**WEEK 2**  
**Topic 1:** PL/SQL Programming

**Exercise 1**  
Control Structures

**Code :**  
  
DECLARE

v\_day NUMBER := 3;

BEGIN

CASE v\_day

WHEN 0 THEN DBMS\_OUTPUT.PUT\_LINE('Sunday');

WHEN 1 THEN DBMS\_OUTPUT.PUT\_LINE('Monday');

WHEN 2 THEN DBMS\_OUTPUT.PUT\_LINE('Tuesday');

WHEN 3 THEN DBMS\_OUTPUT.PUT\_LINE('Wednesday');

WHEN 4 THEN DBMS\_OUTPUT.PUT\_LINE('Thursday');

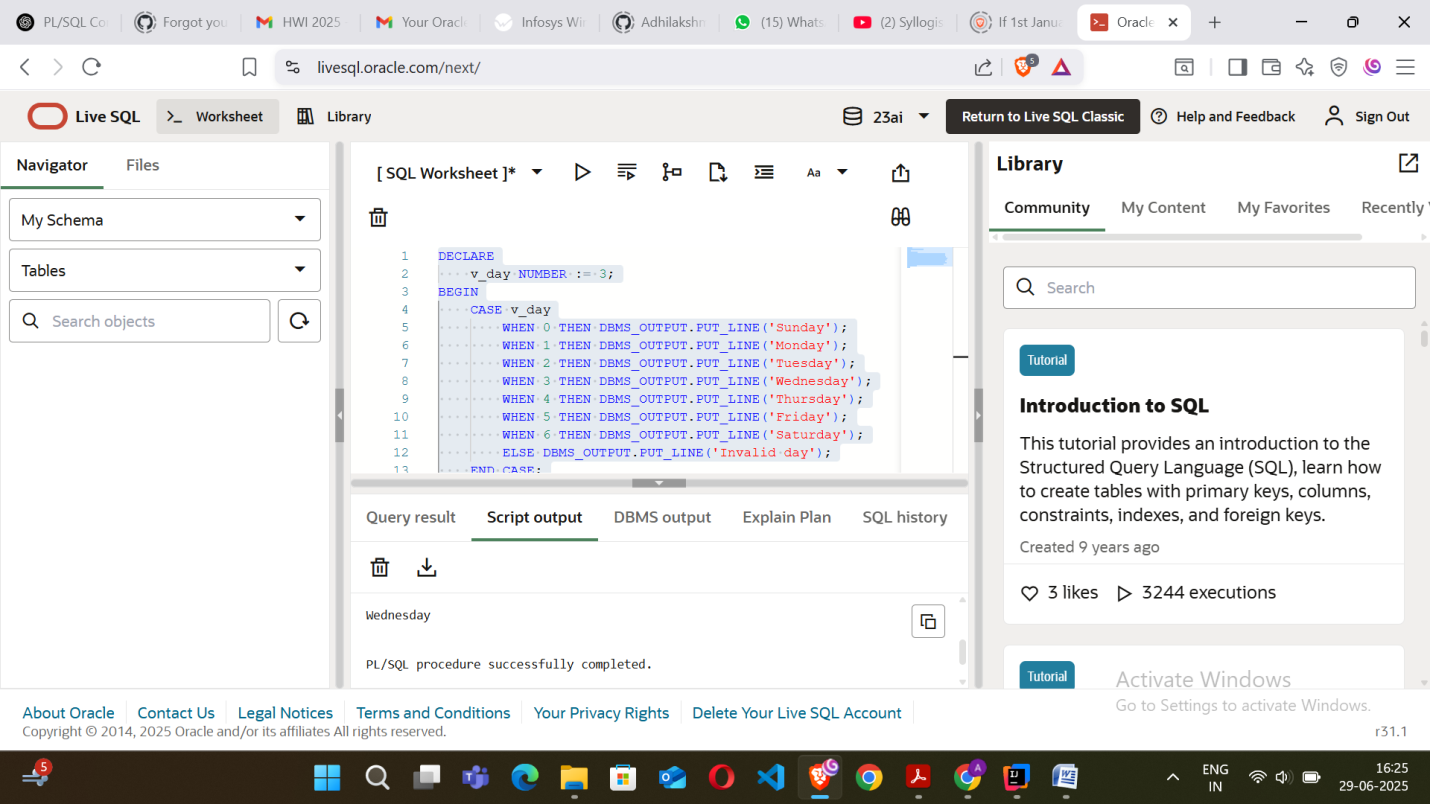
WHEN 5 THEN DBMS\_OUTPUT.PUT\_LINE('Friday');

