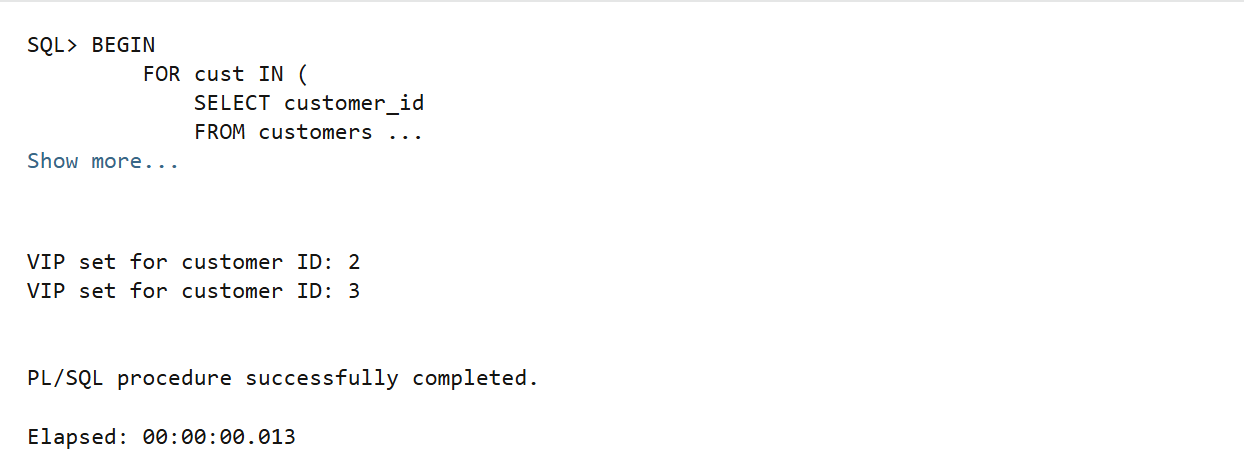
**Exercise1:ControlStructures**

1. **CREATE TABLE customers ( customer\_id NUMBERPRIMARYKEY, name VARCHAR2(50), age balance NUMBER, NUMBER(10, 2), interest\_rate NUMBER(5, 2), isVIP VARCHAR2(5) );**
2. **CREATE TABLE loans ( loan\_id NUMBERPRIMARYKEY, customer\_id NUMBER REFERENCES customers(customer\_id), due\_date DATE );**
3. **INSERT INTO customers VALUES (1, 'Ravi', 65, 8000, 10.5, 'FALSE'); INSERT INTO customers VALUES (2, 'Sneha', 45, 12000, 9.0, 'FALSE');**
4. **INSERT INTO customers VALUES (3, 'Amit', 70, 15000, 8.5, 'FALSE'); INSERT INTO customers VALUES (4, 'Priya', 30, 5000, 11.0, 'FALSE');**
5. **INSERT INTO loans VALUES (101, 1, SYSDATE + 10); INSERT INTO loans VALUES (102, 2, SYSDATE + 45);**
6. **INSERT INTO loans VALUES (103, 3, SYSDATE + 5); INSERT INTO loans VALUES (104, 4, SYSDATE + 25); COMMIT;**
7. **SET SERVEROUTPUT ON; BEGIN FORcust IN (SELECT customer\_id, interest\_rate FROM customers WHEREage>60)LOOP UPDATE customers SET interest\_rate = interest\_rate- 1 WHEREcustomer\_id = cust.customer\_id;**
8. **DBMS\_OUTPUT.PUT\_LINE('Discount applied to customer ID: ' || cust.customer\_id); ENDLOOP; COMMIT;**
9. **END;**
10. **/ BEGIN FORcust IN (SELECT customer\_id, balance FROM customers WHERE balance > 10000) LOOP UPDATE customers SET isVIP = 'TRUE' WHEREcustomer\_id = cust.customer\_id;**
11. **DBMS\_OUTPUT.PUT\_LINE('VIP set for customer ID: ' || cust.customer\_id); ENDLOOP;**
12. **COMMIT;**
13. **END;**
14. **/ BEGIN FORloan\_rec IN ( SELECT l.loan\_id, c.name, l.due\_date FROMloans l JOIN customers c ON c.customer\_id = l.customer\_id WHEREl.due\_date <= SYSDATE + 30 ) LOOP DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan\_rec.loan\_id || ' for customer ' || loan\_rec.name || ' is due on ' || TO\_CHAR(loan\_rec.due\_date, 'DD-Mon-YYYY'));**
15. **ENDLOOP;**
16. **END; /**

