JUnit Testing Exercises

Exercise 1: Setting Up JUnit

pom.xml:-

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>junit-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

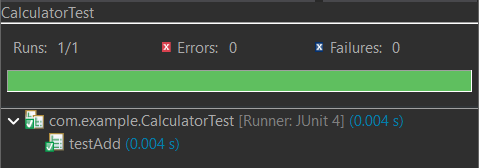
Calculator calc = new Calculator();

int result = calc.add(2, 3);

*assertEquals*(5, result);

}

}



Exercise 2: Writing Basic JUnit Tests

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

*@Test*

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(2, 3);

*assertEquals*(5, result);

}

*@Test*

public void testSubtract() {

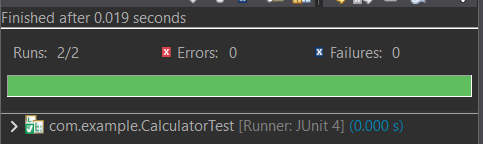
Calculator calc = new Calculator();

int result = calc.subtract(5, 3);

*assertEquals*(2, result);

}

}



Exercise 3: Assertions in JUnit

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

*@Test*

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

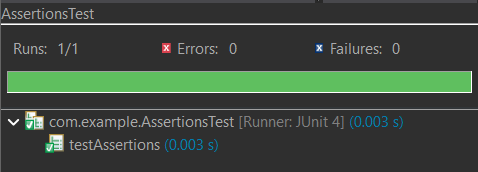
*assertFalse*(5 < 3);

*assertNull*(null);

*assertNotNull*(new Object());

}

}



Exercise 4:

package com.example;

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

public class TestCalculator {

private Calculator calc;

*@BeforeEach*

public void setUp() {

calc = new Calculator();

System.***out***.println("Before each test");

}

*@AfterEach*

public void tearDown() {

System.***out***.println("After each test");

}

*@Test*

public void testAdd() {

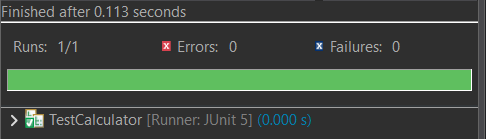
int result = calc.add(10, 20);

Assertions.*assertEquals*(30, result);

}

}





Advanced JUnit Testing Exercises

Exercise 1: Parameterized Tests

package com.example;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

package com.example;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.ValueSource;

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

public class EvenCheckerTest {

private final EvenChecker checker = new EvenChecker();

*@*ParameterizedTest

*@*ValueSource(ints = {2, 4, 6, 8, 10})

void testIsEvenWithEvenNumbers(int number) {

assertTrue(checker.isEven(number));

}

*@*ParameterizedTest

*@*ValueSource(ints = {1, 3, 5, 7, 9})

void testIsEvenWithOddNumbers(int number) {

assertFalse(checker.isEven(number));

}

}

Exercise 2: Test Suites and Categories

package com.example;

import org.junit.platform.suite.api.SelectClasses;

import org.junit.platform.suite.api.Suite;

*@*Suite

*@*SelectClasses({EvenCheckerTest.class, CalculatorTest.class})

public class AllTests {

// Runs both test classes

}

Exercise 3: Test Execution Order

package com.example;

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

*@*TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

*@*Test

*@*Order(1)

void firstTest() {

System.***out***.println("Running first test");

}

*@*Test

*@*Order(2)

void secondTest() {

System.***out***.println("Running second test");

}

*@*Test

*@*Order(3)

void thirdTest() {

System.***out***.println("Running third test");

}

}

Exercise 4: Exception Testing

package com.example;

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("This is an illegal argument");

}

}

package com.example;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

*@*Test

void testThrowsException() {

ExceptionThrower thrower = new ExceptionThrower();

assertThrows(IllegalArgumentException.class, thrower::throwException);

}

}

Exercise 5: Timeout and Performance Testing

package com.example;

public class PerformanceTester {

public void performTask() {

try {

Thread.*sleep*(500); // Simulates work

} catch (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

package com.example;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertTimeout;

import java.time.Duration;

public class PerformanceTesterTest {

*@*Test

void testPerformance() {

PerformanceTester tester = new PerformanceTester();

assertTimeout(Duration.*ofMillis*(1000), tester::performTask); // Must finish within 1 second

}

}

Mockito Hands-On Exercises

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

Exercise 1: Mocking and Stubbing

package com.example;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

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

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

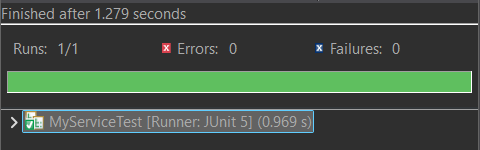
MyService service = new MyService(mockApi);

String result = service.fetchData();

