SQL Exercises

1: Control Structures

USE bank\_db;

DROP PROCEDURE IF EXISTS SendLoanReminders;

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_loan\_id INT;

DECLARE v\_customer\_name VARCHAR(100);

DECLARE v\_customer\_id INT;

DECLARE v\_loan\_amount DECIMAL(10, 2);

DECLARE v\_due\_date DATE;

DECLARE v\_days\_left INT;

DECLARE cur\_loans\_due CURSOR FOR

SELECT

l.LOAN\_ID,

c.CUSTOMER\_NAME,

c.CUSTOMER\_ID,

l.LOAN\_AMOUNT,

l.DUE\_DATE

FROM LOANS l

JOIN CUSTOMERS c ON l.CUSTOMER\_ID = c.CUSTOMER\_ID

WHERE l.DUE\_DATE BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY

ORDER BY l.DUE\_DATE ASC;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_loans\_due;

loan\_loop: LOOP

FETCH cur\_loans\_due INTO v\_loan\_id, v\_customer\_name, v\_customer\_id, v\_loan\_amount, v\_due\_date;

IF done THEN

LEAVE loan\_loop;

END IF;

SET v\_days\_left = DATEDIFF(v\_due\_date, CURDATE());

SELECT CONCAT('REMINDER: Loan ID ', v\_loan\_id, ' for ', v\_customer\_name,

' (Customer ID: ', v\_customer\_id, ')') AS ReminderHeader;

SELECT CONCAT(' Amount: $', FORMAT(v\_loan\_amount, 2),

', Due Date: ', DATE\_FORMAT(v\_due\_date, '%d-%b-%Y'),

' (', v\_days\_left, ' days left)') AS ReminderDetails;

SELECT '' AS Spacer;

END LOOP loan\_loop;

CLOSE cur\_loans\_due;

SELECT 'Loan reminder process complete.' AS Message;

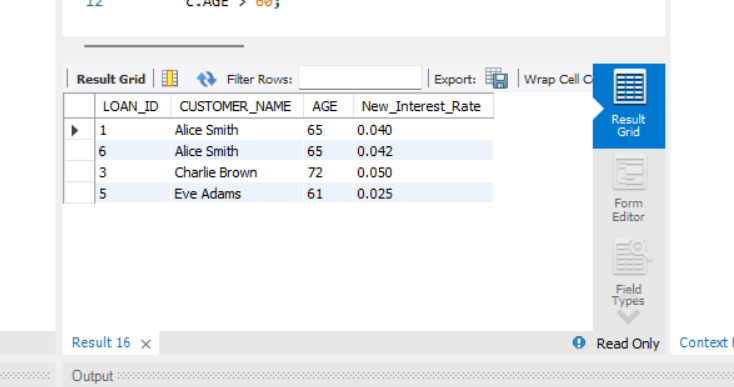
END;

//

DELIMITER ;

CALL SendLoanReminders();

Output



USE bank\_db;

DROP PROCEDURE IF EXISTS PromoteToVIP;

DELIMITER //

CREATE PROCEDURE PromoteToVIP()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_customer\_id INT;

DECLARE v\_customer\_name VARCHAR(100);

DECLARE v\_balance DECIMAL(10,2);

DECLARE v\_is\_vip ENUM('YES', 'NO');

DECLARE cur\_customers CURSOR FOR

SELECT CUSTOMER\_ID, CUSTOMER\_NAME, BALANCE, IS\_VIP

FROM CUSTOMERS

WHERE BALANCE > 10000

FOR UPDATE;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_customers;

customer\_loop: LOOP

FETCH cur\_customers INTO v\_customer\_id, v\_customer\_name, v\_balance, v\_is\_vip;

IF done THEN

LEAVE customer\_loop;

END IF;

IF v\_is\_vip = 'NO' THEN

UPDATE CUSTOMERS

SET IS\_VIP = 'YES'

WHERE CUSTOMER\_ID = v\_customer\_id;

SELECT CONCAT('Promoted: ', v\_customer\_name, ' (ID: ', v\_customer\_id, ') - Balance: $', FORMAT(v\_balance, 2)) AS Status;

ELSE

SELECT CONCAT('Already VIP: ', v\_customer\_name, ' (ID: ', v\_customer\_id, ')') AS Status;

END IF;

END LOOP customer\_loop;

CLOSE cur\_customers;

