1. **JUnit\_Basic Testing Exercises.pdf**

**Exercise-1 Setting up Junit**

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

**Exercise-2 Writing Basic Junit Tests**

**SampleClass.java**

package com.aru.bank;

public class SampleClass {

public int add(int a, int b) {

return a + b;

}

public boolean isPositive(int number) {

return number > 0;

}

}

**SampleClassTest.java**

package com.aru.bank;

import org.junit.Test;

import static org.junit.Assert.\*;

public class SampleClassTest {

SampleClass sample = new SampleClass();

@Test

public void testAdd() {

assertEquals(10, sample.add(7, 3));

}

@Test

public void testIsPositive() {

assertTrue(sample.isPositive(5));

assertFalse(sample.isPositive(-3));

}

}

**Exercise-3 Junit Assertions Practice**

**AssertionsTest.java**

package com.aru.bank;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

// assertEquals

assertEquals(5, 2 + 3);

// assertTrue

assertTrue(5 > 3);

// assertFalse

assertFalse(5 < 3);

// assertNull

assertNull(null);

// assertNotNull

assertNotNull(new Object());

}

}

**Exercise-4: AAA Pattern, Setup and Teardown**

package com.aru.bank;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class SampleClassTest {

SampleClass sample;

@Before

public void setUp() {

sample = new SampleClass(); // Arrange

}

@After

public void tearDown() {

sample = null; // Cleanup

}

@Test

public void testAdd\_AAA() {

// Act

int result = sample.add(10, 15);

// Assert

assertEquals(25, result);

}

}

1. **Junit\_Advanced Testing Exercises.pdf**

**Exercise-1: Parameterized Tests**

**EvenChecker.java**

package com.aru;

public class EvenChecker {

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**EvenCheckerTest.java**

package com.aru;

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

import org.junit.jupiter.params.ParameterizedTest;

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

public class EvenCheckerTest {

EvenChecker checker = new EvenChecker();

@ParameterizedTest

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

public void testIsEvenTrue(int num) {

assertTrue(checker.isEven(num));

}

@ParameterizedTest

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

public void testIsEvenFalse(int num) {

assertFalse(checker.isEven(num));

}

}

**Exercise-2: Test Suites and Categories**

**AllTests.java**

package com.aru;

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

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

@Suite

@SelectClasses({ EvenCheckerTest.class })

public class AllTests {

}

**Exercise-3: Test Execution Order**

**OrderedTests.java**

package com.aru;

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

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)

public class OrderedTests {

@Test

@Order(1)

void testOne() {

System.out.println("Test 1");

assertTrue(true);

}

@Test

@Order(2)

void testTwo() {

System.out.println("Test 2");

assertTrue(true);

}

@Test

@Order(3)

void testThree() {

System.out.println("Test 3");

assertTrue(true);

}

}

**Exercise-4: Exception Testing**

**ExceptionThrower.java**

package com.aru;

public class ExceptionThrower {

public void throwException() {

throw new IllegalArgumentException("Boom!");

}

}

**ExceptionThrowerTest.java**

package com.aru;

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

import org.junit.jupiter.api.Test;

public class ExceptionThrowerTest {

@Test

public void testExceptionThrown() {

ExceptionThrower et = new ExceptionThrower();

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

}

}

**Exercise-5 Timeout/Performance Test**

**PerformanceTester.java**

package com.aru;

public class PerformanceTester {

public void performTask() throws InterruptedException {

Thread.sleep(500); // simulate delay

}

}

**PerformanceTesterTest.java**

package com.aru;

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

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.Timeout;

import java.util.concurrent.TimeUnit;

public class PerformanceTesterTest {

@Test

@Timeout(value = 1, unit = TimeUnit.SECONDS)

public void testPerformance() throws InterruptedException {

PerformanceTester pt = new PerformanceTester();

pt.performTask();

}

}

1. **Mockito Exercises.pdf**

**Exercise-1: Mocking and Stubbing**

**ExternalApi.java**

package com.aru;

public interface ExternalApi {

String getData();

}

**MyService.java**

package com.aru;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.aru;

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**

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

**Exercise-3 Argument Matching**

**ExternalApi.java**

package com.aru;

public interface ExternalApi {

String getData();

void sendData(String data);

