**WEEK-2**

**TDD USING JUNIT5 AND Mockito, SLF4J logging framework**

**Exercise 1: Setting Up Junit [JUNIT5]**

**File name: pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.example</groupId>

<artifactId>junit-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

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

<junit.version>4.13.2</junit.version>

</properties>

<dependencies>

<!-- JUnit 4 for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>11</source>

<target>11</target>

</configuration>

</plugin>

<!-- Maven Surefire Plugin for running tests -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0</version>

</plugin>

</plugins>

</build>

</project>

**File name:Calculator.java**

package com.exercise1;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public double divide(int a, int b) {

if (b == 0) {

throw new IllegalArgumentException("Cannot divide by zero");

}

return (double) a / b;

}

public boolean isEven(int number) {

return number % 2 == 0;

}

}

**File name:CalculatorTest.java**

package com.exercise1;

import org.junit.Test;

import org.junit.Before;

import org.junit.After;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

// This method runs before each test method

calculator = new Calculator();

System.out.println("Setting up test...");

}

@After

public void tearDown() {

System.out.println("Cleaning up after test...");

}

@Test

public void testAdd() {

int result = calculator.add(5, 3);

assertEquals(8, result);

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

assertEquals(0, calculator.add(0, 0));

}

@Test

public void testSubtract() {

assertEquals(2, calculator.subtract(5, 3));

assertEquals(-8, calculator.subtract(-5, 3));

assertEquals(5, calculator.subtract(5, 0));

}

@Test

public void testMultiply() {

assertEquals(15, calculator.multiply(5, 3));

assertEquals(0, calculator.multiply(5, 0));

assertEquals(-15, calculator.multiply(-5, 3));

}

@Test

public void testDivide() {

assertEquals(2.5, calculator.divide(5, 2), 0.001);

assertEquals(-2.5, calculator.divide(-5, 2), 0.001);

}

@Test(expected = IllegalArgumentException.class)

public void testDivideByZero() {

calculator.divide(5, 0);

}

@Test

public void testIsEven() {

assertTrue(calculator.isEven(4));

assertFalse(calculator.isEven(5));

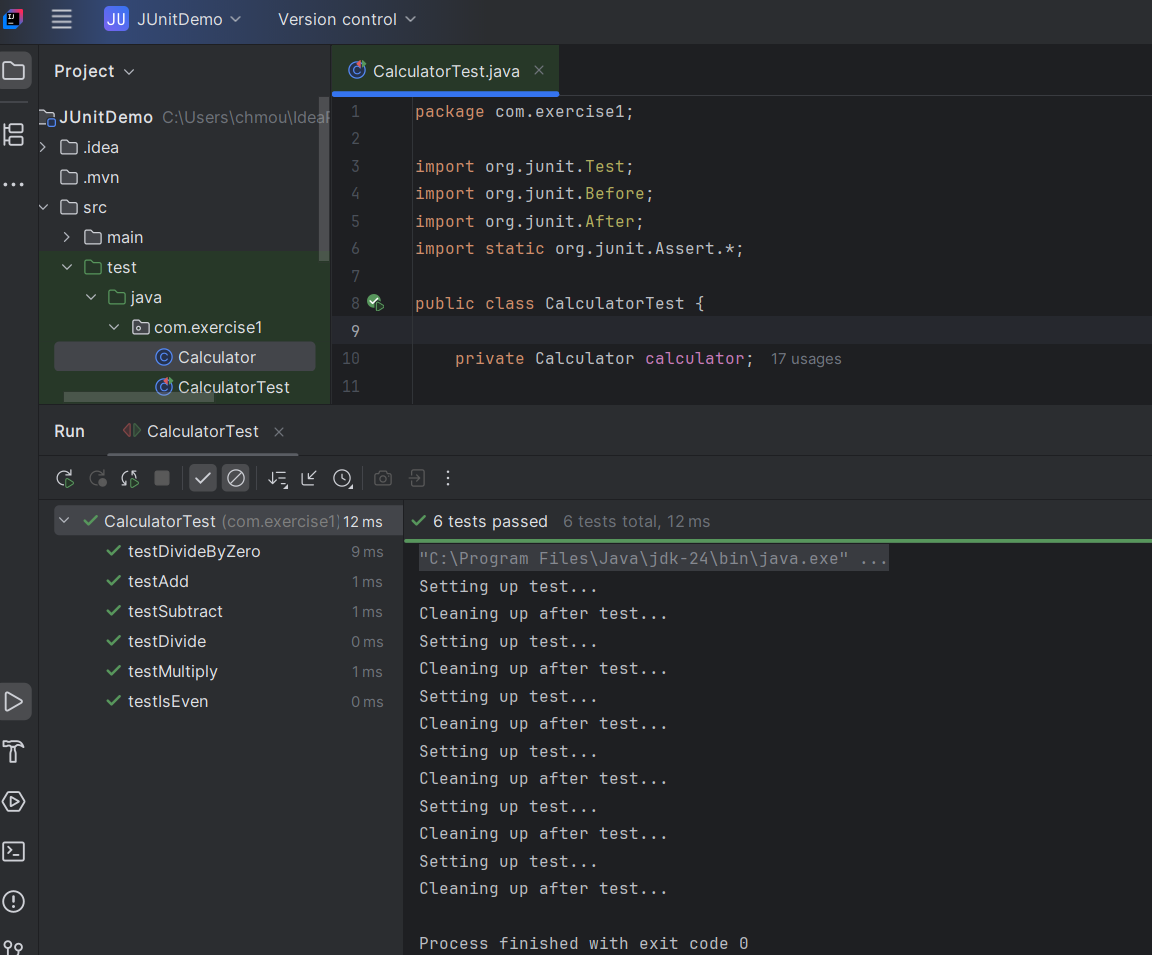
assertTrue(calculator.isEven(0));