SELECT 'VIP promotion process complete.' AS Message;

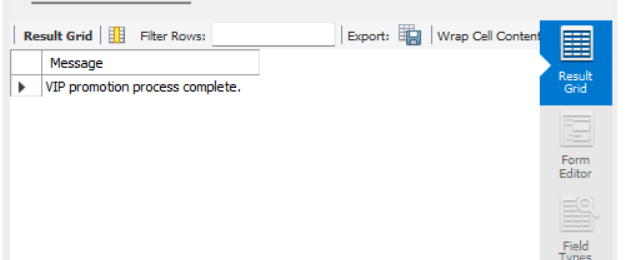
END;

//

DELIMITER ;

CALL PromoteToVIP();

Output



USE bank\_db;

DROP PROCEDURE IF EXISTS SendLoanReminders;

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_loan\_id INT;

DECLARE v\_customer\_name VARCHAR(100);

DECLARE v\_customer\_id INT;

DECLARE v\_loan\_amount DECIMAL(10, 2);

DECLARE v\_due\_date DATE;

DECLARE v\_days\_left INT;

DECLARE cur\_loans\_due CURSOR FOR

SELECT

l.LOAN\_ID,

c.CUSTOMER\_NAME,

c.CUSTOMER\_ID,

l.LOAN\_AMOUNT,

l.DUE\_DATE

FROM LOANS l

JOIN CUSTOMERS c ON l.CUSTOMER\_ID = c.CUSTOMER\_ID

WHERE l.DUE\_DATE BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY

ORDER BY l.DUE\_DATE ASC;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur\_loans\_due;

loan\_loop: LOOP

FETCH cur\_loans\_due INTO v\_loan\_id, v\_customer\_name, v\_customer\_id, v\_loan\_amount, v\_due\_date;

IF done THEN

LEAVE loan\_loop;

END IF;

SET v\_days\_left = DATEDIFF(v\_due\_date, CURDATE());

SELECT CONCAT('REMINDER: Loan ID ', v\_loan\_id, ' for ', v\_customer\_name,

' (Customer ID: ', v\_customer\_id, ')') AS ReminderHeader;

SELECT CONCAT(' Amount: $', FORMAT(v\_loan\_amount, 2),

', Due Date: ', DATE\_FORMAT(v\_due\_date, '%d-%b-%Y'),

' (', v\_days\_left, ' days left)') AS ReminderDetails;

SELECT '' AS Spacer;

END LOOP loan\_loop;

CLOSE cur\_loans\_due;

SELECT 'Loan reminder process complete.' AS Message;

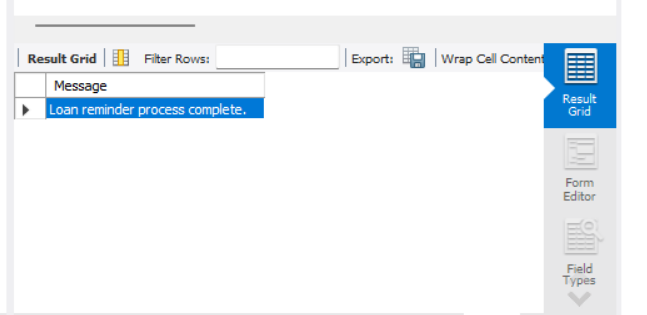
END;

//

DELIMITER ;

CALL SendLoanReminders();

Output



2)Stored Procedures

USE bank\_db;

DROP TABLE IF EXISTS ACCOUNTS;

DROP TABLE IF EXISTS EMPLOYEES;

CREATE TABLE ACCOUNTS (

ACCOUNT\_ID INT PRIMARY KEY,

ACCOUNT\_NUMBER VARCHAR(20) UNIQUE NOT NULL,

ACCOUNT\_TYPE ENUM('Savings', 'Checking') NOT NULL,

BALANCE DECIMAL(15, 2) NOT NULL DEFAULT 0.00

);

CREATE TABLE EMPLOYEES (

EMPLOYEE\_ID INT PRIMARY KEY,

EMPLOYEE\_NAME VARCHAR(100) NOT NULL,

DEPARTMENT VARCHAR(50) NOT NULL,

SALARY DECIMAL(10, 2) NOT NULL DEFAULT 0.00

);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_NUMBER, ACCOUNT\_TYPE, BALANCE) VALUES