}

**Test**

@Test

public void testArgumentMatching() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData(); // this may internally call sendData() if modified

verify(mockApi, never()).sendData(anyString()); // adjust as needed

}

**Exercise-4: Handling Void Methods**

**VoidHandler.java**

package com.aru;

public interface VoidHandler {

void doSomething();

}

**VoidService.java**

package com.aru;

public class VoidService {

private VoidHandler handler;

public VoidService(VoidHandler handler) {

this.handler = handler;

}

public void handle() {

handler.doSomething();

}

}

**VoidServiceTest.java**

@Test

public void testVoidMethod() {

VoidHandler mockHandler = mock(VoidHandler.class);

VoidService service = new VoidService(mockHandler);

service.handle();

verify(mockHandler).doSomething();

}

**Exercise-5: Multiple Return Values**

@Test

public void testMultipleReturns() {

ExternalApi mockApi = 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: Verify Order of Interactions**

@Test

public void testOrder() {

ExternalApi mockApi = mock(ExternalApi.class);

mockApi.getData();

mockApi.getData();

InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi, times(2)).getData();

}

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

@Test

public void testVoidMethodThrows() {

VoidHandler mockHandler = mock(VoidHandler.class);

doThrow(new RuntimeException("fail")).when(mockHandler).doSomething();

VoidService service = new VoidService(mockHandler);

assertThrows(RuntimeException.class, service::handle);

}

1. **Mockito\_AdvancedExercises.pdf**

**Exercise-1: Mocking Databases and Repositories**

**Repository.java**

package com.aru;

public interface Repository {

String getData();

}

**Service.java**

package com.aru;

public class Service {

private Repository repository;

public Service(Repository repository) {

this.repository = repository;

}

public String processData() {

return "Processed " + repository.getData();

}

}

**ServiceTest.java**

package com.aru;

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() {

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 REST APIs**

**RestClient.java**

package com.aru;

public interface RestClient {

String getResponse();

}

**ApiService.java**

package com.aru;

public class ApiService {

private RestClient client;

public ApiService(RestClient client) {

this.client = client;

}

public String fetchData() {

return "Fetched " + client.getResponse();

}

}

**ApiServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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 and FileWriter.java**

package com.aru;

public interface FileReader {

String read();

}

public interface FileWriter {

void write(String data);

}

**FileService.java**

package com.aru;

public class FileService {

private FileReader reader;

private FileWriter writer;

public FileService(FileReader reader, FileWriter writer) {

this.reader = reader;

this.writer = writer;

}

public String processFile() {

return "Processed " + reader.read();

}

}

**FileServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class FileServiceTest {

@Test

public void testServiceWithMockFileIO() {

FileReader mockReader = mock(FileReader.class);

FileWriter mockWriter = mock(FileWriter.class);

when(mockReader.read()).thenReturn("Mock File Content");

FileService service = new FileService(mockReader, mockWriter);

String result = service.processFile();

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

}

}

**Exercise-4: Mocking Network Interactions**

**NetworkClient.java**

package com.aru;

public interface NetworkClient {

String connect();

}

**NetworkService.java**

package com.aru;

public class NetworkService {

private NetworkClient client;

public NetworkService(NetworkClient client) {

this.client = client;

}

public String connectToServer() {

return "Connected to " + client.connect();

}

}

**NetworkServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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 service = new NetworkService(mockNetworkClient);

String result = service.connectToServer();

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

}

}

**Exercise-5: Multiple Return Values**

**MultiReturnServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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 first = service.processData();

String second = service.processData();

assertEquals("Processed First Mock Data", first);

assertEquals("Processed Second Mock Data", second);

}

}

1. **Mockito Mock Dependencies Exercises.pdf**

**Exercise-1: Mocking Service in Controller Test**

**UserControllerTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

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

import org.springframework.http.ResponseEntity;

public class UserControllerTest {

@Test

public void testGetUser() {

UserService mockService = mock(UserService.class);

UserController controller = new UserController();

controller.setUserService(mockService);

User mockUser = new User(1L, "Aru");

when(mockService.getUserById(1L)).thenReturn(mockUser);

