**Assessment**

1. Control Structures

// Conditional Statement: if-else

function checkVotingEligibility(age) {

if (age >= 18) {

console.log("✅ You are eligible to vote.");

} else {

console.log("❌ You are not eligible to vote yet.");

}

}

checkVotingEligibility(20); // Change the age to test

// Loop: for loop with continue

console.log("📄 Printing numbers from 1 to 10 (skipping 5):");

for (let i = 1; i <= 10; i++) {

if (i === 5) {

continue; // skip number 5

}

console.log(i);

}

// Switch-case

function getUserRole(role) {

switch (role) {

case "admin":

console.log("🔐 Access granted to admin panel.");

break;

case "user":

console.log("👤 Access granted to user dashboard.");

break;

case "guest":

console.log("🕵️ Limited access granted.");

break;

default:

console.log("❓ Unknown role.");

}

}

getUserRole("admin"); // Try changing to "user", "guest", "manager"

1. Stored Procedures

Use your database

Create Stored Procedure

CREATE PROCEDURE GetCustomerOrders(IN customerId INT)

BEGIN

SELECT \* FROM orders WHERE customer\_id = customerId;

END $$

DELIMITER ;

CALL GetCustomerOrders(101);

1. Setting Up JUnit

By using maven.

**First , create maven project**

**Add JUnit dependency to pom.xml:**

<dependencies>

<dependency>

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

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

</dependencies>

**Now create a test class,**

// Calculator.java

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**Run test in the terminal**

mvn test

1. **Assertions in JUnit**

**Test class using Assertions**

import org.junit.jupiter.api.Test;

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

class CalculatorTest {

Calculator calc = new Calculator();

@Test

void testAddition() {

assertEquals(7, calc.add(3, 4));

}

@Test

void testDivisionByZero() {

assertThrows(ArithmeticException.class, () -> calc.divide(10, 0));

}

@Test

void testMultipleAssertions() {

assertAll("Check multiple math operations",

() -> assertEquals(9, calc.multiply(3, 3)),

() -> assertNotEquals(10, calc.add(2, 2))

);

}

}

1. **Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit.**

**Arrange-Act-Assert (AAA) Pattern:**

A structured way of writing test methods that improves readability and clarity.

@Test

void testAddition() {

// Arrange

Calculator calc = new Calculator();

// Act

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

// Assert

assertEquals(5, result);

}

**Test Fixtures in JUnit:**

A text fixture is a fixed state of a set of objects used as a baseline for running tests.

* @BeforeEach → Runs before each test
* @AfterEach → Runs after each test
* @BeforeAll → Runs once before all tests (static method)
* @AfterAll → Runs once after all tests (static method)

**Setup and Teardown Methods:**

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

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

class CalculatorTest {

private Calculator calculator;

@BeforeAll

static void beforeAllTests() {

System.out.println("Starting Calculator Tests...");

}

@AfterAll

static void afterAllTests() {

System.out.println("All Calculator Tests Completed.");

}

@BeforeEach

void setUp() {

// Arrange: Test fixture setup

calculator = new Calculator();

System.out.println("Setup for a test.");

}

@AfterEach

void tearDown() {

calculator = null;

System.out.println("Teardown after a test.");

}

@Test

void testAdd() {

int result = calculator.add(10, 5);

assertEquals(15, result);

}

@Test

void testSubtract() {

int result = calculator.subtract(10, 5);

assertEquals(5, result);

}

}

1. **Mocking and Stubbing:**

**Defining classes**

// OrderRepository.java

public interface OrderRepository {

int getOrderCount();

}

// OrderService.java

public class OrderService {

private OrderRepository repository;

public OrderService(OrderRepository repository) {

this.repository = repository;

}

public boolean hasOrders() {

return repository.getOrderCount() > 0;

}

}

**Write test code with Mockito**

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

class OrderServiceTest {

@Test

void testHasOrders() {

// Arrange: Create a mock OrderRepository

OrderRepository mockRepo = mock(OrderRepository.class);

// Stubbing: Set behavior for getOrderCount()

when(mockRepo.getOrderCount()).thenReturn(5);

OrderService service = new OrderService(mockRepo);

// Act: Call method under test

boolean result = service.hasOrders();

// Assert: Check expected behavior

assertTrue(result);

// Verify: Check if getOrderCount() was called exactly once

verify(mockRepo, times(1)).getOrderCount();

}

}

1. Verifying Interactions

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

class EmailService {

public void sendEmail(String recipient) {

// Imagine actual email sending happens here

}

}

class NotificationService {

private EmailService emailService;

public NotificationService(EmailService emailService) {

this.emailService = emailService;

}

public void notifyUser(String userEmail) {

if (userEmail != null) {

emailService.sendEmail(userEmail);

}

}

}

class NotificationServiceTest {

@Test

void testNotifyUser\_SendsEmail() {

// Arrange

EmailService mockEmailService = mock(EmailService.class);

NotificationService notificationService = new NotificationService(mockEmailService);

// Act

notificationService.notifyUser("user@example.com");

// Assert: Verify interaction

verify(mockEmailService).sendEmail("user@example.com");

}

@Test

void testNotifyUser\_DoesNotSendEmailForNull() {

EmailService mockEmailService = mock(EmailService.class);

NotificationService notificationService = new NotificationService(mockEmailService);

notificationService.notifyUser(null);

verify(mockEmailService, never()).sendEmail(anyString());

}

}

1. **Logging Error Messages and Warning Levels**

**Maven Dependencies:**

<dependencies>

<!-- SLF4J API -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>2.0.9</version>

</dependency>

<!-- Logback (SLF4J implementation) -->

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.4.14</version>

</dependency>

</dependencies>

**Java Code 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.trace("This is a TRACE message");

logger.debug("This is a DEBUG message");

logger.info("This is an INFO message");

logger.warn("This is a WARN message");

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

try {

int result = 10 / 0;

} catch (ArithmeticException e) {

logger.error("Exception occurred: {}", e.getMessage(), e);

}

}

}