assertFalse(calculator.isEven(-3));

}

}

**Output:**

****

**Exercise 3: Assertions in JUnit Scenario**

**File name:pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.example</groupId>

<artifactId>junit-assertions-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<junit.version>5.9.2</junit.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>compile</scope>

</dependency>

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.11.0</version>

<scope>compile</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M9</version>

</plugin>

</plugins>

</build>

</project>

**File name:AssertionsTest.java**

package com.exercise3;

import org.junit.Test;

import org.junit.jupiter.api.DisplayName;

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

public class AssertionsTest {

@Test

@DisplayName("Testing various JUnit assertions")

public void testAssertions() {

assertEquals(5, 2 + 3, "2 + 3 should equal 5");

assertEquals("Hello", "Hello", "Strings should be equal");

assertTrue(5 > 3, "5 should be greater than 3");

assertTrue("Java".contains("a"), "Java should contain 'a'");

assertFalse(5 < 3, "5 should not be less than 3");

assertFalse("Java".isEmpty(), "Java string should not be empty");

String nullString = null;

assertNull(nullString, "String should be null");

assertNotNull(new Object(), "Object should not be null");

assertNotNull("Hello", "String should not be null");

}

@Test

@DisplayName("Testing additional assertion methods")

public void testMoreAssertions() {

String str1 = "Hello";

String str2 = str1;

assertSame(str1, str2, "References should point to same object");

String str3 = new String("Hello");

String str4 = new String("Hello");

assertNotSame(str3, str4, "References should point to different objects");

int[] expected = {1, 2, 3};

int[] actual = {1, 2, 3};

assertArrayEquals(expected, actual, "Arrays should be equal");

assertThrows(ArithmeticException.class, () -> {

int result = 10 / 0;

}, "Should throw ArithmeticException for division by zero");

assertTimeout(java.time.Duration.ofSeconds(2), () -> {

Thread.sleep(1000);

}, "Operation should complete within 2 seconds");

}

@Test

@DisplayName("Testing assertion with custom messages")

public void testAssertionsWithMessages() {

String name = "John";

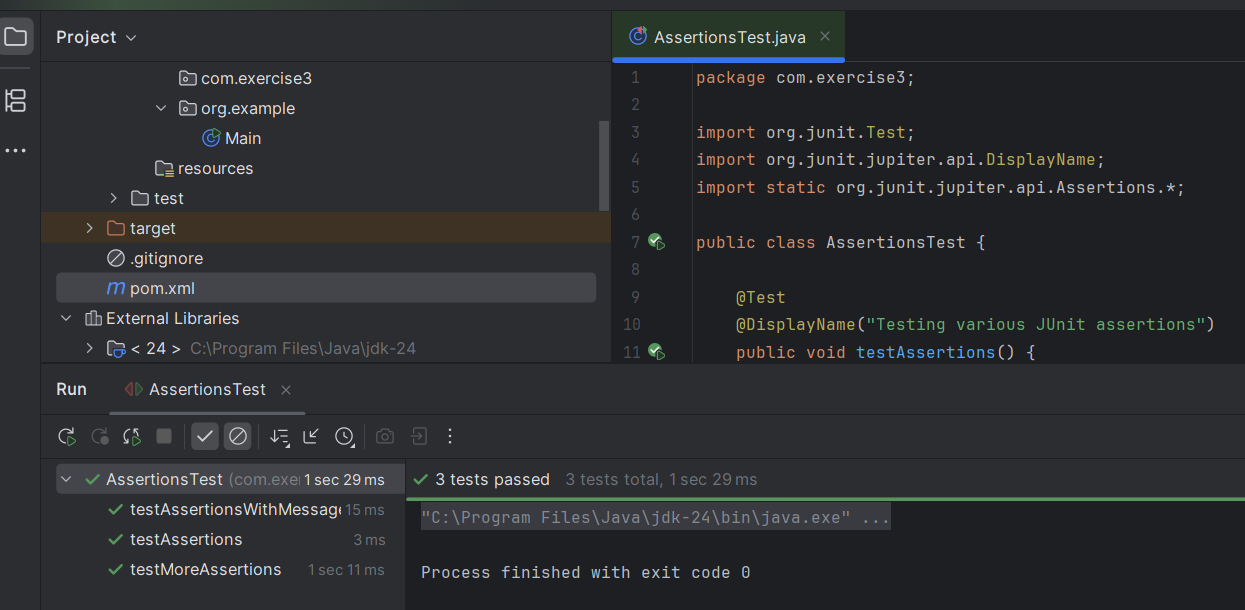
int age = 25;

assertEquals("John", name, () -> "Expected name to be John but was: " + name);

assertTrue(age >= 18, () -> "Person should be adult, age: " + age);