**OUTPUT:**

****

**Exercise3:StoredProcedures**

**BEGIN EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts';**

**EXCEPTION WHENOTHERSTHENNULL;**

**END;**

**/ CREATE TABLE savings\_accounts ( account\_id NUMBERPRIMARYKEY, customer\_id NUMBER, balance );**

**NUMBER(10,2) INSERT INTO savings\_accounts VALUES (1, 101, 10000); INSERT INTO savings\_accounts VALUES (2, 102, 15000); COMMIT; BEGIN EXECUTE IMMEDIATE 'DROP TABLE employees';**

**EXCEPTION WHENOTHERSTHENNULL;**

**END;**

**/ CREATE TABLE employees ( emp\_id NUMBERPRIMARYKEY, name VARCHAR2(50), department VARCHAR2(30), salary );**

**NUMBER(10,2) INSERT INTO employees VALUES (1, 'Alice', 'HR', 50000);**

**INSERT INTO employees VALUES (2, 'Bob', 'HR', 60000); INSERT INTO employees VALUES (3, 'Charlie', 'IT', 70000);**

**COMMIT;**

**BEGIN EXECUTE IMMEDIATE 'DROP TABLE accounts';**

**EXCEPTION WHENOTHERSTHENNULL;**

**END;**

**/ CREATE TABLE accounts ( account\_id NUMBER PRIMARY KEY, customer\_id NUMBER, balance NUMBER(10,2) );**

**INSERT INTO accounts VALUES (101, 1, 20000);**

**INSERT INTO accounts VALUES (102, 1, 5000);**

**COMMIT;**

**CREATE ORREPLACEPROCEDUREProcessMonthlyInterest IS BEGIN FORacc IN (SELECT account\_id, balance FROM savings\_accounts) LOOP UPDATE savings\_accounts SET balance = balance + (balance \* 0.01) WHEREaccount\_id = acc.account\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.account\_id); ENDLOOP; COMMIT; END;**

**/ CREATE ORREPLACEPROCEDUREUpdateEmployeeBonus ( dept\_name IN VARCHAR2, bonus\_pct IN NUMBER ) IS BEGIN FORempIN(SELECT emp\_id, name FROMemployees WHERE department = dept\_name) LOOP UPDATE employees SET salary = salary + (salary \* bonus\_pct / 100) WHEREemp\_id=emp.emp\_id;**

**DBMS\_OUTPUT.PUT\_LINE('Bonus applied to ' || emp.name || ' in ' || dept\_name); ENDLOOP;**

**COMMIT;   
END;**

**/ CREATE ORREPLACEPROCEDURETransferFunds ( from\_account IN NUMBER, to\_account IN NUMBER, amount ) IS IN NUMBER insufficient\_funds EXCEPTION;**

**from\_balance NUMBER; BEGIN SELECT balance INTO from\_balance FROM accounts WHERE account\_id = from\_account;**

**IF from\_balance < amount THEN RAISE insufficient\_funds; ELSE UPDATE accounts SET balance = balance- amount WHERE account\_id = from\_account; UPDATE accounts SET balance = balance + amount WHERE account\_id = to\_account;**

**DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from account ' || from\_account || ' to ' || to\_account); COMMIT; ENDIF; EXCEPTION WHENinsufficient\_funds THEN DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');**

