with MockMvc**

**UserController.java**

**package** com.firstProject.JUnitDemo;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

**public** **class** UserController {

@Autowired

**private** UserService userService;

@GetMapping("/{id}")

**public** ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

**return** ResponseEntity.ok(user);

}

}

**UserControllerTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** **static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

**import** org.junit.jupiter.api.BeforeEach;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.mock.mockito.MockBean;

**import** org.springframework.test.web.servlet.MockMvc;

**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

@WebMvcTest(UserController.**class**)

**public** **class** UserControllerTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@BeforeEach

**void** setup() {

when(userService.getUserById(1L)).thenReturn(**new** User(1L, "Alice"));

}

@Test

**public** **void** testGetUserById() **throws** Exception {

mockMvc.perform(MockMvcRequestBuilders.get("/users/1"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Alice"));

}

}

**Exercise 4: Integration Test with Spring Boot**

**UserIntegrationTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

**import** **static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

**import** org.springframework.boot.test.context.SpringBootTest;

**import** org.springframework.boot.test.mock.mockito.MockBean;

**import** org.springframework.test.web.servlet.MockMvc;

**import** java.util.Optional;

**import** **static** org.mockito.Mockito.\*;

@SpringBootTest

@AutoConfigureMockMvc

**public** **class** UserIntegrationTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserRepository userRepository;

@Test

**public** **void** testFullUserFlow() **throws** Exception {

User user = **new** User(100L, "Zara");

when(userRepository.findById(100L)).thenReturn(Optional.*of*(user));

mockMvc.perform(get("/users/100"))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Zara"));

}

}

**Exercise 5: Test Controller POST Endpoint**

**UserController.java**

**package** com.firstProject.JUnitDemo;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

**public** **class** UserController {

@Autowired

**private** UserService userService;

@GetMapping("/{id}")

**public** ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

**return** ResponseEntity.ok(user);

}

@PostMapping

**public** ResponseEntity<User> createUser(@RequestBody User user) {

**return** ResponseEntity.ok(userService.saveUser(user));

}

}

**UserService.java**

**package** com.firstProject.JUnitDemo;

**public** **class** UserService {

**private** UserRepository userRepository;

**public** UserService(UserRepository userRepository) {

**this**.userRepository = userRepository;

}

**public** User getUserById(Long id) {

**return** userRepository.findById(id).orElse(**null**);

}

**public** User saveUser(User user) {

**return** user;

}

}

**UserPostService.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;

**import** **static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.mock.mockito.MockBean;

**import** org.springframework.test.web.servlet.MockMvc;

**import** **static** org.mockito.Mockito.\*;

@WebMvcTest(UserController.**class**)

**public** **class** UserPostTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Autowired

**private** ObjectMapper objectMapper;

@Test

**public** **void** testCreateUser() **throws** Exception {

User user = **new** User(10L, "Neo");

when(userService.saveUser(any(User.**class**))).thenReturn(user);

mockMvc.perform(post("/users")

.contentType("application/json")

.content(objectMapper.writeValueAsString(user)))

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Neo"));

}

}

**Exercise 6: Test Service Exception Handling**

**UserService.java**

**package** com.firstProject.JUnitDemo;

**import** java.util.NoSuchElementException;

**public** **class** UserService {

**private** UserRepository userRepository;

**public** UserService(UserRepository userRepository) {

**this**.userRepository = userRepository;

}

**public** User getUserById(Long id) {

**return** userRepository.findById(id).orElse(**null**);

}

**public** User saveUser(User user) {

**return** user;

}

**public** User getUserById(Long id) {

**return** userRepository.findById(id)

.orElseThrow(() -> **new** NoSuchElementException("User not found"));

}

}

**UserServiceExceptionTest.java**

**package** com.firstProject.JUnitDemo;

**import** org.junit.jupiter.api.Test;

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

**import** **static** org.mockito.Mockito.\*;

**import** java.util.Optional;

**import** java.util.NoSuchElementException;

**public** **class** UserServiceExceptionTest {

@Test

**public** **void** testUserNotFoundThrowsException() {

UserRepository mockRepo = mock(UserRepository.**class**);

UserService userService = **new** UserService(mockRepo);

when(mockRepo.findById(999L)).thenReturn(Optional.*empty*());

NoSuchElementException thrown = assertThrows(

NoSuchElementException.**class**,

() -> userService.getUserById(999L),

"Expected getUserById() to throw"