}

}

**Output:**

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

**Teardown Methods in JUnit**

**File name: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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>BankAccountApp</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 4 for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<!-- Compiler plugin to use Java 8 or above -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

**File name:BankAccount.java**

package org.exercise4;

public class BankAccount {

private String accountNumber;

private double balance;

private String ownerName;

private boolean isActive;

public BankAccount(String accountNumber, String ownerName, double initialBalance) {

this.accountNumber = accountNumber;

this.ownerName = ownerName;

this.balance = initialBalance;

this.isActive = true;

}

public void deposit(double amount) {

if (!isActive) {

throw new IllegalStateException("Account is not active");

}

if (amount <= 0) {

throw new IllegalArgumentException("Deposit amount must be positive");

}

balance += amount;

}

public void withdraw(double amount) {

if (!isActive) {

throw new IllegalStateException("Account is not active");

}

if (amount <= 0) {

throw new IllegalArgumentException("Withdrawal amount must be positive");

}

if (amount > balance) {

throw new IllegalArgumentException("Insufficient funds");

}

balance -= amount;

}

public double getBalance() {

return balance;

}

public String getAccountNumber() {

return accountNumber;

}

public String getOwnerName() {

return ownerName;

}

public boolean isActive() {

return isActive;

}

public void closeAccount() {

isActive = false;

}

public void transfer(BankAccount targetAccount, double amount) {

this.withdraw(amount);

targetAccount.deposit(amount);

}

}

**File name:BankAccountTest.java**

import org.junit.\*;

import static org.junit.Assert.\*;

import org.exercise4.BankAccount;

public class BankAccountTest {

// Test fixtures - shared test data

private BankAccount sourceAccount;

private BankAccount targetAccount;

private final String TEST\_ACCOUNT\_1 = "ACC001";

private final String TEST\_ACCOUNT\_2 = "ACC002";

private final String OWNER\_NAME\_1 = "John Doe";

private final String OWNER\_NAME\_2 = "Jane Smith";

private final double INITIAL\_BALANCE = 1000.0;

// Setup method - runs before each test

@Before

public void setUp() {

System.out.println("Setting up test fixtures...");

// Initialize test objects

sourceAccount = new BankAccount(TEST\_ACCOUNT\_1, OWNER\_NAME\_1, INITIAL\_BALANCE);

targetAccount = new BankAccount(TEST\_ACCOUNT\_2, OWNER\_NAME\_2, 500.0);

System.out.println("Test fixtures ready");

}

// Teardown method - runs after each test

@After

public void tearDown() {

System.out.println("Cleaning up test fixtures...");

// Clean up resources

sourceAccount = null;

targetAccount = null;

System.out.println("Test cleanup completed");

}

// Setup method - runs once before all tests in the class

@BeforeClass

public static void setUpClass() {

System.out.println("=== Starting BankAccount Test Suite ===");

}

// Teardown method - runs once after all tests in the class

@AfterClass

public static void tearDownClass() {

System.out.println("=== BankAccount Test Suite Completed ===");

}

@Test

public void testDeposit\_ValidAmount\_IncreasesBalance() {

// ARRANGE

double depositAmount = 250.0;

double expectedBalance = INITIAL\_BALANCE + depositAmount;

// ACT

sourceAccount.deposit(depositAmount);

// ASSERT

assertEquals("Balance should increase by deposit amount",

expectedBalance, sourceAccount.getBalance(), 0.01);

}

@Test

public void testWithdraw\_ValidAmount\_DecreasesBalance() {

// ARRANGE

double withdrawAmount = 300.0;

double expectedBalance = INITIAL\_BALANCE - withdrawAmount;

// ACT

sourceAccount.withdraw(withdrawAmount);

// ASSERT

assertEquals("Balance should decrease by withdrawal amount",

expectedBalance, sourceAccount.getBalance(), 0.01);

}

@Test(expected = IllegalArgumentException.class)

public void testWithdraw\_InsufficientFunds\_ThrowsException() {

// ARRANGE

double withdrawAmount = INITIAL\_BALANCE + 100.0; // More than available

// ACT

sourceAccount.withdraw(withdrawAmount);

// ASSERT - Exception expected (handled by annotation)

}

@Test(expected = IllegalArgumentException.class)

public void testDeposit\_NegativeAmount\_ThrowsException() {

// ARRANGE

double negativeAmount = -50.0;

