WEEK 2 :

Skill : PL/SQL Programming :

**Exercise 1 : Control structures**   
DECLARE

num NUMBER := &Enter\_a\_number; -- Prompt for input

BEGIN

IF num > 0 THEN

DBMS\_OUTPUT.PUT\_LINE('The number is positive.');

ELSIF num < 0 THEN

DBMS\_OUTPUT.PUT\_LINE('The number is negative.');

ELSE

DBMS\_OUTPUT.PUT\_LINE('The number is zero.');

END IF;

END;

/

OUTPUT :

The number is negative.

**Exercise 3 : Stored procedures :**

Students table :

CREATE TABLE students (

student\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

marks NUMBER

);

Sample data :

INSERT INTO students VALUES (1, 'Harshitha', 88);

INSERT INTO students VALUES (2, 'Rahul', 72);

INSERT INTO students VALUES (3, 'Sneha', 95);

COMMIT;

Stored procedure code :

CREATE OR REPLACE PROCEDURE get\_student\_details (

p\_id IN students.student\_id%TYPE,

p\_name OUT students.name%TYPE,

p\_marks OUT students.marks%TYPE

)

IS

BEGIN

SELECT name, marks

INTO p\_name, p\_marks

FROM students

WHERE student\_id = p\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

p\_name := 'Not Found';

p\_marks := 0;

END;

/

DECLARE

v\_name students.name%TYPE;

v\_marks students.marks%TYPE;

BEGIN

get\_student\_details(1, v\_name, v\_marks);

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_name);

DBMS\_OUTPUT.PUT\_LINE('Marks: ' || v\_marks);

END;

/

OUTPUT :

Name: Harshitha

Marks: 88

Skill : TDD using Junit5 and Mockito :

**Exercise 1 : Setting up Junit**

Pom.xml In Maven :

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>junit-demo</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- JUnit 5 API -->

<dependency>

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

<artifactId>junit-jupiter-api</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

<!-- JUnit 5 Engine -->

<dependency>

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

<artifactId>junit-jupiter-engine</artifactId>

<version>5.9.3</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Plugin to run tests -->

<plugin>

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

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

<version>2.22.2</version>

</plugin>

</plugins>

</build>

</project>

Calculator.java :

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

CalculatorTest.java :

package com.example;

import org.junit.jupiter.api.Test;

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

public class CalculatorTest {

@Test

void testAdd() {

Calculator calc = new Calculator();

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

assertEquals(8, result, "5 + 3 should equal 8");

}

}

OUTPUT :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

**Exercise 3 : Assertions in Junit :**

MathUtils.java :

package com.example;

public class MathUtils {

public int add(int a, int b) {

return a + b;

}

public boolean isEven(int num) {

return num % 2 == 0;

}

public int[] getArray() {

return new int[] {1, 2, 3};

}

public void divide(int a, int b) {

int result = a / b;

}

}

MathUtilsTest.java :

package com.example;

import org.junit.jupiter.api.Test;

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

public class MathUtilsTest {

MathUtils math = new MathUtils();

@Test

void testAddition() {

assertEquals(10, math.add(7, 3), "7 + 3 should be 10");

}

@Test

void testIsEven() {

assertTrue(math.isEven(4), "4 should be even");

assertFalse(math.isEven(5), "5 should not be even");

}

@Test

void testArrayEquality() {

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

assertArrayEquals(expected, math.getArray(), "Arrays should match");

}

@Test

void testException() {

Exception exception = assertThrows(ArithmeticException.class, () -> {

math.divide(10, 0);

});

assertEquals("/ by zero", exception.getMessage());

}

@Test

void testMultipleAssertions() {

assertAll("Grouped assertions",

() -> assertEquals(5, math.add(2, 3)),

() -> assertTrue(math.isEven(6)),

() -> assertArrayEquals(new int[]{1, 2, 3}, math.getArray())

);

}

}

OUTPUT :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0

**Exercise 4 : Arrange-Act-Assert(AAA) Pattern, test fixtures, Setup and Teardown Methods in Junit :**

BankAccount.java :

package com.example;

public class BankAccount {

private int balance;

public BankAccount(int initialBalance) {

this.balance = initialBalance;

}

public void deposit(int amount) {

balance += amount;

}

public void withdraw(int amount) {

if (amount > balance) {

throw new IllegalArgumentException("Insufficient balance");

}

balance -= amount;

}

public int getBalance() {

return balance;

}

}

BankAccountTest.java :

package com.example;

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

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

public class BankAccountTest {

BankAccount account;

@BeforeEach

void setUp() {

// Setup - runs before each test

account = new BankAccount(100);

}

@AfterEach

void tearDown() {

// Teardown - runs after each test

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

}

@Test

void testDeposit() {

// Arrange done in setUp()

// Act

account.deposit(50);

// Assert

assertEquals(150, account.getBalance(), "Balance should be 150 after deposit");

}

@Test

void testWithdrawValidAmount() {

// Act

account.withdraw(40);

// Assert

assertEquals(60, account.getBalance(), "Balance should be 60 after withdrawal");

}

@Test

void testWithdrawMoreThanBalance() {

// Act + Assert

Exception exception = assertThrows(IllegalArgumentException.class, () -> {

account.withdraw(200);

});

assertEquals("Insufficient balance", exception.getMessage());

}

}

OUTPUT :

Tests run: 3, Failures: 0, Errors: 0, Skipped: 0

Test completed.