WHEN 6 THEN DBMS\_OUTPUT.PUT\_LINE('Saturday');

ELSE DBMS\_OUTPUT.PUT\_LINE('Invalid day');

END CASE;

END;

**O/P :**   
  


**Exercise 2:**   
Stored Procedure  
**Code :**  
  
CREATE OR REPLACE PROCEDURE check\_even\_odd(p\_number IN NUMBER) IS

BEGIN

IF MOD(p\_number, 2) = 0 THEN

DBMS\_OUTPUT.PUT\_LINE(p\_number || ' is Even');

ELSE

DBMS\_OUTPUT.PUT\_LINE(p\_number || ' is Odd');

END IF;

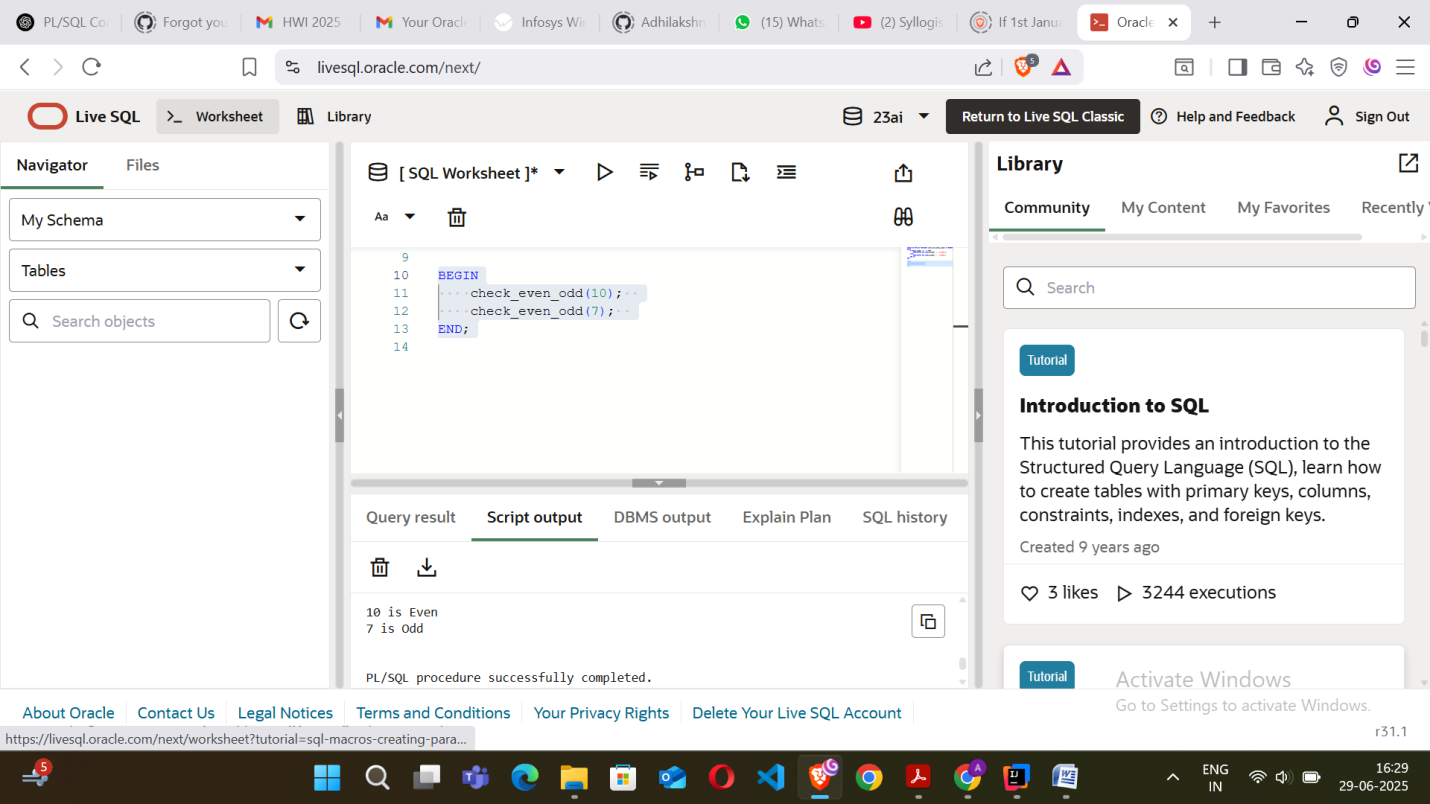
END;