// ACT

sourceAccount.deposit(negativeAmount);

// ASSERT - Exception expected (handled by annotation)

}

@Test

public void testTransfer\_ValidAmount\_UpdatesBothAccounts() {

// ARRANGE

double transferAmount = 200.0;

double expectedSourceBalance = INITIAL\_BALANCE - transferAmount;

double expectedTargetBalance = 500.0 + transferAmount;

// ACT

sourceAccount.transfer(targetAccount, transferAmount);

// ASSERT

assertEquals("Source account balance should decrease",

expectedSourceBalance, sourceAccount.getBalance(), 0.01);

assertEquals("Target account balance should increase",

expectedTargetBalance, targetAccount.getBalance(), 0.01);

}

@Test

public void testCloseAccount\_ActiveAccount\_DeactivatesAccount() {

// ARRANGE

assertTrue("Account should initially be active", sourceAccount.isActive());

// ACT

sourceAccount.closeAccount();

// ASSERT

assertFalse("Account should be inactive after closing", sourceAccount.isActive());

}

@Test(expected = IllegalStateException.class)

public void testDeposit\_InactiveAccount\_ThrowsException() {

// ARRANGE

sourceAccount.closeAccount(); // Make account inactive

double depositAmount = 100.0;

// ACT

sourceAccount.deposit(depositAmount);

// ASSERT - Exception expected (handled by annotation)

}

@Test

public void testAccountCreation\_ValidParameters\_SetsPropertiesCorrectly() {

// ARRANGE

String accountNumber = "TEST123";

String ownerName = "Test Owner";

double initialBalance = 750.0;

// ACT

BankAccount newAccount = new BankAccount(accountNumber, ownerName, initialBalance);

// ASSERT

assertEquals("Account number should match", accountNumber, newAccount.getAccountNumber());

assertEquals("Owner name should match", ownerName, newAccount.getOwnerName());

assertEquals("Initial balance should match", initialBalance, newAccount.getBalance(), 0.01);

assertTrue("New account should be active", newAccount.isActive());

}

@Test

public void testMultipleDeposits\_ValidAmounts\_AccumulatesCorrectly() {

// ARRANGE

double deposit1 = 100.0;

double deposit2 = 150.0;

double deposit3 = 75.0;

double expectedFinalBalance = INITIAL\_BALANCE + deposit1 + deposit2 + deposit3;

// ACT

sourceAccount.deposit(deposit1);

sourceAccount.deposit(deposit2);

sourceAccount.deposit(deposit3);

// ASSERT

assertEquals("Multiple deposits should accumulate correctly",

expectedFinalBalance, sourceAccount.getBalance(), 0.01);

}

@Test

public void testMixedTransactions\_DepositAndWithdraw\_CalculatesCorrectBalance() {

// ARRANGE

double deposit1 = 200.0;

double withdrawal1 = 150.0;

double deposit2 = 100.0;

double withdrawal2 = 75.0;

double expectedBalance = INITIAL\_BALANCE + deposit1 - withdrawal1 + deposit2 - withdrawal2;

sourceAccount.deposit(deposit1);

sourceAccount.withdraw(withdrawal1);

sourceAccount.deposit(deposit2);