(1, 'S001', 'Savings', 5000.00),

(2, 'S002', 'Savings', 1200.50),

(3, 'S003', 'Savings', 9800.75),

(4, 'C001', 'Checking', 1500.00),

(5, 'C002', 'Checking', 200.00),

(6, 'S004', 'Savings', 25000.00);

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, DEPARTMENT, SALARY) VALUES

(1, 'John Doe', 'Sales', 50000.00),

(2, 'Jane Smith', 'Sales', 60000.00),

(3, 'Peter Jones', 'Marketing', 45000.00),

(4, 'Mary Lee', 'IT', 70000.00),

(5, 'David Kim', 'Marketing', 55000.00);

USE bank\_db;

DROP PROCEDURE IF EXISTS ProcessMonthlyInterest;

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_account\_id INT;

DECLARE v\_account\_number VARCHAR(20);

DECLARE v\_balance DECIMAL(15, 2);

DECLARE v\_interest\_rate DECIMAL(5, 3) DEFAULT 0.01; -- 1% monthly interest

DECLARE v\_new\_balance DECIMAL(15, 2);

DECLARE cur\_savings\_accounts CURSOR FOR

SELECT ACCOUNT\_ID, ACCOUNT\_NUMBER, BALANCE

FROM ACCOUNTS

WHERE ACCOUNT\_TYPE = 'Savings'

FOR UPDATE;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

SELECT 'Processing monthly interest for savings accounts...' AS Message;

SELECT '---------------------------------------------------' AS Message;

OPEN cur\_savings\_accounts;

account\_loop: LOOP

FETCH cur\_savings\_accounts INTO v\_account\_id, v\_account\_number, v\_balance;

IF done THEN

LEAVE account\_loop;

END IF;

SET v\_new\_balance = v\_balance \* (1 + v\_interest\_rate);

START TRANSACTION;

UPDATE ACCOUNTS

SET BALANCE = v\_new\_balance

WHERE ACCOUNT\_ID = v\_account\_id;

COMMIT;

SELECT CONCAT('Account: ', v\_account\_number, ' (ID: ', v\_account\_id, ')') AS AccountInfo;

SELECT CONCAT(' Old Balance: $', FORMAT(v\_balance, 2),

', New Balance: $', FORMAT(v\_new\_balance, 2)) AS BalanceInfo;

SELECT '' AS Spacer;

END LOOP account\_loop;

CLOSE cur\_savings\_accounts;

SELECT 'Monthly interest processing complete.' AS Message;

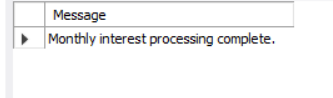
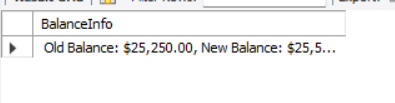
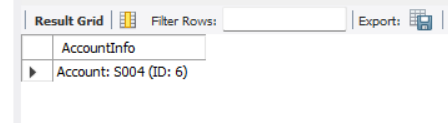
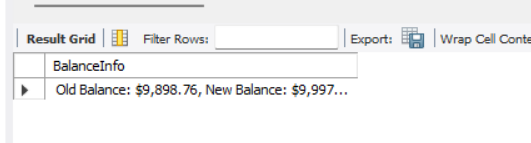
END //

DELIMITER ;

USE bank\_db;

CALL ProcessMonthlyInterest();

Output



USE bank\_db;

DROP PROCEDURE IF EXISTS UpdateEmployeeBonus;

DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus(

IN p\_department\_name VARCHAR(50),

IN p\_bonus\_percentage DECIMAL(5, 3)

)

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_employee\_id INT;

DECLARE v\_employee\_name VARCHAR(100);

DECLARE v\_current\_salary DECIMAL(10, 2);

DECLARE v\_new\_salary DECIMAL(10, 2);

DECLARE cur\_employees CURSOR FOR

SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY

FROM EMPLOYEES

WHERE DEPARTMENT = p\_department\_name

FOR UPDATE;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

SELECT CONCAT('Applying ', FORMAT(p\_bonus\_percentage \* 100, 1), '% bonus to employees in ', p\_department\_name, ' department...') AS Message;

SELECT '----------------------------------------------------------------------' AS Message;

OPEN cur\_employees;

employee\_loop: LOOP

FETCH cur\_employees INTO v\_employee\_id, v\_employee\_name, v\_current\_salary;