BEGIN

check\_even\_odd(10);

check\_even\_odd(7);

END;

**O/P:**  
  


**Topic 2:** TDD using JUnit5 & Mockito,SLF4J Logging Framework

**Exercise 1:**

Setting Up JUnit  
  
**Code :**  
  
// BankAccount.java

package Ex1;

public class BankAccount {

private int balance = 0;

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;

}

public static void main(String[] args) {

BankAccount account = new BankAccount();

System.out.println("Initial Balance: " + account.getBalance());

account.deposit(1000);

System.out.println("After depositing 1000: " + account.getBalance());

account.withdraw(400);

System.out.println("After withdrawing 400: " + account.getBalance());

try {

account.withdraw(700);

} catch (IllegalArgumentException e) {

System.out.println("Exception: " + e.getMessage());

}

System.out.println("Final Balance: " + account.getBalance());

}

}  
  
//BankAccountTest.java

package Ex1;

import org.junit.jupiter.api.Test;

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

public class BankAccountTest {

@Test

void testDeposit() {

BankAccount account = new BankAccount();

account.deposit(1000);

assertEquals(1000, account.getBalance());

}

@Test

void testWithdraw() {

BankAccount account = new BankAccount();

account.deposit(1000);

account.withdraw(400);

assertEquals(600, account.getBalance());

}

@Test

void testWithdrawInsufficientBalance() {

BankAccount account = new BankAccount();

account.deposit(500);

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

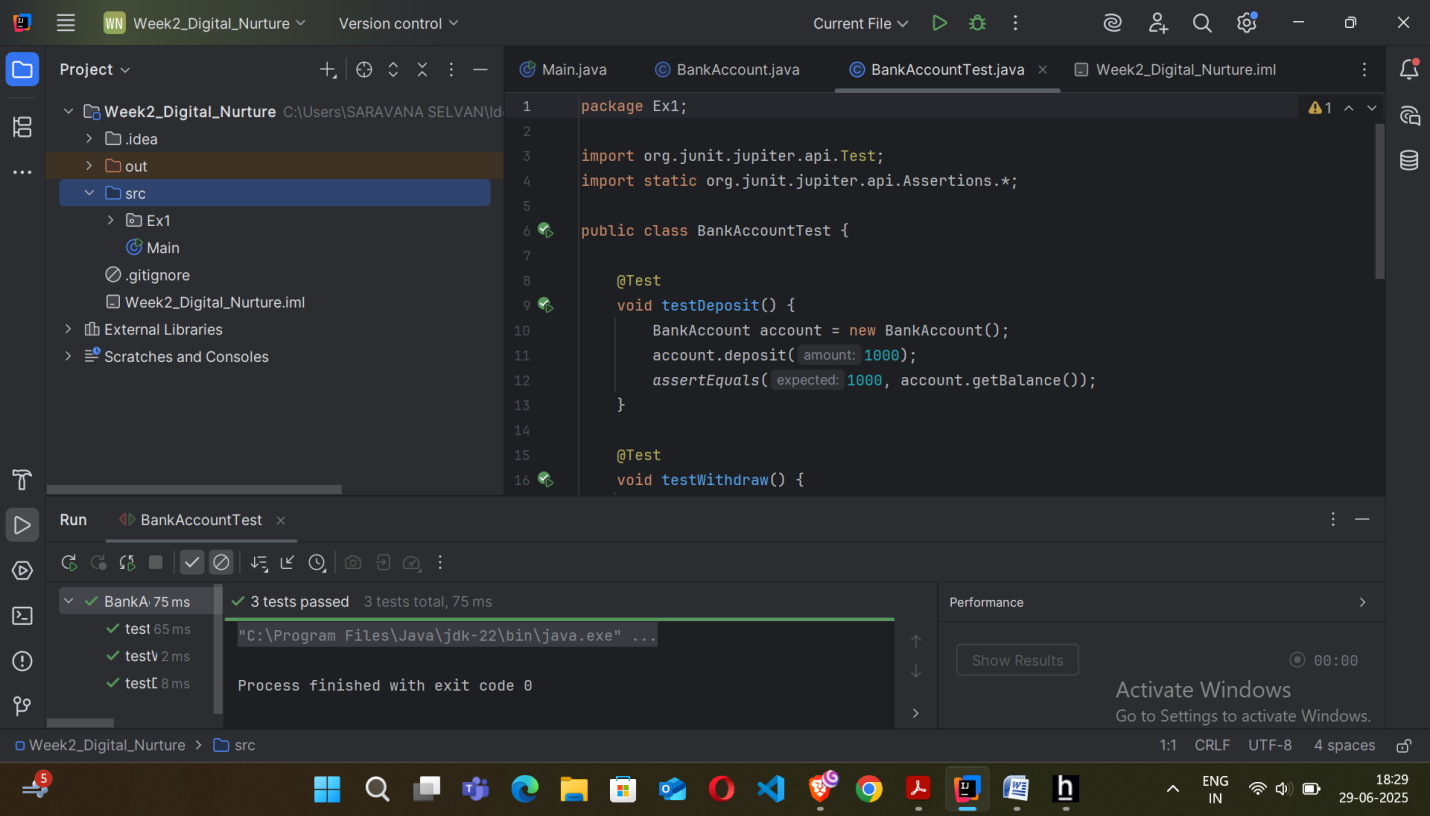
account.withdraw(600);