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

}

}



Exercise 2: Verifying Interactions

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

*@Test*

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

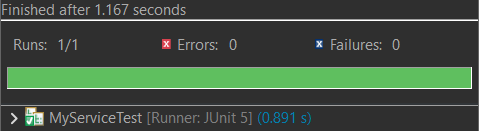
service.fetchData();

// Verify interaction

*verify*(mockApi).getData();

}

}



Exercise 3: Argument Matching

import static org.mockito.Mockito.\*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testArgumentMatching() {

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

*when*(mockApi.getDataWithParam(*anyString*())).thenReturn("Matched Data");

MyService service = new MyService(mockApi);

String result = service.fetchDataWithParam("test123");

assertEquals("Matched Data", result);

*verify*(mockApi).getDataWithParam(*eq*("test123")); // Verifies exact argument

}

}

Exercise 4: Handling Void Methods

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testVoidMethod() {

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

*doNothing*().when(mockApi).sendData(*anyString*());

MyService service = new MyService(mockApi);

service.sendDataToApi("Hello");

*verify*(mockApi).sendData("Hello");

}

}

Exercise 5: Mocking and Stubbing with Multiple Returns

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testMultipleReturns() {

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

*when*(mockApi.getData())

.thenReturn("First Call")

.thenReturn("Second Call");

MyService service = new MyService(mockApi);

assertEquals("First Call", service.fetchData());

assertEquals("Second Call", service.fetchData());

}

}

Exercise 6: Verifying Interaction Order

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

public class MyServiceTest {

*@Test*

public void testInteractionOrder() {

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

MyService service = new MyService(mockApi);

service.stepOne();

service.stepTwo();

InOrder inOrder = *inOrder*(mockApi);

inOrder.verify(mockApi).stepOneCall();

inOrder.verify(mockApi).stepTwoCall();

}

}

Exercise 7: Handling Void Methods with Exceptions

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

public class MyServiceTest {

*@Test*

public void testVoidMethodException() {

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

*doThrow*(new RuntimeException("API error")).when(mockApi).sendData(*anyString*());

MyService service = new MyService(mockApi);

*assertThrows*(RuntimeException.class, () -> service.sendDataToApi("error"));

*verify*(mockApi).sendData("error");

}

}

Advanced Mockito Hands-On Exercises

Exercise 1: Mocking Databases and Repositories

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Assume: Repository has a method getData()

// Assume: Service has a constructor Service(Repository) and method processData()

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)

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Assume: RestClient has method getResponse()

// Assume: ApiService has constructor ApiService(RestClient) and method fetchData()

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

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Assume: FileReader has method read()

// Assume: FileWriter is injected but not directly used in this test

// Assume: FileService has constructor FileService(FileReader, FileWriter) and method processFile()

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);

}

}

Exercise 4: Mocking Network Interactions

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Assume: NetworkClient has method connect()

// Assume: NetworkService has constructor NetworkService(NetworkClient) and method connectToServer()

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

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

// Assume: Repository has method getData()

// Assume: Service has constructor Service(Repository) and method processData()

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

package com.example;

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

import org.junit.jupiter.api.Test;

public class CalculatorServiceTest {

*@Test*

public void testAdd() {

CalculatorService service = new CalculatorService();

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

*assertEquals*(5, result);

}

}

Exercise 2: Mocking a Repository in a Service Test

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.Optional;

public class UserServiceTest {

*@Mock*

private UserRepository userRepository;

*@InjectMocks*

private UserService userService;

public UserServiceTest() {

MockitoAnnotations.*openMocks*(this);

}

*@Test*

public void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("Alice");

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

User result = userService.getUserById(1L);

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

}

}

Exercise 3: Testing a REST Controller with MockMvc

package com.example;

import static org.mockito.Mockito.\*;

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

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

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

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.http.MediaType;

public class UserControllerTest {

private MockMvc mockMvc;

private UserService userService = *mock*(UserService.class);

*@BeforeEach*

public void setup() {

mockMvc = MockMvcBuilders.standaloneSetup(new UserController(userService)).build();

}

*@Test*

public void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Alice");

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

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

.andExpect(status().isOk())

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

}

}

Exercise 4: Integration Test with Spring Boot

import org.junit.jupiter.api.Test;

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

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

import org.springframework.boot.test.web.client.TestRestTemplate;

import org.springframework.http.ResponseEntity;

*@*SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

public class IntegrationTest {

*@*Autowired

private TestRestTemplate restTemplate;

*@Test*

public void testFullFlow() {

ResponseEntity<User> response = restTemplate.getForEntity("/users/1", User.class);

assertEquals(200, response.getStatusCodeValue());

}

}

Exercise 5: Test Controller POST Endpoint

package com.example;

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

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

import static org.mockito.Mockito.\*;