);

assertEquals("User not found", thrown.getMessage());

}

}

**Exercise 7: Test Custom Repository Query**

**UserRepository.java**

**import** java.util.List;

**public** **interface** UserRepository {

Optional<User> findById(Long id);

List<User> findByName(String name);

}

**UserRepositoryTest.java**

**package** com.firstProject.JUnitDemo;

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

**import** **static** org.mockito.Mockito.\*;

**import** java.util.List;

**import** java.util.Arrays;

**import** org.junit.jupiter.api.Test;

**public** **class** UserRepositoryTest {

@Test

**public** **void** testFindByName() {

UserRepository mockRepo = mock(UserRepository.**class**);

List<User> users = Arrays.*asList*(

**new** User(1L, "Alice"),

**new** User(2L, "Alice")

);

when(mockRepo.findByName("Alice")).thenReturn(users);

List<User> result = mockRepo.findByName("Alice");

assertEquals(2, result.size());

assertEquals("Alice", result.get(0).getName());

}

}

**Exercise 8: Test Controller Exception Handling**

**GlobalExceptionalHandler.java**

**package** com.firstProject.JUnitDemo;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.\*;

**import** java.util.NoSuchElementException;

@ControllerAdvice

**public** **class** GlobalExceptionHandler {

@ExceptionHandler(NoSuchElementException.**class**)

**public** ResponseEntity<String> handleNotFound(NoSuchElementException ex) {

**return** ResponseEntity

.status(HttpStatus.NOT\_FOUND)

.body("User not found");

}

}

**UserControllerExceptionalTest.java**

**package** com.firstProject.JUnitDemo;

**import** **static** org.mockito.Mockito.\*;

**import** **static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;

**import** **static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

**import** java.util.NoSuchElementException;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.mock.mockito.MockBean;

**import** org.springframework.test.web.servlet.MockMvc;

@WebMvcTest(UserController.**class**)

**public** **class** UserControllerExceptionTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Test

**public** **void** testUserNotFoundReturns404() **throws** Exception {

when(userService.getUserById(404L)).thenThrow(**new** NoSuchElementException("User not found"));

mockMvc.perform(get("/users/404"))

.andExpect(status().isNotFound())

.andExpect(content().string("User not found"));

}

}

**Exercise 9: Parameterized Test with JUnit**

**CalculatorParamTest.java**

**package** com.firstProject.JUnitDemo;

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

**import** org.junit.jupiter.params.ParameterizedTest;

**import** org.junit.jupiter.params.provider.CsvSource;

**public** **class** CalculatorParamTest {

**private** **final** Calculator calculator = **new** Calculator();

@ParameterizedTest

@CsvSource({

"2, 3, 5",

"-1, 1, 0",

"-2, -3, -5",

"0, 0, 0"

})

**void** testAddMultiple(**int** a, **int** b, **int** expected) {

assertEquals(expected, calculator.add(a, b));

}

**Mocking Dependencies in Spring Tests using Mockito**

**Exercise 1: Mocking a Service Dependency in a Controller Test**

**User.java**

**package** com.example.mockito\_test;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.Id;

@Entity

**public** **class** User {

@Id

**private** Long id;

**private** String name;

// Constructors

**public** User() {}

**public** User(Long id, String name) {

**this**.id = id;

**this**.name = name;

}

// Getters and Setters

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

**UserService.java**

**package** com.example.mockito\_test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

@Service

**public** **class** UserService {

@Autowired

**private** UserRepository userRepository;

**public** User getUserById(Long id) {

**return** userRepository.findById(id).orElse(**null**);

}

}

**UserRepository.java**

**package** com.example.mockito\_test;

**import** org.springframework.data.jpa.repository.JpaRepository;

**import** org.springframework.stereotype.Repository;

@Repository

**public** **interface** UserRepository **extends** JpaRepository<User, Long> {

}

**UserController.java**

**package** com.example.mockito\_test;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/users")

**public** **class** UserController {

@Autowired

**private** UserService userService;

@GetMapping("/{id}")

**public** ResponseEntity<User> getUser(@PathVariable Long id) {

User user = userService.getUserById(id);

**return** ResponseEntity.*ok*(user);

}

}

**UserControllerTest.java**

**package** com.example.mockito\_test;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.Mockito;

**import** org.mockito.junit.jupiter.MockitoExtension;

**import** org.junit.jupiter.api.extension.ExtendWith;

**import** org.springframework.http.ResponseEntity;

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

@ExtendWith(MockitoExtension.**class**)