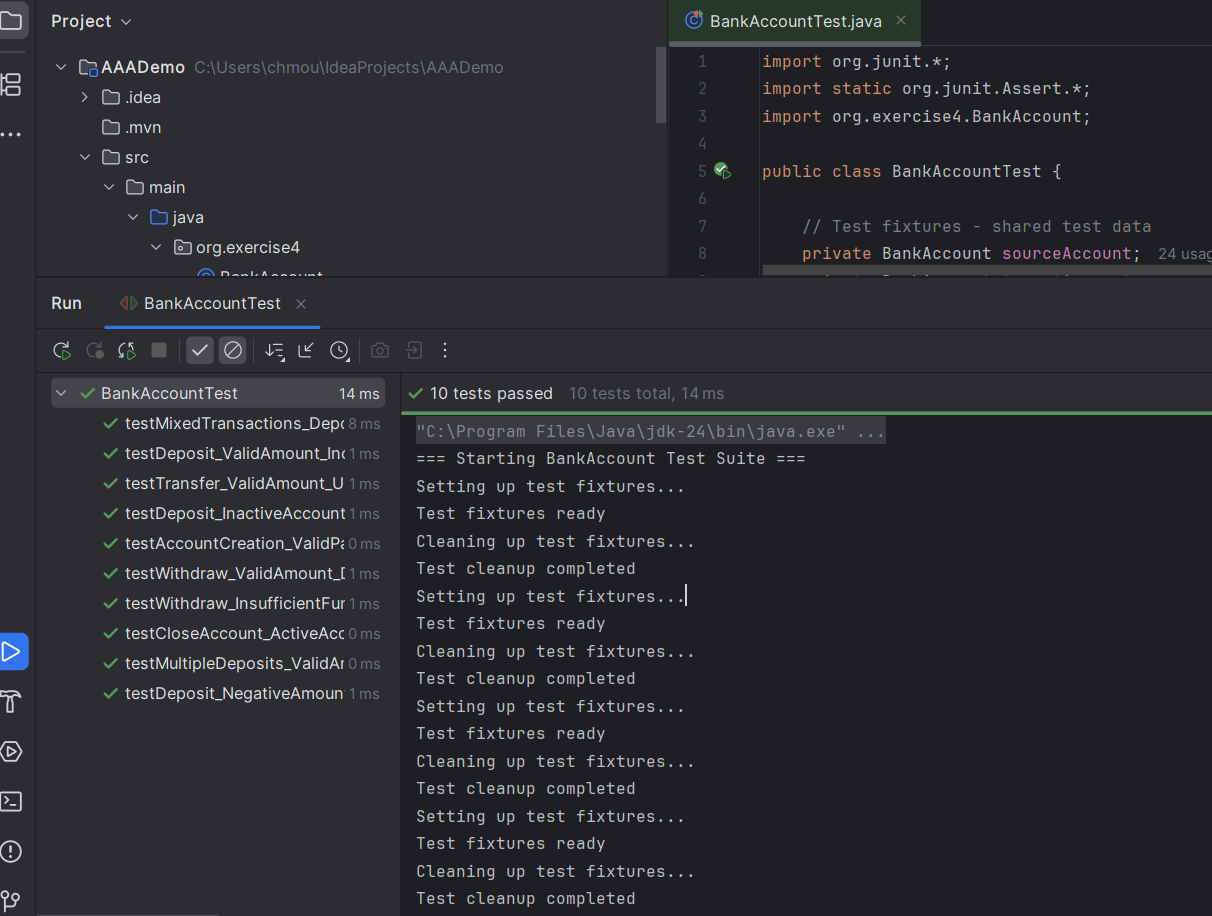
sourceAccount.withdraw(withdrawal2);

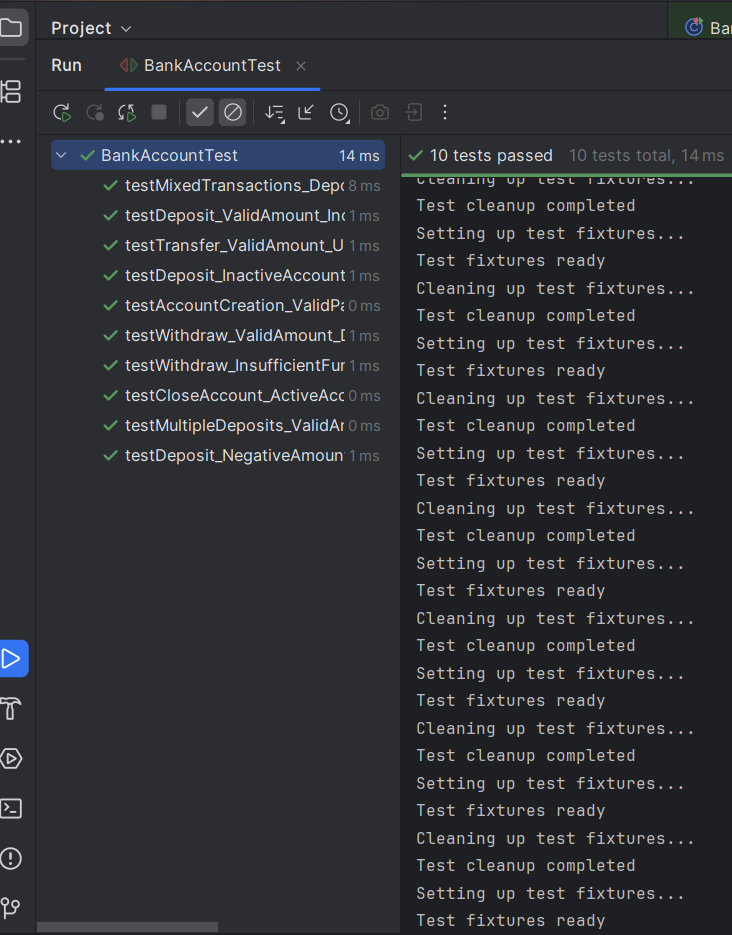
assertEquals("Mixed transactions should calculate correct final balance",

expectedBalance, sourceAccount.getBalance(), 0.01);

}

**Output:**

****

****

**Exercise 1: Mocking and Stubbing [Mockito]**

**File name:pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.exercise1</groupId>

<artifactId>MockingAndStubbing</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

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

</properties>

<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>

</dependencies>

</project>

**File name:MyService.java**

package com.exercise1

public class MyService {

private ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

}

**File name:ExternalApi.java**

package com.exercise1;

public interface ExternalApi {

String getData();

}

**File name:MyServiceTest.java**

import static org.mockito.Mockito.\*;