IF done THEN

LEAVE employee\_loop;

END IF;

SET v\_new\_salary = v\_current\_salary \* (1 + p\_bonus\_percentage);

START TRANSACTION;

UPDATE EMPLOYEES

SET SALARY = v\_new\_salary

WHERE EMPLOYEE\_ID = v\_employee\_id;

COMMIT;

SELECT CONCAT('Employee: ', v\_employee\_name, ' (ID: ', v\_employee\_id, ')') AS EmployeeInfo;

SELECT CONCAT(' Old Salary: $', FORMAT(v\_current\_salary, 2),

', New Salary: $', FORMAT(v\_new\_salary, 2)) AS SalaryInfo;

SELECT '' AS Spacer;

END LOOP employee\_loop;

CLOSE cur\_employees;

SELECT CONCAT('Bonus update for ', p\_department\_name, ' department complete.') AS Message;

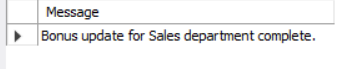
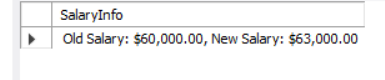
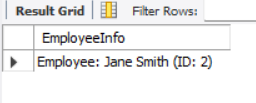
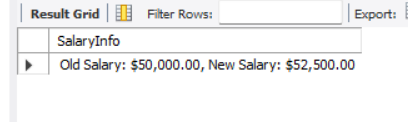
END //

DELIMITER ;

USE bank\_db;

CALL UpdateEmployeeBonus('Sales', 0.05); -- 5% bonus for Sales

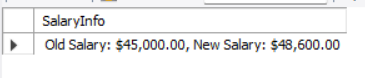
Output

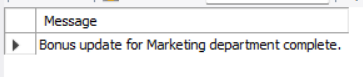


USE bank\_db;

CALL UpdateEmployeeBonus('Marketing', 0.08); -- 8% bonus for Marketing

Output





USE bank\_db;

DROP PROCEDURE IF EXISTS TransferFunds;

DELIMITER //

CREATE PROCEDURE TransferFunds(

IN p\_source\_account\_id INT,

IN p\_destination\_account\_id INT,

IN p\_amount DECIMAL(15, 2)

)

BEGIN

DECLARE v\_source\_balance DECIMAL(15, 2);

DECLARE v\_source\_account\_number VARCHAR(20);

DECLARE v\_destination\_account\_number VARCHAR(20);

SELECT BALANCE, ACCOUNT\_NUMBER INTO v\_source\_balance, v\_source\_account\_number

FROM ACCOUNTS

WHERE ACCOUNT\_ID = p\_source\_account\_id;

SELECT ACCOUNT\_NUMBER INTO v\_destination\_account\_number

FROM ACCOUNTS

WHERE ACCOUNT\_ID = p\_destination\_account\_id;

IF v\_source\_account\_number IS NULL THEN

SELECT CONCAT('Error: Source account with ID ', p\_source\_account\_id, ' not found.') AS Message;

ELSEIF v\_destination\_account\_number IS NULL THEN

SELECT CONCAT('Error: Destination account with ID ', p\_destination\_account\_id, ' not found.') AS Message;

ELSEIF p\_amount <= 0 THEN

SELECT 'Error: Transfer amount must be positive.' AS Message;

ELSEIF v\_source\_balance < p\_amount THEN

SELECT CONCAT('Transfer failed from ', v\_source\_account\_number, ': Insufficient balance (Available: $', FORMAT(v\_source\_balance, 2), ', Requested: $', FORMAT(p\_amount, 2), ').') AS Message;

ELSE

START TRANSACTION;

UPDATE ACCOUNTS

SET BALANCE = BALANCE - p\_amount

WHERE ACCOUNT\_ID = p\_source\_account\_id;

UPDATE ACCOUNTS

SET BALANCE = BALANCE + p\_amount

WHERE ACCOUNT\_ID = p\_destination\_account\_id;

COMMIT;

SELECT CONCAT('Funds transfer successful: $', FORMAT(p\_amount, 2), ' transferred from ', v\_source\_account\_number, ' to ', v\_destination\_account\_number, '.') AS Message;

SELECT CONCAT('New balance for ', v\_source\_account\_number, ': $', FORMAT(v\_source\_balance - p\_amount, 2)) AS SourceBalance;