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.springframework.http.MediaType;

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

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

public class UserPostTest {

private MockMvc mockMvc;

private UserService userService = *mock*(UserService.class);

*@BeforeEach*

public void setup() {

mockMvc = MockMvcBuilders.standaloneSetup(new UserController(userService)).build();

}

*@Test*

public void testCreateUser() throws Exception {

User user = new User(1L, "Bob");

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

mockMvc.perform(post("/users")

.contentType(MediaType.APPLICATION\_JSON)

.content(new ObjectMapper().writeValueAsString(user)))

.andExpect(status().isOk())

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

}

}

Exercise 6: Test Service Exception Handling

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class ServiceExceptionTest {

*@Test*

public void testUserNotFound() {

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

UserService service = new UserService(repo);

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

User result = service.getUserById(999L);

*assertNull*(result); // Or assertThrows if using exception

}

}

Exercise 7: Test Custom Repository Query

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import java.util.List;

public class CustomQueryTest {

*@Test*

public void testFindByName() {

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

User user = new User(1L, "Alice");

*when*(repo.findByName("Alice")).thenReturn(List.*of*(user));

List<User> users = repo.findByName("Alice");

*assertEquals*(1, users.size());

*assertEquals*("Alice", users.get(0).getName());

}

}

Exercise 8: Test Controller Exception Handling

package com.example;

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

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.springframework.web.context.WebApplicationContext;

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

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

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

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import java.util.NoSuchElementException;

*@*WebMvcTest

public class ExceptionHandlerTest {

private MockMvc mockMvc;

*@BeforeEach*

public void setup() {

UserService mockService = mock(UserService.class);

when(mockService.getUserById(1L)).thenThrow(new NoSuchElementException());

mockMvc = MockMvcBuilders

.standaloneSetup(new UserController(mockService))

.setControllerAdvice(new GlobalExceptionHandler())

.build();

}

*@Test*

public void testHandleNotFoundException() throws Exception {

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

.andExpect(status().isNotFound())

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

}

}

Exercise 9: Parameterized Test with JUnit

package com.example;

import org.junit.jupiter.params.ParameterizedTest;

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

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

public class CalculatorParameterizedTest {

private final CalculatorService calculator = new CalculatorService();

*@ParameterizedTest*

*@CsvSource*({

"1, 2, 3",

"3, 5, 8",

"10, -5, 5"

})

public void testAddMultipleInputs(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

import static org.mockito.Mockito.\*;

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

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

import com.fasterxml.jackson.databind.ObjectMapper;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.springframework.http.MediaType;

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

import org.springframework.test.web.servlet.setup.MockMvcBuilders;

import org.springframework.http.ResponseEntity;

public class UserControllerTest {

private MockMvc mockMvc;

private UserService userService;

*@BeforeEach*

public void setup() {

userService = *mock*(UserService.class);

UserController userController = new UserController();

userController.userService = userService; // Inject mock manually

mockMvc = MockMvcBuilders.standaloneSetup(userController).build();

}

*@Test*

public void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Sayan");

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

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

.andExpect(status().isOk())

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

}

}

Exercise 2: Mocking a Repository in a Service Test

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.Optional;

public class UserServiceTest {

*@Mock*

private UserRepository userRepository;

*@InjectMocks*

private UserService userService;

*@BeforeEach*

public void setup() {

MockitoAnnotations.*openMocks*(this);

}

*@Test*

public void testGetUserById() {

User user = new User();

user.setId(1L);

user.setName("Sayan");

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

User result = userService.getUserById(1L);

*assertNotNull*(result);

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

}

}

Exercise 3: Mocking a Service Dependency in an Integration Test

package com.example;

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

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.boot.test.mock.mockito.MockBean;

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

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

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

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

*@*SpringBootTest

*@*AutoConfigureMockMvc

public class UserIntegrationTest {

*@*Autowired

private MockMvc mockMvc;

*@*MockBean

private UserService userService;

*@Test*

public void testGetUser() throws Exception {

User user = new User();

user.setId(1L);

user.setName("Mocked User");

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

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

.andExpect(status().isOk())

.andExpect(jsonPath("$.name").value("Mocked User"));

}

}

Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>junit-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<!-- Mockito -->

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

</project>

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

package com.example;

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 = "Sayan";

int loginAttempts = 3;

***logger***.info("User {} has attempted to log in {} times", user, loginAttempts);

***logger***.debug("Detailed debug log for user: {}", user);

}

}

Exercise 3: Using Different Appenders

<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{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

</configuration>

package com.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class MultipleAppenderExample {

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

public static void main(String[] args) {

***logger***.info("Logging to both console and file");

***logger***.warn("This is a warning with multiple appenders");

***logger***.error("Error message written to both outputs");

}

}