1.CONTROL STRUCTURE

CREATE TABLE customers (

customer\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

dob DATE,

balance NUMBER,

isVIP CHAR(1) DEFAULT 'N'

);

CREATE TABLE loan1 (

loan\_id NUMBER PRIMARY KEY,

customer\_id NUMBER REFERENCES customers(customer\_id),

interest\_rate NUMBER,

due\_date DATE

);

INSERT INTO customers (customer\_id, name, dob, balance, isVIP)

VALUES (1, 'Ravi Kumar', TO\_DATE('15-08-1960','DD-MM-YYYY'), 12000, 'N');

INSERT INTO customers (customer\_id, name, dob, balance, isVIP)

VALUES (2, 'Priya Sharma', TO\_DATE('25-04-1985','DD-MM-YYYY'), 8000, 'N');

INSERT INTO customers (customer\_id, name, dob, balance, isVIP)

VALUES (3, 'Anil Verma', TO\_DATE('05-03-1955','DD-MM-YYYY'), 15000, 'N');

INSERT INTO customers (customer\_id, name, dob, balance, isVIP)

VALUES (4, 'Deepa Nair', TO\_DATE('20-12-1990','DD-MM-YYYY'), 9500, 'N');

INSERT INTO customers (customer\_id, name, dob, balance, isVIP)

VALUES (5, 'Manoj Das', TO\_DATE('10-06-1962','DD-MM-YYYY'), 20000, 'N');

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (101, 1, 9.5, TO\_DATE('15-07-2025','DD-MM-YYYY'));

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (102, 2, 10.0, TO\_DATE('10-08-2025','DD-MM-YYYY'));

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (103, 3, 8.5, TO\_DATE('25-07-2025','DD-MM-YYYY'));

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

VALUES (104, 4, 10.5, TO\_DATE('05-07-2025','DD-MM-YYYY'));

INSERT INTO loans (loan\_id, customer\_id, interest\_rate, due\_date)

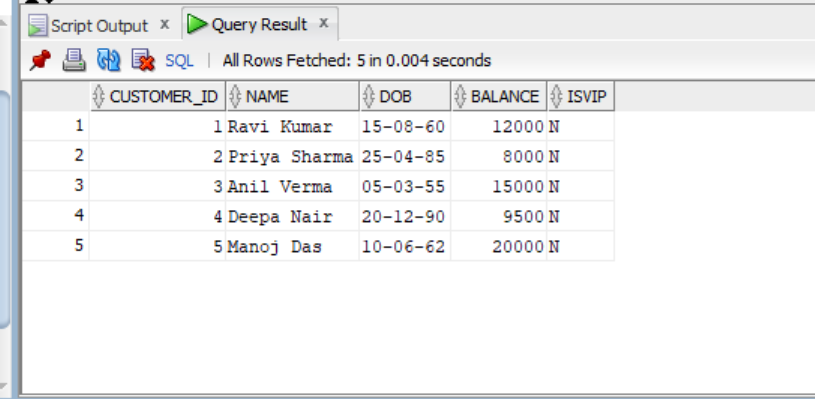
VALUES (105, 5, 9.0, TO\_DATE('30-07-2025','DD-MM-YYYY'));

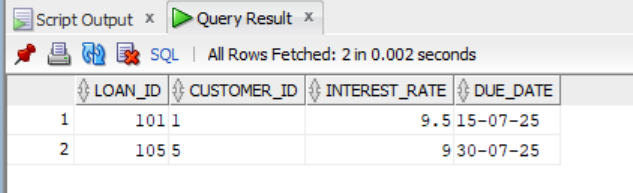
COMMIT;

SELECT \* FROM customers;

SELECT \* FROM loans;

OUTPUT:





2.STORED PROCEDURE

CREATE TABLE savings\_accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

CREATE TABLE employees (

emp\_id NUMBER PRIMARY KEY,

name VARCHAR2(100),

department VARCHAR2(50),

salary NUMBER

);

CREATE TABLE accounts (

account\_id NUMBER PRIMARY KEY,

balance NUMBER

);

INSERT INTO savings\_accounts VALUES (1, 5000);

INSERT INTO savings\_accounts VALUES (2, 8000);

INSERT INTO employees VALUES (1, 'Arjun', 'HR', 30000);

INSERT INTO employees VALUES (2, 'Divya', 'IT', 40000);

INSERT INTO accounts VALUES (101, 10000);

INSERT INTO accounts VALUES (102, 7000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR rec IN (SELECT account\_id, balance FROM savings\_accounts) LOOP

UPDATE savings\_accounts

SET balance = balance + (rec.balance \* 0.01)

WHERE account\_id = rec.account\_id;

END LOOP;

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_name IN VARCHAR2,

bonus\_pct IN NUMBER

) AS

BEGIN

UPDATE employees

SET salary = salary + (salary \* bonus\_pct / 100)

WHERE department = dept\_name;

COMMIT;

END;

/

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_acc IN NUMBER,

to\_acc IN NUMBER,

amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = from\_acc;