SELECT CONCAT('New balance for ', v\_destination\_account\_number, ': $', FORMAT((SELECT BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID = p\_destination\_account\_id), 2)) AS DestBalance;

END IF;

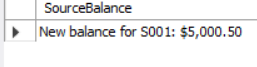
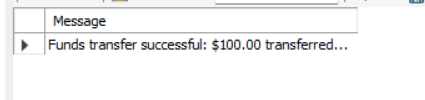
END //

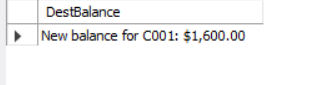
DELIMITER ;

USE bank\_db;

CALL TransferFunds(1, 4, 100.00); -- Transfer $100 from S001 to C001

Output





3) Setting Up Junit :

PasswordValidator.java

package com.jothees;

import java.util.Scanner;

public class PasswordValidator {

public boolean isValid(String password) {

if (password == null) return false;

return password.length() >= 8 &&

password.matches(".\*[A-Z].\*") &&

password.matches(".\*[a-z].\*") &&

password.matches(".\*\\d.\*") &&

password.matches(".\*[!@#$%^&\*].\*");

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the password : ");

String input = scanner.nextLine();

PasswordValidator validator = new PasswordValidator();

if (validator.isValid(input)) {

System.out.println("Password is valid ");

} else {

System.out.println("Password is invalid ");

}

scanner.close();

}

}

PasswordValidatorTest.java

package com.jothees;

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

public void testValidPassword() {

PasswordValidator validator = new PasswordValidator();

assertTrue(validator.isValid("Abcd@123"));

}

public void testShortPassword() {

PasswordValidator validator = new PasswordValidator();

assertFalse(validator.isValid("Ab@1"));

}

public void testMissingUppercase() {

PasswordValidator validator = new PasswordValidator();

assertFalse(validator.isValid("abcd@123")); }

public void testMissingDigit() {

PasswordValidator validator = new PasswordValidator();

assertFalse(validator.isValid("Abcd@xyz"));

}

public void testMissingSpecialChar() {

PasswordValidator validator = new PasswordValidator();

assertFalse(validator.isValid("Abcd1234"));

}

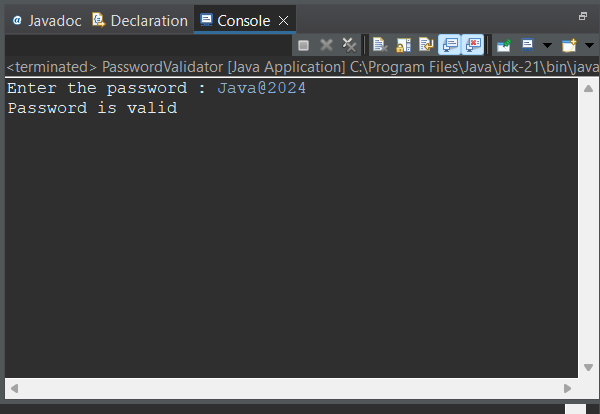
public void testNullPassword() {

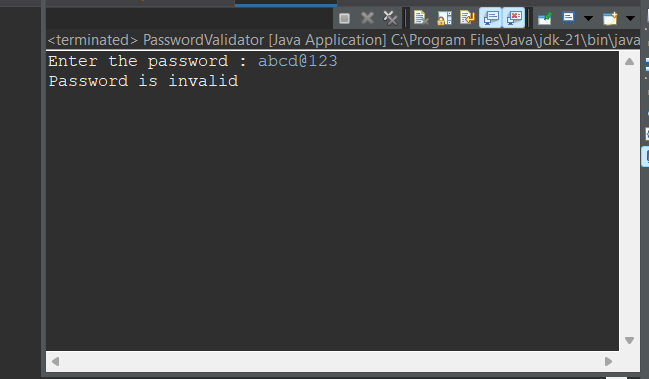
PasswordValidator validator = new PasswordValidator();

assertFalse(validator.isValid(null));

}

}





4) Assertions in Junit :

Main.java

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Calculator calc = new Calculator();

System.out.print("Enter first number: ");

int a = sc.nextInt();

System.out.print("Enter second number: ");

int b = sc.nextInt();

System.out.println("Addition: " + calc.add(a, b));

System.out.println("Subtraction: " + calc.subtract(a, b));

System.out.println("Multiplication: " + calc.multiply(a, b));