});

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

}

}

**O/P:**  
  
  
**Exercise 2:**

Assertions in JUnit  
  
**Code :**   
  
//LoginValidator.java

package Ex2;

public class LoginValidator {

public boolean isValidUser(String username, String password) {

return "admin".equals(username) && "1234".equals(password);

}

public boolean isStrongPassword(String password) {

return password != null && password.length() >= 4;

}

}

//LoginValidatorTest.java

package Ex2;

import org.junit.jupiter.api.Test;

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

public class LoginValidatorTest {

LoginValidator validator = new LoginValidator();

@Test

void testValidUserCredentials() {

assertTrue(validator.isValidUser("admin", "1234"), "Valid credentials should return true");

}

@Test

void testInvalidUserCredentials() {

assertFalse(validator.isValidUser("user", "pass"), "Invalid credentials should return false");

}

@Test

void testStrongPassword() {

assertTrue(validator.isStrongPassword("abcd"), "Password with 4+ characters should be strong");

}

@Test

void testWeakPassword() {

assertFalse(validator.isStrongPassword("abc"), "Password with less than 4 characters should be weak");

}

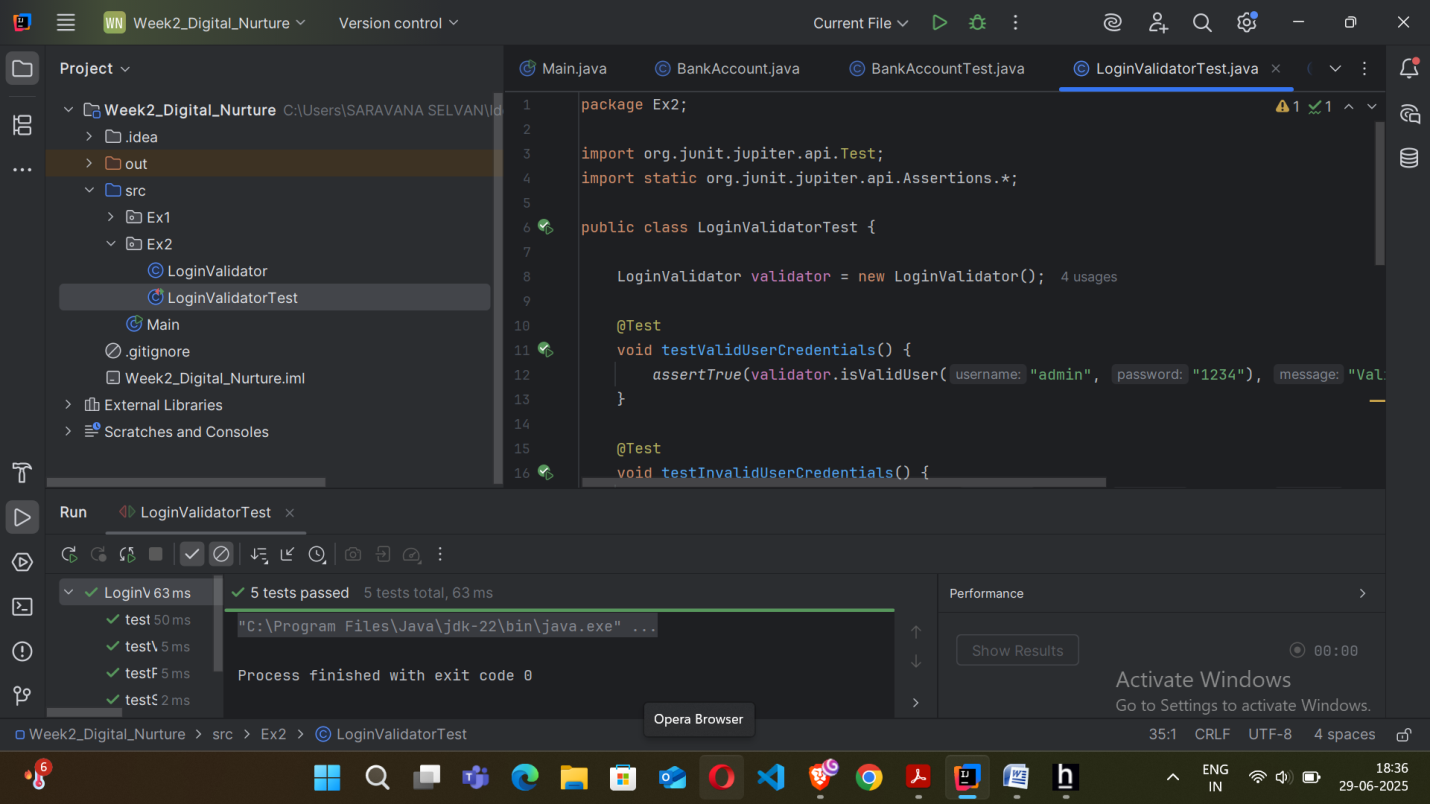
@Test

void testPasswordNotNull() {

assertNotNull("pass", "Password should not be null");

}

}

**O/P:**  
  
  
  
**Exercise 3:**

Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit  
  
**Code :**   
  
//Calculator.java

package Ex3;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

//CalculatorTest.java

package Ex3;

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

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

public class CalculatorTest {

private Calculator calculator;

@BeforeEach

void setUp() {

calculator = new Calculator(); // Fixture setup

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

}

@AfterEach

void tearDown() {

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

}

@Test

void testAddition() {

// Arrange

int a = 5;

int b = 3;

// Act

int result = calculator.add(a, b);

// Assert

assertEquals(8, result);

}

@Test

void testSubtraction() {

// Arrange

int a = 10;

int b = 4;