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

import com.exercise1.ExternalApi;

import com.exercise1.MyService;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testFetchDataReturnsMockedValue() {

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

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

MyService service = new MyService(mockApi);

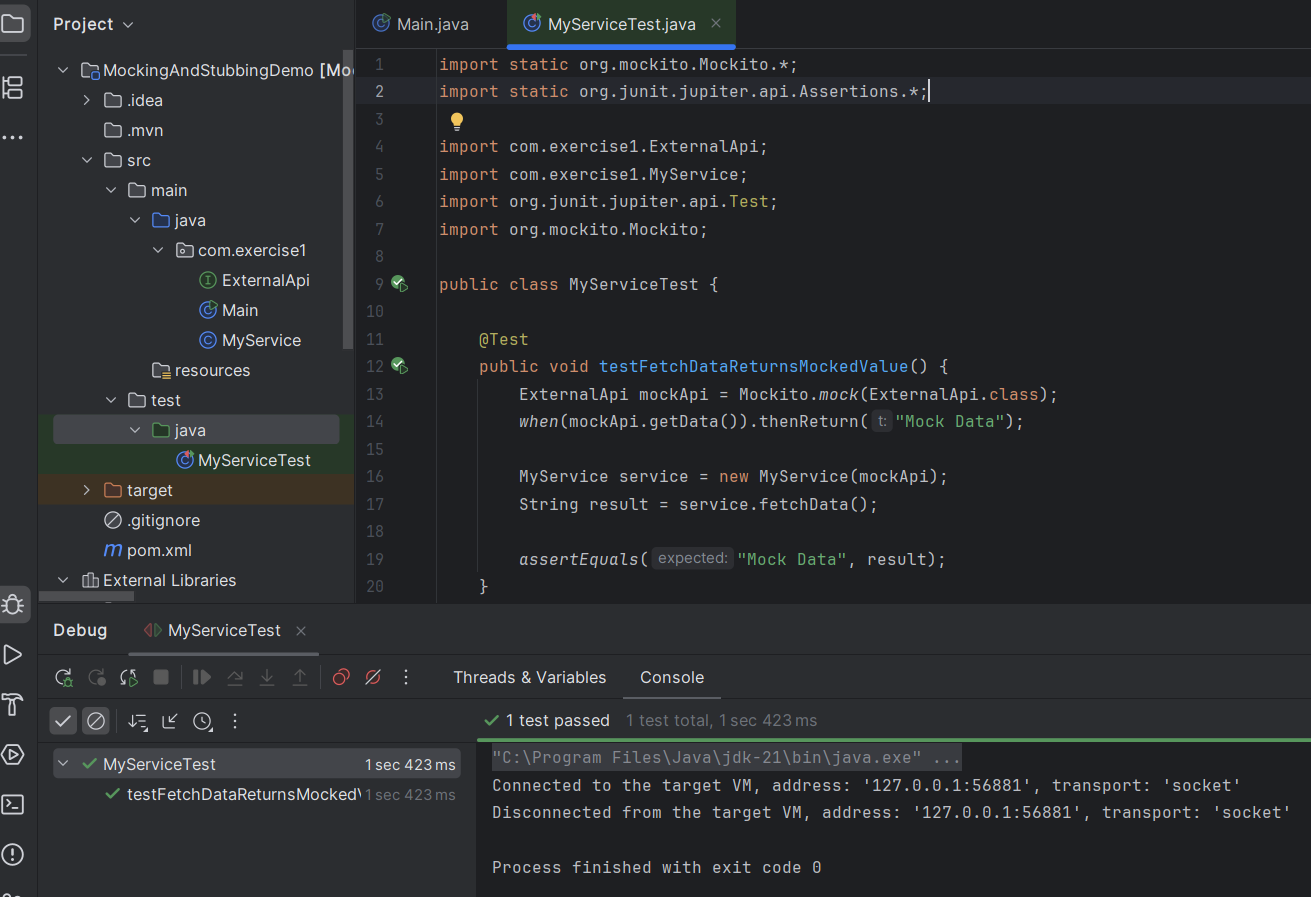
String result = service.fetchData()

assertEquals("Mock Data", result);

}

}

**Output:**

****

**Exercise 2: Verifying Interactions**

**File name:pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.exercise2</groupId>

<artifactId>mockito-verification-demo</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>21</maven.compiler.source>

<maven.compiler.target>21</maven.compiler.target>

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

<junit.version>5.10.1</junit.version>

<mockito.version>5.8.0</mockito.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>${mockito.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-junit-jupiter</artifactId>

<version>${mockito.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.11.0</version>

<configuration>

<source>21</source>

<target>21</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.2.2</version>

<configuration>

<argLine>

-XX:+EnableDynamicAgentLoading

-Djdk.instrument.traceUsage=false

-Xshare:off

</argLine>

</configuration>

</plugin>

</plugins>

</build>

</project>

**File name:MyService.java**

package com.exercise2;

public class MyService {

private final ExternalApi externalApi;

public MyService(ExternalApi externalApi) {

this.externalApi = externalApi;