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

assertEquals(200, response.getStatusCodeValue());

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

}

}

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

**UserServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

public class UserServiceTest {

@Test

public void testGetUserById() {

UserRepository mockRepo = mock(UserRepository.class);

UserService service = new UserService();

service.setUserRepository(mockRepo);

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

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

assertEquals("TestUser", service.getUserById(1L).getName());

}

}

**Exercise-3: Integration Test with Mocked Service**

**User IntegrationTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

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

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

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

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

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

import static org.mockito.Mockito.when;

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 testUserEndpoint() throws Exception {

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

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

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

.andExpect(status().isOk())

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

}

}

1. **SLF4J Logging Exercises.pdf**

**Exercise-1: Error and Warning Log**

**LoggingExample.java**

package com.aru;

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**

public class ParamLoggingExample {

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

public static void main(String[] args) {

String user = "Aru";

int code = 200;

logger.info("User {} received status {}", user, code);

}

}

**Exercise-3: Custom Appenders(Log to File + console)**

**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>

**LogConfigDemo.java**

package com.aru;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LogConfigDemo {

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

public static void main(String[] args) {

logger.debug("Debug level log");

logger.info("Info level log");

logger.error("Error level log");

}

}

**4. Junit\_Spring Test Exercises.pdf**

**Exercise-1: Test a Service that Adds Two Numbers**

**CalculatorService.java**

package com.aru;

import org.springframework.stereotype.Service;

@Service

public class CalculatorService {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

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

public class CalculatorServiceTest {

CalculatorService service = new CalculatorService();

@Test

public void testAdd() {

assertEquals(5, service.add(2, 3));

}

}

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

**UserServiceTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

import java.util.Optional;

public class UserServiceTest {

@Test

public void testGetUserById() {

UserRepository mockRepo = mock(UserRepository.class);

UserService service = new UserService();

service.setUserRepository(mockRepo);

User mockUser = new User(1L, "Aru");

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

User result = service.getUserById(1L);

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

}

}

**Exercise-3: Test Controller with MockMvc**

**UserControllerTest.java**

package com.aru;

import org.junit.jupiter.api.Test;

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

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

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

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

import static org.mockito.Mockito.when;

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

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

@WebMvcTest(UserController.class)

public class UserControllerTest {

@Autowired

private MockMvc mockMvc;

@MockBean

private UserService userService;

@Test

public void testGetUser() throws Exception {

User mockUser = new User(1L, "Aru");

when(userService.getUserById(1L)).thenReturn(mockUser);

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

.andExpect(status().isOk())

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

}

}

**Exercise-4: Integration Test(Controller to DB)**

@SpringBootTest

@AutoConfigureMockMvc

public class UserFlowTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testUserFlow() throws Exception {

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

.andExpect(status().isOk());

}

}

**Exercise-5: POST Endpoint Test**

@Test

public void testCreateUser() throws Exception {

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

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

mockMvc.perform(post("/users")

.contentType("application/json")

.content("{\"id\":1,\"name\":\"Aru\"}"))

.andExpect(status().isOk())

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

}

**Exercise-6: Service Exception Handling**

@Test

public void testServiceThrowsException() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findById(99L)).thenThrow(new NoSuchElementException());

UserService service = new UserService();

service.setUserRepository(mockRepo);

assertThrows(NoSuchElementException.class, () -> service.getUserById(99L));

}

**Exercise-7: Custom Query Method**

@Test

public void testFindByName() {

UserRepository repo = mock(UserRepository.class);

when(repo.findByName("Aru")).thenReturn(List.of(new User(1L, "Aru")));

assertEquals(1, repo.findByName("Aru").size());

}

**Exercise -8: @ControllerAdvice Handling**

**GlobalExceptionHandlerTest.java**

@Test

public void testNotFoundException() throws Exception {

when(userService.getUserById(99L)).thenThrow(new NoSuchElementException());

mockMvc.perform(get("/users/99"))

.andExpect(status().isNotFound())

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

}

**Exercise-9: Parameterized Test**

@ParameterizedTest

@ValueSource(ints = {1, 2, 3, 4})

public void testEvenNumbers(int num) {

assertTrue(num > 0);

}