// Act

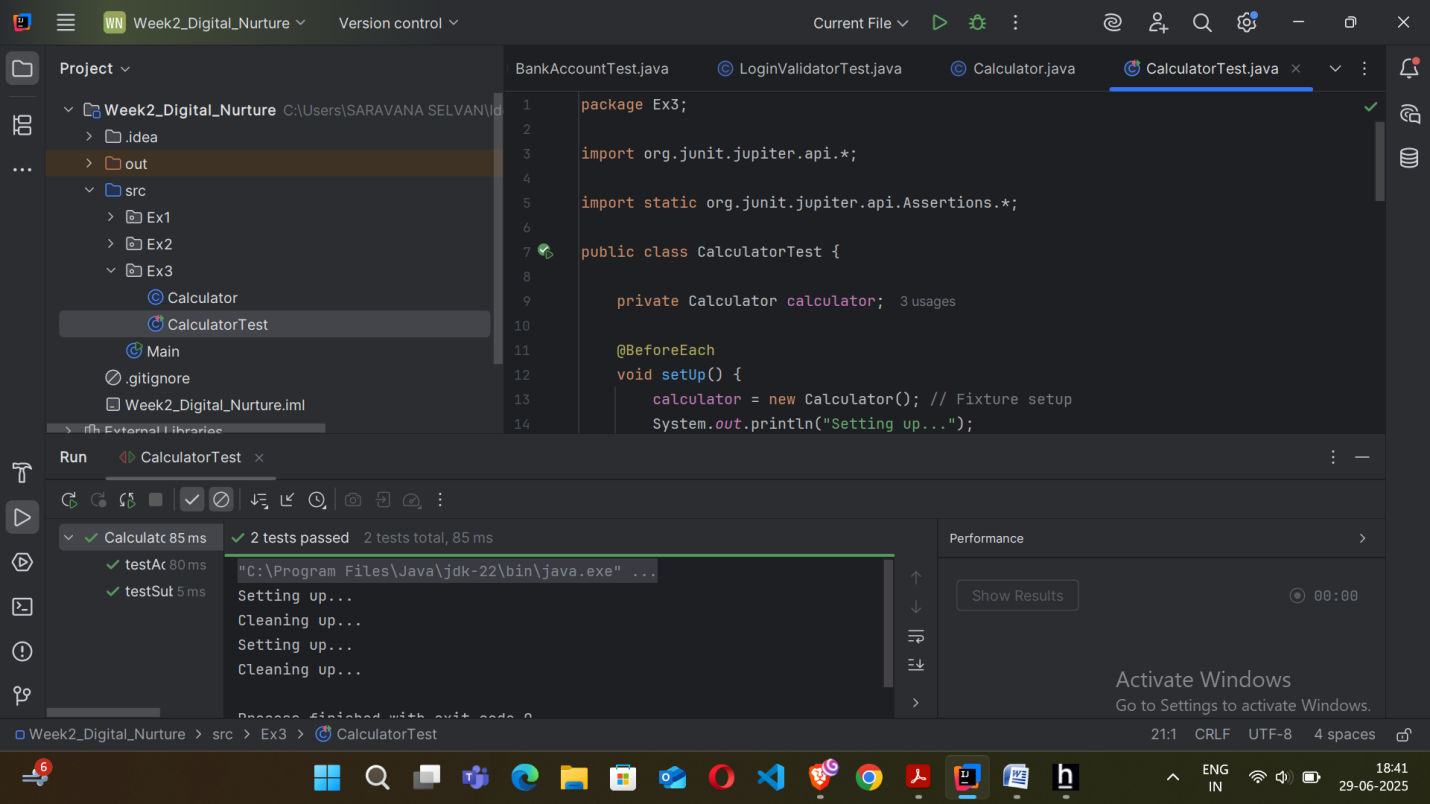
int result = calculator.subtract(a, b);

// Assert

assertEquals(6, result);

}

}

**O/P:**  
  


**Exercise 4:**

Mocking and Stubbing  
  
**Code :**

//UserService.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testRegisterUser\_MockingAndStubbing() {

EmailService emailService = mock(EmailService.class);

when(emailService.sendEmail("john@example.com", "Welcome!")).thenReturn(true);

UserService userService = new UserService(emailService);

boolean result = userService.registerUser("john@example.com");

assertTrue(result);

}

}  
  
//EmailService.java  
  
package com.example.ex2;

public interface EmailService {

boolean sendEmail(String to, String content);

}

//UserServiceTest.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testRegisterUser\_MockingAndStubbing() {

EmailService emailService = mock(EmailService.class);

when(emailService.sendEmail("john@example.com", "Welcome!")).thenReturn(true);

