**WEEK-2**

**Exercise-1: Control Structures**

DECLARE

CURSOR cur\_customers\_loans IS

SELECT c.customer\_id, c.age, l.loan\_id, l.interest\_rate

FROM customers c

JOIN loans l ON c.customer\_id = l.customer\_id;

CURSOR cur\_all\_customers IS

SELECT customer\_id, balance

FROM customers;

CURSOR cur\_due\_loans IS

SELECT l.loan\_id, c.customer\_id, c.name, l.due\_date

FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE l.due\_date <= SYSDATE + 30;

BEGIN

FOR rec1 IN cur\_customers\_loans LOOP

IF rec1.age > 60 THEN

UPDATE loans

SET interest\_rate = interest\_rate - 1

WHERE loan\_id = rec1.loan\_id;

END IF;

END LOOP;

FOR rec2 IN cur\_all\_customers LOOP

IF rec2.balance > 10000 THEN

UPDATE customers

SET isvip = 'TRUE'

WHERE customer\_id = rec2.customer\_id;

END IF;

END LOOP;

FOR rec3 IN cur\_due\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec3.loan\_id ||

' for customer ' || rec3.name ||

' (ID: ' || rec3.customer\_id || ') is due on ' ||

TO\_CHAR(rec3.due\_date, 'DD-MON-YYYY'));

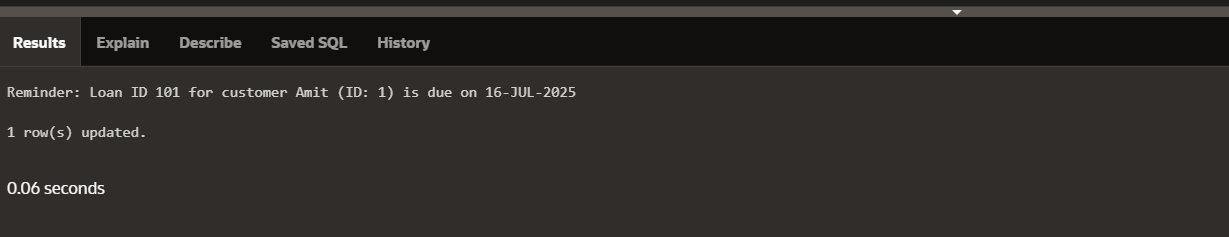
END LOOP;

COMMIT;

END;

/

**OUTPUT:**



**Exercise 2: Stored Procedures**

-- Procedure 1: ProcessMonthlyInterest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE accounts

SET balance = balance + (balance \* 0.01)

WHERE LOWER(account\_type) = 'savings';

COMMIT;

END;

/

-- Procedure 2: UpdateEmployeeBonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_dept\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE employees

SET salary = salary + (salary \* p\_bonus\_percent / 100)

WHERE dept\_id = p\_dept\_id;

COMMIT;

END;

/

-- Procedure 3: TransferFunds

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM accounts

WHERE account\_id = p\_from\_account

FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

END IF;

UPDATE accounts

SET balance = balance - p\_amount

WHERE account\_id = p\_from\_account;

UPDATE accounts

SET balance = balance + p\_amount

WHERE account\_id = p\_to\_account;

COMMIT;

END;

/

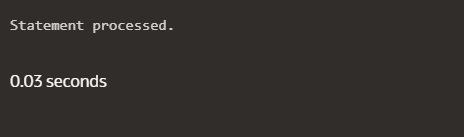
**OUTPUT:**

BEGIN

ProcessMonthlyInterest;

END;

/



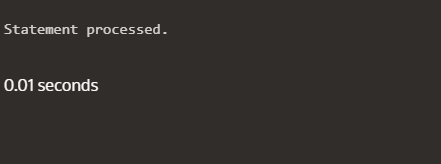
/

BEGIN

UpdateEmployeeBonus(20, 10);

END;

/



BEGIN

TransferFunds(101, 202, 1000);

END;

/



**Exercise 3: Setting Up Junit**

**Calculatortest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class Calculatortest {

    @Test

    public void testAdd() {

        Calculator calc = new Calculator();

        int result = calc.add(2, 3);

        assertEquals(5, result);

    }

}

**Calculator.java**

package com.example;