System.out.println("Division: " + (b != 0 ? calc.divide(a, b) : "Cannot divide by zero"));

}

}

Calculator.java

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 int divide(int a, int b) {

if (b == 0) throw new ArithmeticException("Cannot divide by zero");

return a / b;

}

}

CalculatorTest.java

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class MyFirstTest {

private Calculator calc;

*@Before*

public void setUp() {

calc = new Calculator();

}

public void testAddition() {

*assertEquals*(15, calc.add(10, 5));

}

public void testSubtraction() {

*assertEquals*(5, calc.subtract(10, 5));

}

public void testMultiplication() {

*assertEquals*(50, calc.multiply(10, 5));

}

public void testDivision() {

*assertEquals*(2, calc.divide(10, 5));

}

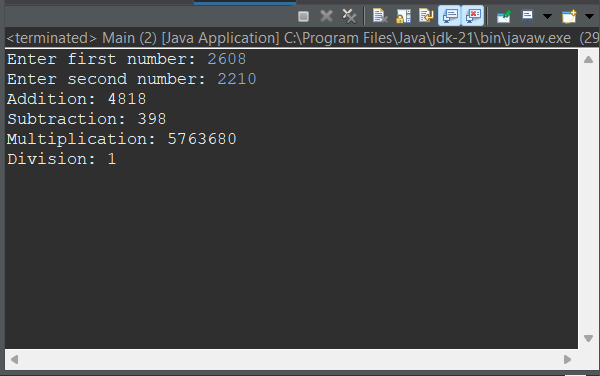
*@Test*(expected = ArithmeticException.class)

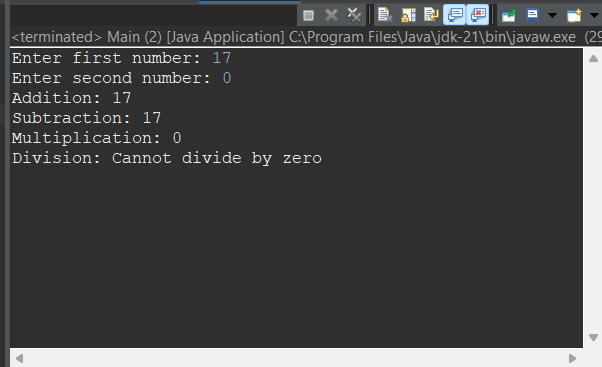
public void testDivisionByZero() {

calc.divide(10, 0);

}

}





5) Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit :

PasswordValidator.java

package com.jothees;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class PasswordValidatorTest {

private PasswordValidator validator;

@Before

public void setUp() {

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

validator = new PasswordValidator();

}

@After

public void tearDown() {

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

validator = null;

}

@Test

public void testValidPassword() {

boolean result = validator.isValid("Abcd@123");

assertTrue(result);

}

@Test

public void testShortPassword() {

boolean result = validator.isValid("Ab@1");

assertFalse(result);

}

@Test

public void testMissingUppercase() {

boolean result = validator.isValid("abcd@123");

assertFalse(result);

}

@Test

public void testMissingDigit() {

boolean result = validator.isValid("Abcd@xyz");

assertFalse(result);

}

@Test

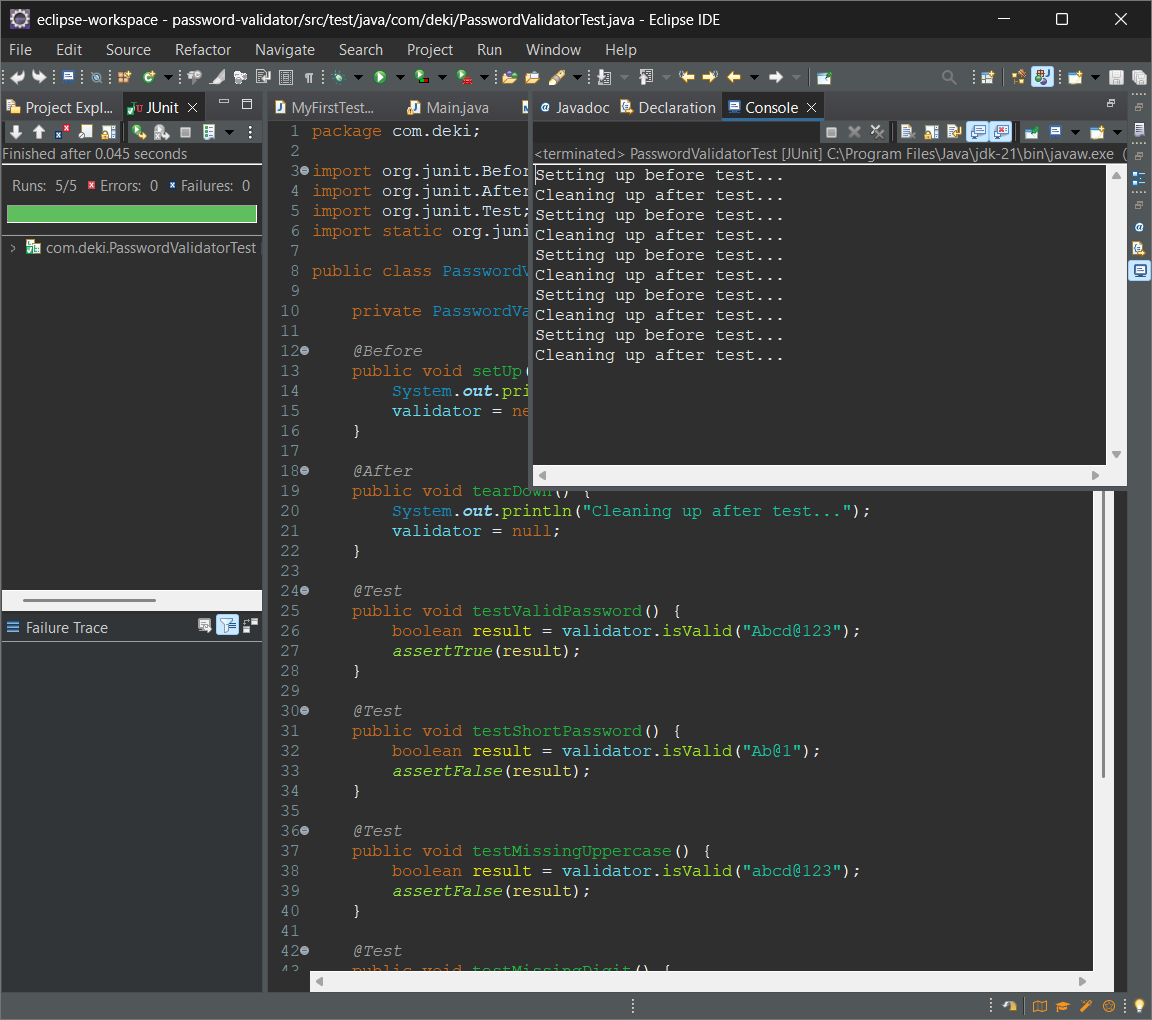
public void testNullPassword() {

boolean result = validator.isValid(null);

assertFalse(result);

}

}



6) Mocking and Stubbing :

MyServiceTest.java

package com.jothees;

import org.junit.jupiter.api.Test;

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

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testWithUserInput() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String userInput = "HelloUser";

String result = service.fetchData(userInput);

System.out.println("Output: " + result); // Optional

assertEquals("Mock Data | Input: HelloUser", result);

}

}

MyService.java

package com.jothees;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData(String userInput) {

return api.getData() + " | Input: " + userInput;

}

}

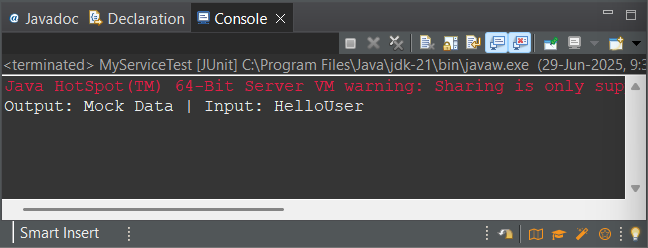
ExternalApi.java

package com.jothees;

public interface ExternalApi {

String getData();

}



7) Verifying Interactions :

MyService.java

package com.jothees;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

MyServiceTest.java

package com.jothees;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@*Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

ExternalApi

package com.jothees;

public interface ExternalApi {

void getData();

}



8) Logging Error Messages and Warning Levels :

LoggingExample

package com.example.logging;

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");

}

}

logback.xml

<configuration>

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

<encoder>

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

</encoder>

</appender>

<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