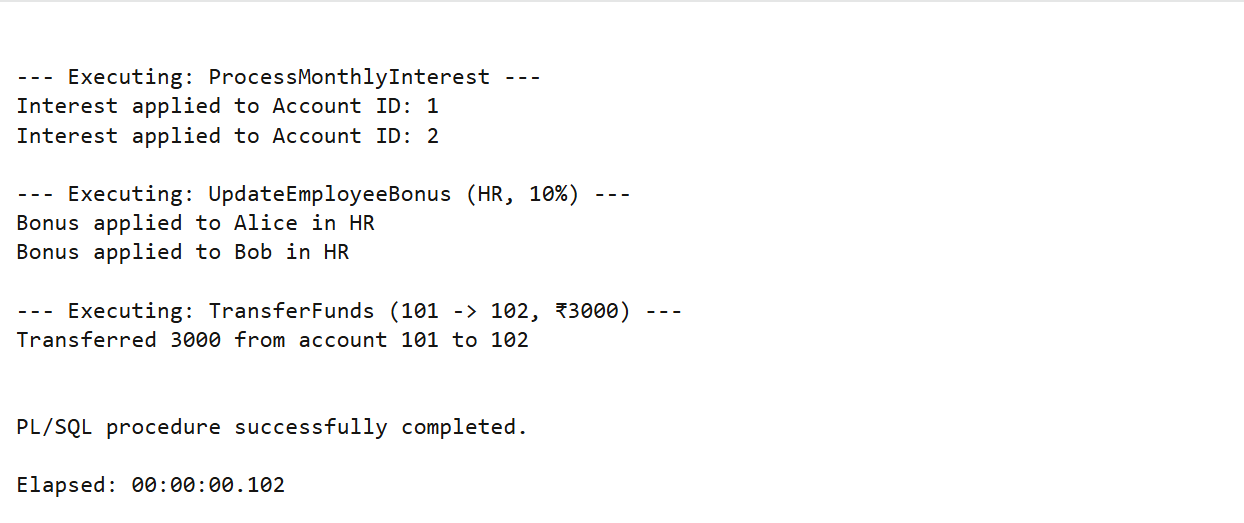
**WHENNO\_DATA\_FOUNDTHEN DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Account not found.'); END; / SET SERVEROUTPUTON;**

**BEGIN DBMS\_OUTPUT.PUT\_LINE('--- Executing: ProcessMonthlyInterest---'); ProcessMonthlyInterest;**

**DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Executing: UpdateEmployeeBonus (HR, 10%)---'); UpdateEmployeeBonus('HR', 10); DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Executing: TransferFunds (101-> 102, ₹3000)---'); TransferFunds(101, 102, 3000);**