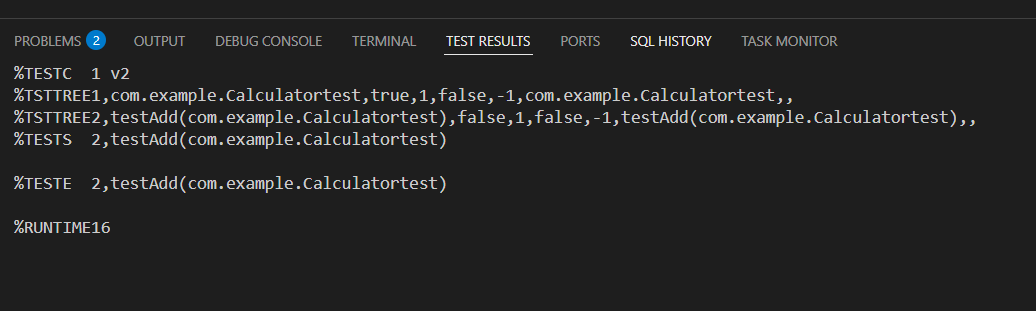
public class Calculator {

    public int add(int a, int b) {

        return a + b;}

}

**OUTPUT:**



**Exercise 4 : Assertions in Junit**

**Assertions.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

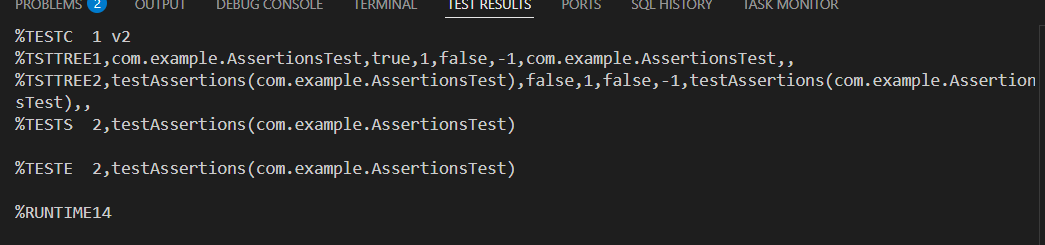
assertNull(null);

assertNotNull(new Object());

}

}

**OUTPUT:**



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

**Calculatortest.java**

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class Calculatortest {

    private Calculator calculator;

    @Before

    public void setUp() {

        calculator = new Calculator();

    }

    @After

    public void tearDown() {

        calculator = null;

    }

    @Test

    public void testAdd() {

        int a = 2;

        int b = 3;

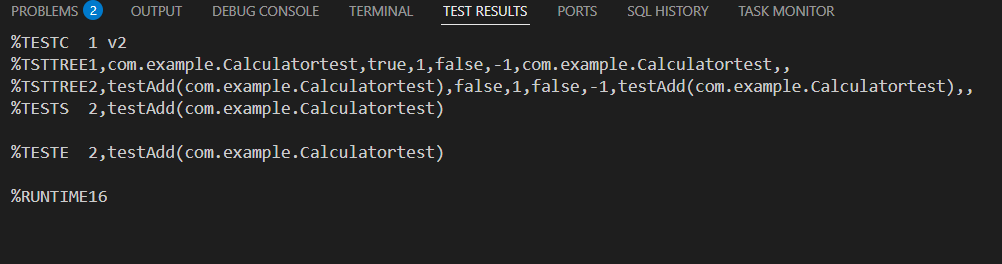
        int result = calculator.add(a, b);

        assertEquals(5, result);

    }

}

**OUTPUT:**



**Exercise 6: Mocking and Stubbing**

**MyServiceTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

    @Test

    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);

    }

}

**MyService.java**

package com.example;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

    public String fetchData() {

        return api.getData();

    }

}

**ExternalApi.java**

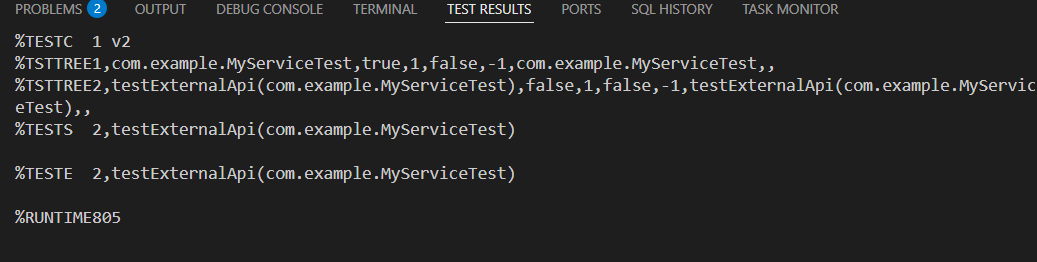
package com.example;

public interface ExternalApi {

    String getData();

}

**OUTPUT:**



**Exercise 7 : Verifying Interactions**

**MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

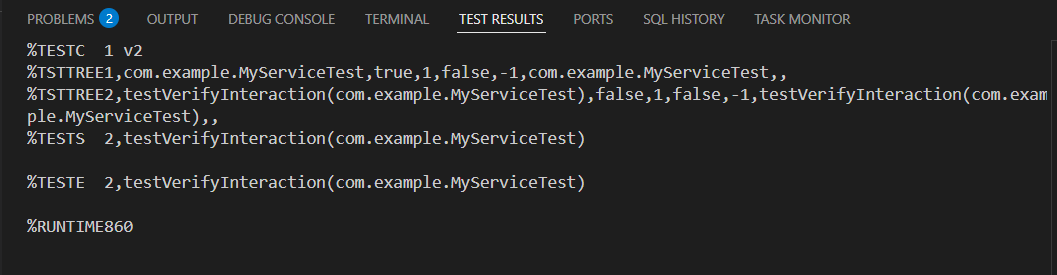
service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**



**Exercise 8 : Logging Error Messages and Warning Levels**

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

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

}

}

**OUTPUT:**