}

public String fetchData() {

return externalApi.getData();

}

public String fetchDataWithParameter(String param) {

return externalApi.getData(param);

}

public void saveUserData(String userData) {

externalApi.saveData(userData);

}

public boolean processUserData(String data, int times) {

return externalApi.processData(data, times);

}

}

**File name:MyServiceTest.java**

import static org.mockito.Mockito.\*;

import com.exercise2.ExternalApi;

import com.exercise2.MyService;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

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

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension

@ExtendWith(MockitoExtension.class)

public class MyServiceTest {

@Mock

private ExternalApi mockApi;

private MyService service;

@BeforeEach

void setUp() {

service = new MyService(mockApi);

}

@Test

public void testVerifyInteraction() {

service.fetchData();

verify(mockApi).getData();

}

@Test

public void shouldVerifyMethodCalledWithSpecificArguments() {

String userInput = "customer\_data";

service.fetchDataWithParameter(userInput);

verify(mockApi).getData(userInput);

}

@Test

public void shouldVerifyMultipleMethodCalls() {

String userData = "john\_doe\_profile";

service.saveUserData(userData);

service.fetchData();

verify(mockApi).saveData(userData);

verify(mockApi).getData();

}

@Test

public void shouldVerifyMethodWithMultipleParameters() {

String dataToProcess = "analytics\_data";

int processingTimes = 3;

service.processUserData(dataToProcess, processingTimes);

verify(mockApi).processData(dataToProcess, processingTimes);

}

@Test

public void shouldConfirmNoMethodsWereCalled() {

verifyNoInteractions(mockApi);

}

@Test

public void shouldVerifyExactNumberOfCalls() {

service.fetchData();

service.fetchData();

service.fetchData();

verify(mockApi, times(3)).getData();

}

@Test

public void shouldVerifyMethodCalledAtLeastOnce() {

service.fetchData();

service.fetchData();

verify(mockApi, atLeastOnce()).getData();

@Test

public void shouldVerifyMethodNeverCalled() {

service.fetchDataWithParameter("test\_param");

verify(mockApi, never()).getData();

verify(mockApi).getData("test\_param");}

@Test

public void shouldVerifyUsingArgumentMatchers() {

service.fetchDataWithParameter("any\_string\_value");

verify(mockApi).getData(anyString());

verify(mockApi).getData(eq("any\_string\_value"));}

@Test

public void shouldVerifyMethodCallOrder() {

String importantData = "critical\_user\_data"

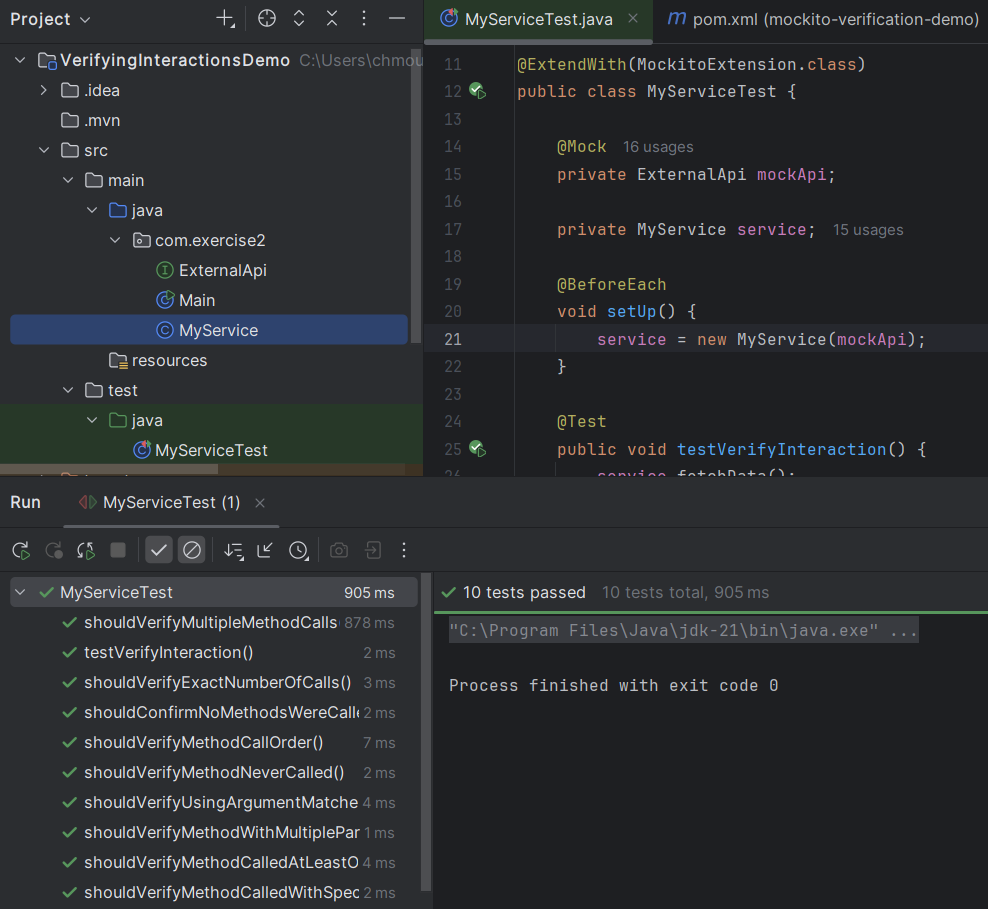
service.fetchData();