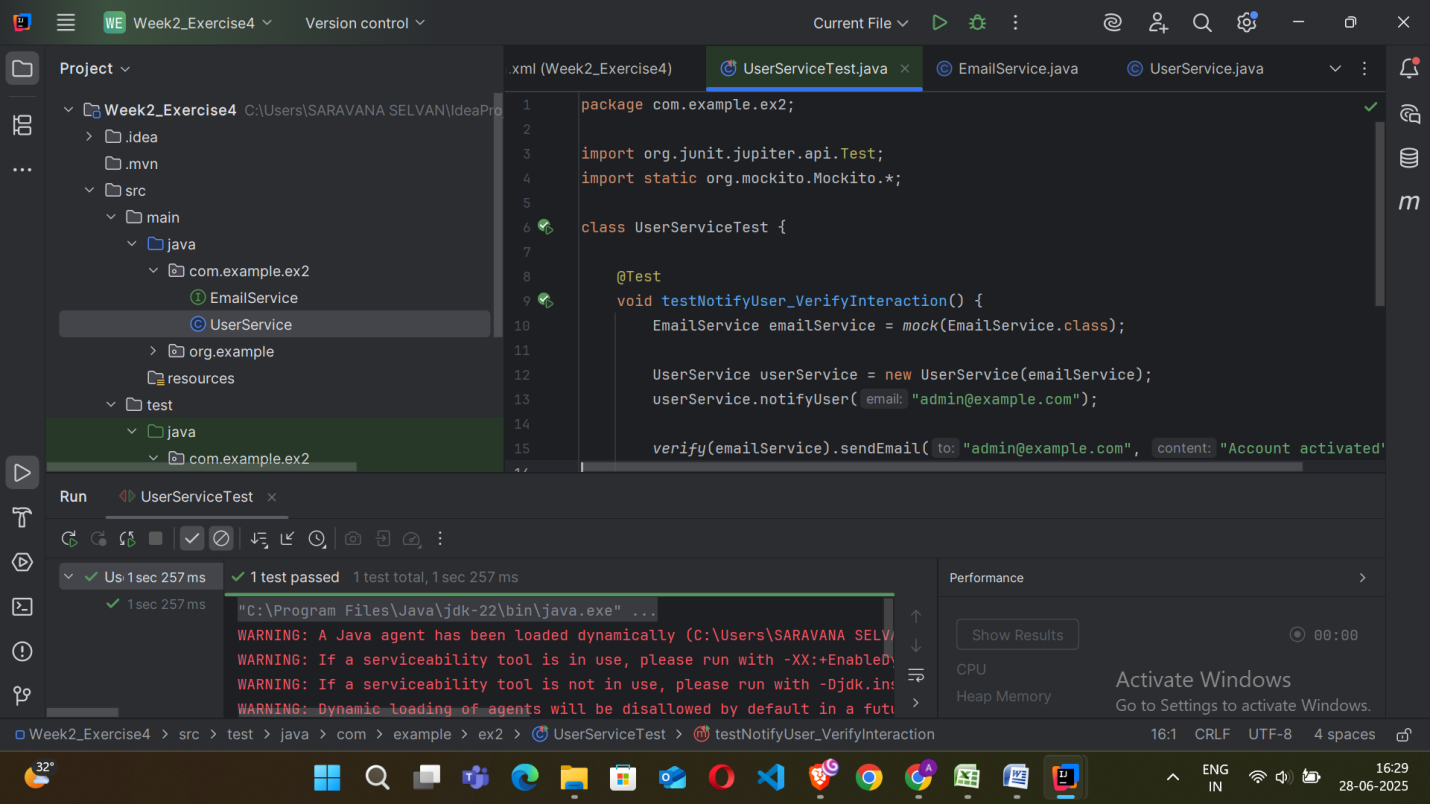
UserService userService = new UserService(emailService);

boolean result = userService.registerUser("john@example.com");

assertTrue(result);

}

}

**O/P:**  
  


**Exercise 5:** Verifying Interactions  
  
**Code :**   
  
//UserService.java  
  
package com.example.ex2;

public class UserService {

private final EmailService emailService;

public UserService(EmailService emailService) {

this.emailService = emailService;

}

public void notifyUser(String email) {

emailService.sendEmail(email, "Account activated");

}

}  
  
//EmailService.java  
  
package com.example.ex2;

public interface EmailService {

boolean sendEmail(String to, String content);

}  
  
//UserServiceTest.java  
  
package com.example.ex2;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

class UserServiceTest {

@Test

void testNotifyUser\_VerifyInteraction() {

EmailService emailService = mock(EmailService.class);

UserService userService = new UserService(emailService);