**END;**

**OUTPUT**

****

**JUNIT EXCERCISE**

**Exercise1: Setting Up Junit**

**Calculator.java**

**public class Calculator {**

**public int add(int a, int b) {**

**return a + b;**

**}**

**}**

**CalculatorTest.java**

**import org.junit.Test;**

**import static org.junit.Assert.\*;**

**public class CalculatorTest {**

**@Test**

**public void testAdd() {**

**Calculator calc = new Calculator();**

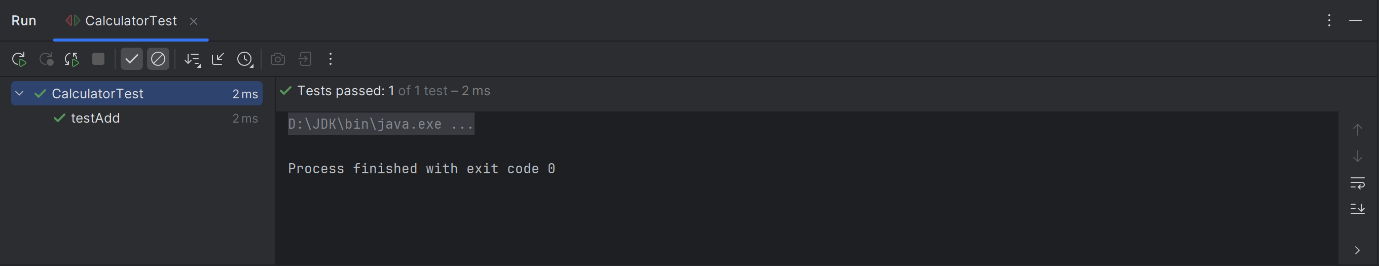
**int result = calc.add(3, 5);**

**assertEquals(8, result);**

**}**

**}**

**OUTPUT:**

****

**EXERCISE 3:**

**import org.junit.Test;**

**import static org.junit.Assert.\*;**

**public class AssertionsTest {**

**@Test**

**public void testAssertions() {**

**// Check equality**

**assertEquals(5, 2 + 3);**

**// Check boolean conditions**

**assertTrue(5 > 3);**

**assertFalse(5 < 3);**

**// Null checks**

**assertNull(null);**

**assertNotNull(new Object());**

**}**

**}**

**OUTPUT:**

****

**Mockito exercises**

**Exercise1:Mocking and Stubbing**

**ExternalApi.java**

**public interface ExternalApi {**

**String getData();**

**}**

**MyService.java**

**public class MyService {**

**private ExternalApi api;**

**public MyService(ExternalApi api) {**

**this.api = api;**

**}**

**public String fetchData() {**

**return api.getData();**

**}**

**}**

**MyServiceTest.java**

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

**import static org.mockito.Mockito.\*;**

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

**import org.mockito.Mockito;**

**public class MyServiceTest {**

**@Test**

**public void testExternalApi() {**

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

**when(mockApi.getData()).thenReturn("MockData");**

**MyService service = new MyService(mockApi);**

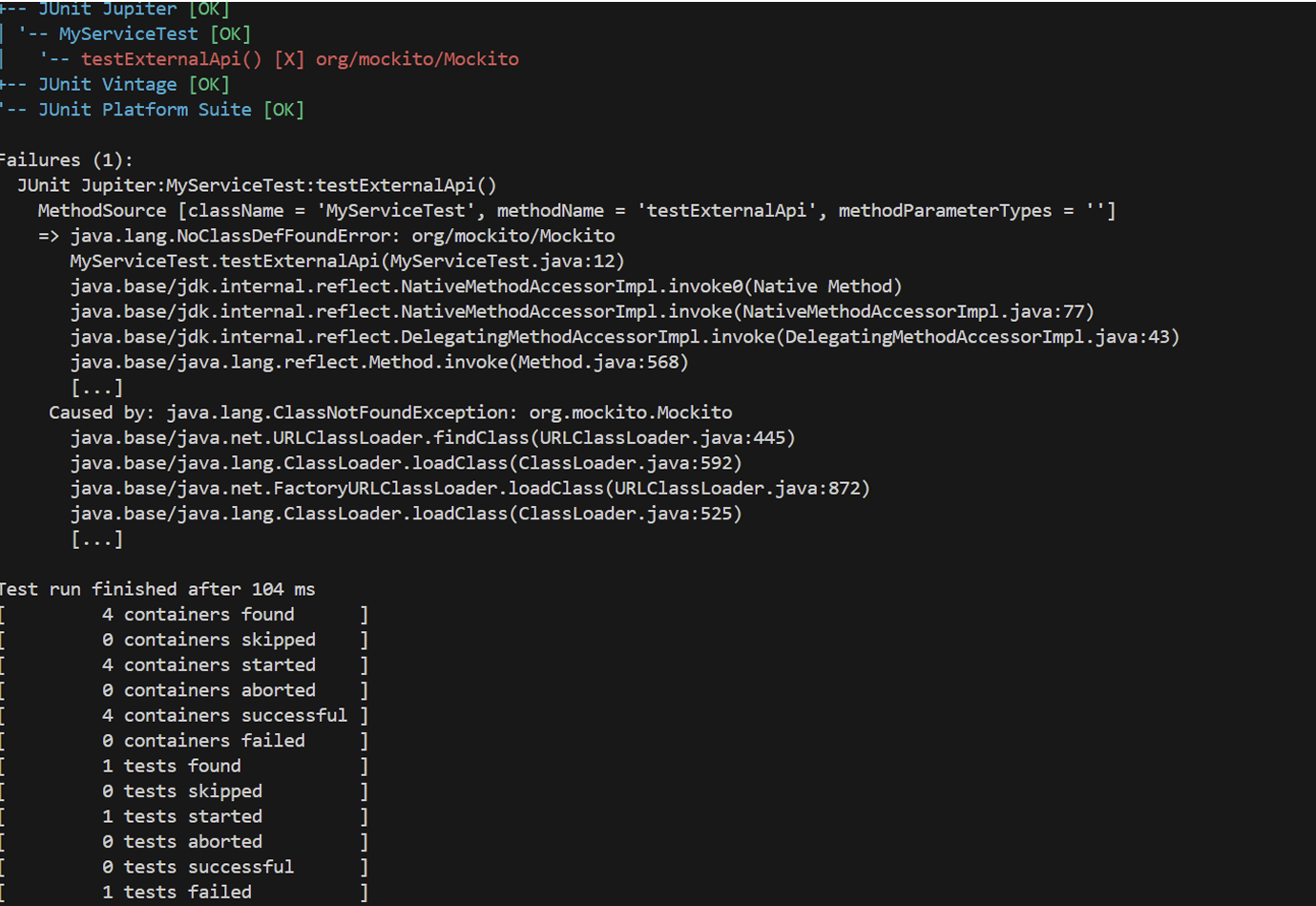
**String result = service.fetchData();**