service.saveUserData(importantData);

var orderVerifier = inOrder(mockApi);

orderVerifier.verify(mockApi).getData();

orderVerifier.verify(mockApi).saveData(importantData); }}

**Output:**

**SLF4J logging framework**

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

**File name:pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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.example</groupId>

<artifactId>logging-example</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>8</maven.compiler.source>

<maven.compiler.target>8</maven.compiler.target>

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

</properties>

<dependencies>

<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>

**File name:logback.xml**

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

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

<encoder>

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

</encoder>

</appender>

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/application.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/application.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxFileSize>10MB</maxFileSize>

<maxHistory>30</maxHistory>

<totalSizeCap>1GB</totalSizeCap>

</rollingPolicy>

<encoder>

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

</encoder>

</appender>

<appender name="ERROR\_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/error.log</file>

<filter class="ch.qos.logback.classic.filter.LevelFilter">

<level>ERROR</level>

<onMatch>ACCEPT</onMatch>

<onMismatch>DENY</onMismatch>

</filter>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/error.%d{yyyy-MM-dd}.%i.log</fileNamePattern>

<maxFileSize>10MB</maxFileSize>

<maxHistory>90</maxHistory>

</rollingPolicy>

<encoder>

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

</encoder>

</appender>

<logger name="DatabaseService" level="DEBUG" additivity="false">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

</logger>

<root level="INFO">

<appender-ref ref="CONSOLE"/>

<appender-ref ref="FILE"/>

<appender-ref ref="ERROR\_FILE"/>

</root>

</configuration>

**File name:DatabaseService.java**

package com.exercise1;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class DatabaseService {

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

public void connectToDatabase() {

logger.info("Attempting to connect to database");

try {

simulateConnection();

logger.info("Successfully connected to database");

} catch (Exception e) {

logger.error("Failed to connect to database", e);

logger.warn("Application will run in offline mode");

}

}

private void simulateConnection() throws Exception {

if (Math.random() > 0.7) {

throw new Exception("Database connection timeout");

}

}

public void saveData(String data) {

if (data == null) {

logger.warn("Attempted to save null data - operation skipped");

return;

if (data.length() > 1000) {

logger.warn("Data size ({} characters) exceeds recommended limit", data.length());

}

logger.debug("Saving data: {}", data.substring(0, Math.min(50, data.length())) + "...");

try {

logger.info("Data saved successfully");

} catch (Exception e) {

logger.error("Failed to save data", e);

throw new RuntimeException("Save operation failed", e);

}

}

}

**File name:LoggingExample.java**

package com.exercise1;

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) {

LoggingExample example = new LoggingExample();

example.demonstrateLoggingLevels();

example.simulateApplicationFlow();

example.handleDivisionOperation(10, 0);

example.handleDivisionOperation(10, 2);

}

public void demonstrateLoggingLevels() {

logger.trace("TRACE level message - most detailed");

logger.debug("DEBUG level message - debugging information");

logger.info("INFO level message - general information");

logger.warn("WARN level message - potential problem");

logger.error("ERROR level message - error condition");

String username = "john\_doe";

int attempts = 3;

logger.warn("User {} has failed login {} times", username, attempts);

logger.error("Critical error occurred for user {} at attempt {}", username, attempts);

}

public void simulateApplicationFlow() {

logger.info("Starting application simulation");

try {

processUserData("user123");

} catch (Exception e) {

logger.error("Failed to process user data", e);

}

logger.info("Application simulation completed");

}

private void processUserData(String userId) {

logger.debug("Processing data for user: {}", userId);

if (userId == null || userId.trim().isEmpty()) {

logger.warn("Received empty or null user ID");

return;

}

if (userId.length() < 3) {

logger.warn("User ID '{}' is suspiciously short", userId);

}

logger.info("Successfully processed data for user: {}", userId);

}

public void handleDivisionOperation(int dividend, int divisor) {

logger.debug("Attempting division: {} / {}", dividend, divisor);

try {

if (divisor == 0) {

logger.error("Division by zero attempted with dividend: {}", dividend);

throw new ArithmeticException("Division by zero");

}

double result = (double) dividend / divisor;

logger.info("Division successful: {} / {} = {}", dividend, divisor, result);

} catch (ArithmeticException e) {

logger.error("Arithmetic error during division operation", e);

logger.warn("Returning default value due to error");

}

}

}

**Output:**