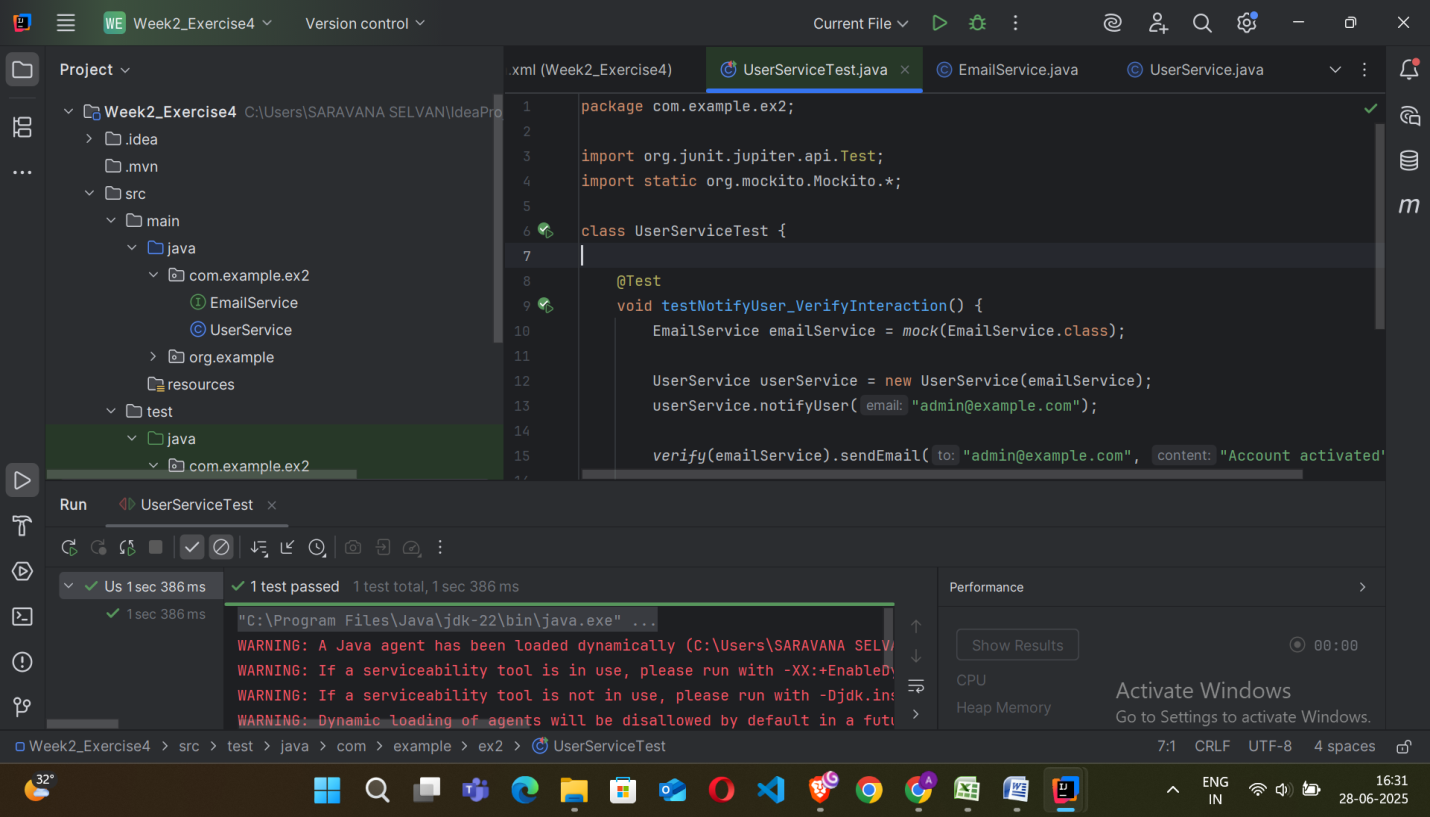
userService.notifyUser("admin@example.com");

verify(emailService).sendEmail("admin@example.com", "Account activated");

verifyNoMoreInteractions(emailService);

}

}

**O/P:**  
  


**Exercise 6:**

Logging Error Messages and Warning Levels  
  
**Code :**   
  
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingDemo {

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

public static void main(String[] args) {

logger.info("Application started.");

int speed = 120;

if (speed > 100) {

logger.warn("Over-speeding! Current speed: {} km/h", speed);

}

try {

int x = 10 / 0;

} catch (Exception e) {

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

}

logger.info("Application ended.");

}

}

**O/P:**  
  