Test completed.

Test completed.

**Exercise 1 : Mocking and Stubbing :**

User.java :

package com.example;

public class User {

private int id;

private String name;

// Constructor, Getters, Setters

public User(int id, String name) {

this.id = id;

this.name = name;

}

public int getId() { return id; }

public String getName() { return name; }

}

UserRepository.java :

package com.example;

public interface UserRepository {

User findUserById(int id);

}

UserService.java :

package com.example;

public class UserService {

private UserRepository repository;

public UserService(UserRepository repository) {

this.repository = repository;

}

public String getUserName(int userId) {

User user = repository.findUserById(userId);

return user != null ? user.getName() : "Unknown User";

}

}

UserServiceTest.java :

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

void testGetUserName() {

// Arrange - create mock and stub

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findUserById(1)).thenReturn(new User(1, "Harshitha"));

UserService service = new UserService(mockRepo);

// Act

String result = service.getUserName(1);

// Assert

assertEquals("Harshitha", result);

}

@Test

void testUserNotFound() {

UserRepository mockRepo = mock(UserRepository.class);

when(mockRepo.findUserById(2)).thenReturn(null); // Stub to return null

UserService service = new UserService(mockRepo);

String result = service.getUserName(2);

assertEquals("Unknown User", result);

}

}

OUTPUT :

Test Case 1: testGetUserName() :

when(mockRepo.findUserById(1)).thenReturn(new User(1, "Harshitha"));

Test passed — Output: "Harshitha"

Test Case 2: testUserNotFound() :

when(mockRepo.findUserById(2)).thenReturn(null);

Test passed — Output: "Unknown User"

Final output :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0

**Exercise 2 Verifying interactions :**

UserService.java :

package com.example;

public class UserService {

private UserRepository repository;

private EmailService emailService;

public UserService(UserRepository repository, EmailService emailService) {

this.repository = repository;

this.emailService = emailService;

}

public void registerUser(User user) {

repository.save(user);

emailService.sendWelcomeEmail(user.getEmail());

}

}

Supporting Classes :

package com.example;

public class User {

private String name;

private String email;

public User(String name, String email) {

this.name = name;

this.email = email;

}

public String getEmail() { return email; }

}

package com.example;

public interface UserRepository {

void save(User user);

}

public interface EmailService {

void sendWelcomeEmail(String email);

}

Test Class with Verification :

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class UserServiceTest {

@Test

void testRegisterUser\_VerifiesInteractions() {

// Arrange

UserRepository mockRepo = mock(UserRepository.class);

EmailService mockEmail = mock(EmailService.class);

UserService service = new UserService(mockRepo, mockEmail);

User user = new User("Harshitha", "harshitha@example.com");

// Act

service.registerUser(user);

// Assert — verify interactions

verify(mockRepo).save(user);

verify(mockEmail).sendWelcomeEmail("harshitha@example.com");

// Optionally verify how many times

verify(mockRepo, times(1)).save(user); // default is once

verify(mockEmail, never()).sendWelcomeEmail("wrong@example.com");

wrong email

}

}

OUTPUT :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0

**Exercise 1 : logging error messages and warning levels :**

Pom.xml :

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>2.0.9</version>

</dependency>

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.4.11</version>

</dependency>

</dependencies>

LoggingExample.java :

package com.example;

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.info("Application started");

try {

int a = 10 / 0; // Will throw exception

} catch (ArithmeticException e) {

logger.error("An error occurred: Division by zero", e);

}

String configStatus = "missing";

if ("missing".equals(configStatus)) {

logger.warn("Configuration file is missing");

}

logger.info("Application ended");

}

}

logback.xml :

<configuration>

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

<encoder>

<pattern>%d{HH:mm:ss} %-5level - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>  
OUTPUT :

12:00:01 INFO - Application started

12:00:01 ERROR - An error occurred: Division by zero

java.lang.ArithmeticException: / by zero

at com.example.LoggingExample.main(LoggingExample.java:10)

12:00:01 WARN - Configuration file is missing

12:00:01 INFO - Application ended