IF v\_balance >= amount THEN

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_acc;

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_acc;

COMMIT;

ELSE

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

END IF;

END;

/

EXEC ProcessMonthlyInterest;

EXEC UpdateEmployeeBonus('HR', 10);

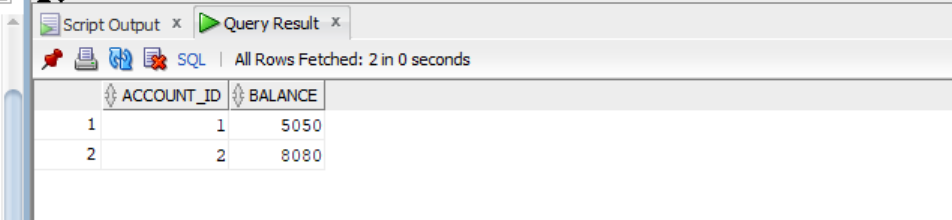
EXEC TransferFunds(101, 102, 2000);

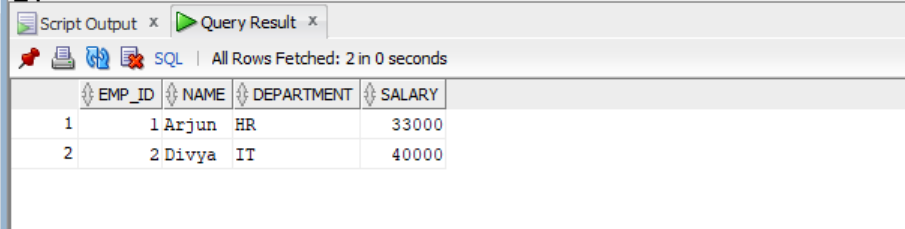
SELECT \* FROM savings\_accounts;

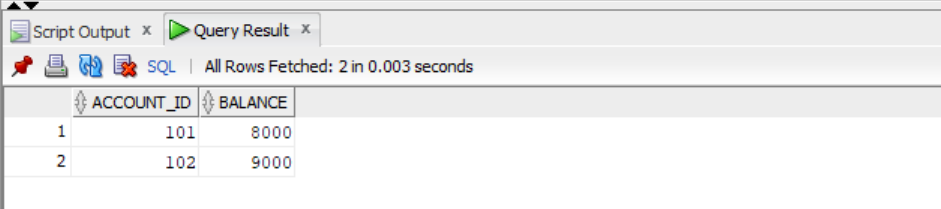
SELECT \* FROM employees;

SELECT \* FROM accounts;

OUTPUT:







3.SETTING UP JUNIT

Program

package ict.apply;

public class program {

public int add(int a)

{

return a;

}

public int add(int a,int b)

{

return a+b;

}

public int add(int a,int b,int c)

{

return a+b+c;

}

}

Test

package ict.apply;

import org.junit.Test;

import static org.junit.Assert.\*;

public class test {

public void test()

{

program obj=new program();

int r=obj.add(2);

*assertEquals*(2,r);

}

public void test1()

{

program obj1=new program();

int r1=obj1.add(2,3);

*assertEquals*(6,r1);

}

public void test2()

{

program obj2=new program();

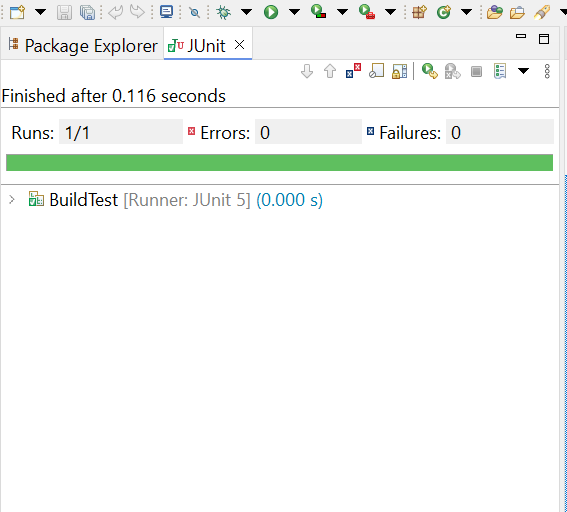
int r2=obj2.add(2,2,2);

*assertEquals*(8,r2);

}

}

OUTPUT:



4.ASSERTION IN JUNIT

package ict.ex1;

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

import org.junit.jupiter.api.Test;

public class Assertionstest {

public void testAssertions() {

*assertEquals*(5, 2 + 3);

*assertTrue*(5 > 3);

*assertFalse*(5 < 3);