**public** **class** UserControllerTest {

@Mock

**private** UserService userService;

@InjectMocks

**private** UserController userController;

@Test

**public** **void** testGetUser() {

// Arrange

User user = **new** User(1L, "Prabha");

Mockito.when(userService.getUserById(1L)).thenReturn(user);

// Act

ResponseEntity<User> response = userController.getUser(1L);

// Assert

assertEquals(200, response.getStatusCodeValue());

assertEquals("Prabha", response.getBody().getName());

}

}

**Exercise 2: Mocking a Repository in a Service Test**

**UserServiceTest.java**

**package** com.example.mockito\_test;

**import** org.junit.jupiter.api.Test;

**import** org.junit.jupiter.api.extension.ExtendWith;

**import** org.mockito.InjectMocks;

**import** org.mockito.Mock;

**import** org.mockito.Mockito;

**import** org.mockito.junit.jupiter.MockitoExtension;

**import** java.util.Optional;

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

@ExtendWith(MockitoExtension.**class**)

**public** **class** UserServiceTest {

@Mock

**private** UserRepository userRepository;

@InjectMocks

**private** UserService userService;

@Test

**public** **void** testGetUserById\_returnsUser() {

// Arrange

User user = **new** User(1L, "Lakshmi");

Mockito.when(userRepository.findById(1L)).thenReturn(Optional.*of*(user));

// Act

User result = userService.getUserById(1L);

// Assert

assertNotNull(result);

assertEquals("Lakshmi", result.getName());

}

@Test

**public** **void** testGetUserById\_returnsNullForMissingUser() {

// Arrange

Mockito.when(userRepository.findById(2L)).thenReturn(Optional.*empty*());

// Act

User result = userService.getUserById(2L);

// Assert

assertNull(result);

}

}

**Exercise 3: Mocking a Service Dependency in an Integration Test**

**UserIntegrationTest.java**

**package** com.example.mockito\_test;

**import** org.junit.jupiter.api.Test;

**import** org.mockito.Mockito;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

**import** org.springframework.boot.test.context.SpringBootTest;

**import** org.springframework.boot.test.mock.mockito.MockBean;

**import** org.springframework.http.MediaType;

**import** org.springframework.test.web.servlet.MockMvc;

**import** **static** org.mockito.ArgumentMatchers.anyLong;

**import** **static** org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

**import** **static** org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;

@SpringBootTest

@AutoConfigureMockMvc

**public** **class** UserIntegrationTest {

@Autowired

**private** MockMvc mockMvc;

@MockBean

**private** UserService userService;

@Test

**public** **void** testGetUser\_returnsValidResponse() **throws** Exception {

// Arrange

User mockUser = **new** User(1L, "Lakshmi");

Mockito.when(userService.getUserById(anyLong())).thenReturn(mockUser);

// Act & Assert

mockMvc.perform(get("/users/1"))

.andExpect(status().isOk())

.andExpect(content().contentType(MediaType.***APPLICATION\_JSON***))

.andExpect(jsonPath("$.id").value(1))

.andExpect(jsonPath("$.name").value("Lakshmi"));

}

}

**Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**LoggingExample.java**

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** LoggingExample {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(LoggingExample.**class**);

**public** **static** **void** main(String[] args) {

***logger***.error("This is an error message");

***logger***.warn("This is a warning message");

}

}

**Exercise 2: Parameterized Logging**

**ParameterizedLogging.java**

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** ParameterizedLogging {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(ParameterizedLogging.**class**);

**public** **static** **void** main(String[] args) {

String user = "Lakshmi";

**int** score = 98;

***logger***.info("User {} scored {} points in the assessment.", user, score);

}

}

**Exercise 3: Using Different Appenders**

**logback.xml**

<configuration>

<appender name="console" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="file" class="ch.qos.logback.core.FileAppender">

<file>logs/app.log</file>

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

</configuration>

**AppendersExample.java**

**package** com.example.mockito\_test;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** AppendersExample {

**private** **static** **final** Logger ***logger*** = LoggerFactory.*getLogger*(AppendersExample.**class**);

**public** **static** **void** main(String[] args) {

***logger***.info("Logging");

***logger***.debug("Debug");

***logger***.warn("Warning");

***logger***.error("Error");

}

}