**assertEquals("MockData", result);**

**}**

**}**

**OUTPUT:**

****

**Exercise2: Verifying Interactions:**

**ExternalApi.java**

**package Exercise7\_VerifyingInteractions;**

**public interface ExternalApi {**

**void getData();**

**}**

**MyService.java**

**package Exercise7\_VerifyingInteractions;**

**public class MyService {**

**private ExternalApi api;**

**public MyService(ExternalApi api) {**

**this.api = api;**

**}**

**public void fetchData() {**

**api.getData();**

**}**

**}**

**MyServiceTest.java**

**package Exercise7\_VerifyingInteractions;**

**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);**

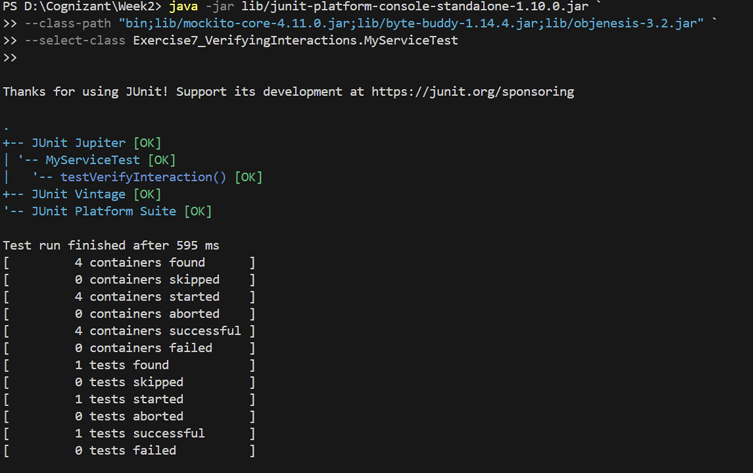
**service.fetchData();**

**verify(mockApi).getData();**

**}**

**}**

**OUTPUT:**

****

**Logging using SLF4J**

**Exercise1:LoggingError**

**package LoggingErrorMessages;**

**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) {**

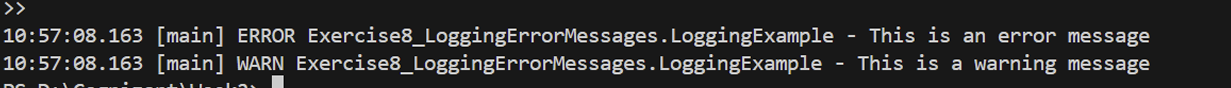
**logger.error("This is an error message");**

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

**}**

**}**

**Output**

****