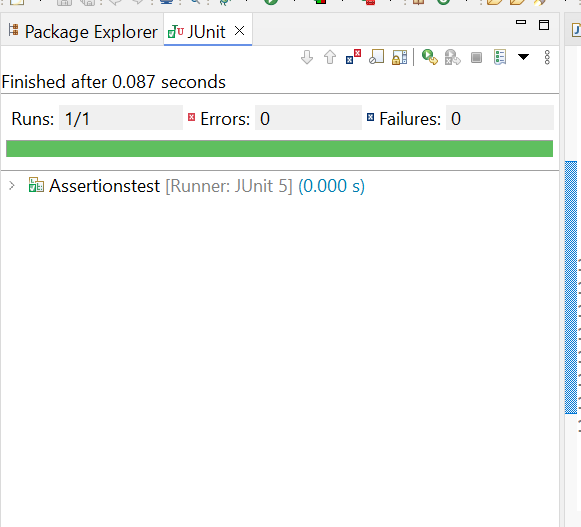
*assertNull*(null);

*assertNotNull*(new Object());

}

}

OUTPUT:



5.ARRANGE-ACT-ASSERT PATTERN PROGRAM

package ict.apply;

public class program1 {

public int add(int a, int b) {

return a + b;

}

public int multiply(int a, int b) {

return a \* b;

}

}

package ict.apply;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.Test;

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

public class ProgramTest {

private program1 program;

public void setUp() {

System.*out*.println("Setting up Program object...");

program = new program1(); // Arrange

}

public void tearDown() {

System.*out*.println("Cleaning up...\n");

program = null;

}

public void testAddition() {

int result = program.add(3, 4);

*assertEquals*(7, result);

System.*out*.println("Addition test passed.");

}

public void testMultiplication() {

int result = program.multiply(5, 2);

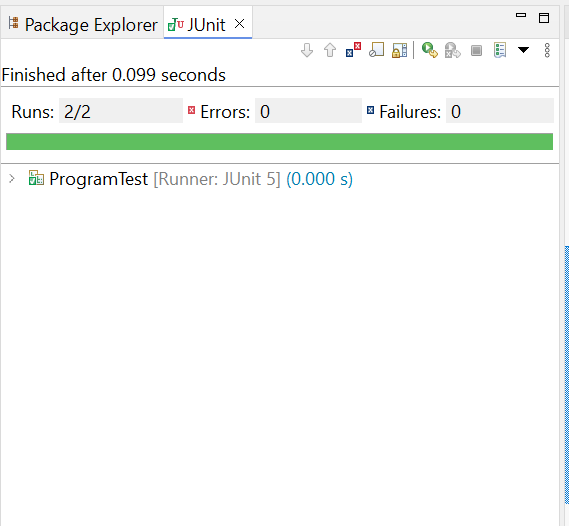
*assertEquals*(10, result);

System.*out*.println("Multiplication test passed.");

}

}

OUTPUT:



6.MOCKING AND STUBBING

package ict.ex;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class MyServicetest {

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

package ict.ex;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

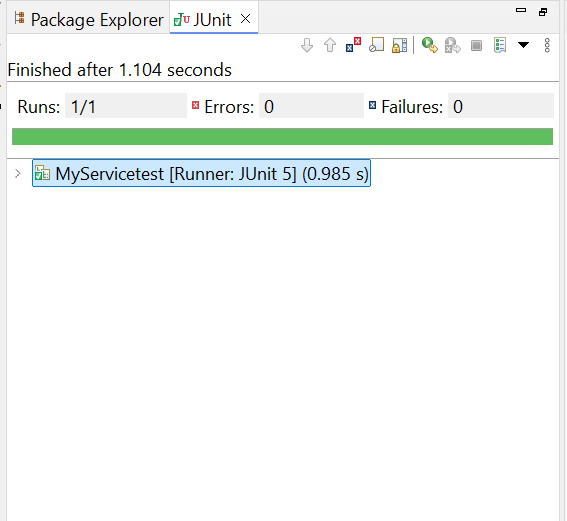
package ict.ex;

public interface ExternalApi {

String getData();

}

OUTPUT:



7.VERIFY INSTRUCTION

package ict.ex2;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

interface ExternalApi {

String getData();

}

class MyService {

ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

public class Myservicetest {

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

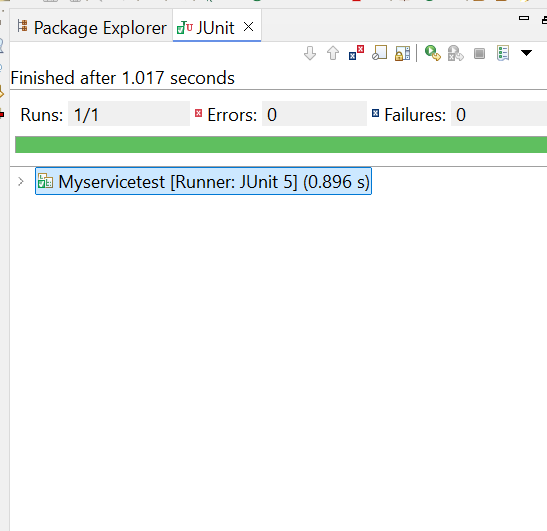
service.fetchData();

*verify*(mockApi).getData();

}

}

OUTPUT:



8.LOGGING ERROR MESSAGES AND WARNING LEVELS

package ict.ex3;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class login {

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");

}

}

OUTPUT:

