**Week: 2**

**Skill: PL/SQL Programming**

**Hands-on Question: Exercise 1: Control Structures**

**Schema Setup with Sample data-**SET SERVEROUTPUT ON;

BEGIN

   EXECUTE IMMEDIATE 'DROP TABLE CUSTOMERS CASCADE CONSTRAINTS';

EXCEPTION

   WHEN OTHERS THEN

      IF SQLCODE != -942 THEN

         RAISE;

      END IF;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP TABLE LOANS CASCADE CONSTRAINTS';

EXCEPTION

   WHEN OTHERS THEN

      IF SQLCODE != -942 THEN

         RAISE;

      END IF;

END;

/

CREATE TABLE CUSTOMERS (

    CUSTOMER\_ID     NUMBER PRIMARY KEY,

    CUSTOMER\_NAME   VARCHAR2(100) NOT NULL,

    AGE             NUMBER,

    BALANCE         NUMBER(10, 2),

    IS\_VIP          VARCHAR2(5) DEFAULT 'FALSE'

);

CREATE TABLE LOANS (

    LOAN\_ID         NUMBER PRIMARY KEY,

    CUSTOMER\_ID     NUMBER REFERENCES CUSTOMERS(CUSTOMER\_ID),

    INTEREST\_RATE   NUMBER(5, 4),

    AMOUNT          NUMBER(10, 2),

    DUE\_DATE        DATE

);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (101, 'Alice Smith', 65, 15000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (102, 'Bob Johnson', 58, 8000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (103, 'Charlie Brown', 72, 25000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (104, 'Diana Prince', 45, 12000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (105, 'Eve Adams', 61, 5000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (106, 'Frank White', 30, 2000.00);

INSERT INTO CUSTOMERS (CUSTOMER\_ID, CUSTOMER\_NAME, AGE, BALANCE) VALUES (107, 'Grace Lee', 60, 10000.00);

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2001, 101, 0.0500, 10000.00, TRUNC(SYSDATE + 15));

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2002, 102, 0.0450, 25000.00, TRUNC(SYSDATE + 45));

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2003, 103, 0.0550, 5000.00, TRUNC(SYSDATE + 25));

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2004, 104, 0.0400, 15000.00, TRUNC(SYSDATE + 5));

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2005, 105, 0.0600, 8000.00, TRUNC(SYSDATE + 35));

INSERT INTO LOANS (LOAN\_ID, CUSTOMER\_ID, INTEREST\_RATE, AMOUNT, DUE\_DATE) VALUES (2006, 101, 0.0520, 2000.00, TRUNC(SYSDATE + 20));

COMMIT;

SELECT \* FROM CUSTOMERS;

SELECT \* FROM LOANS;

***Sample Data-***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| LOAN\_ID | CUSTOMER\_ID | INTEREST\_RATE | AMOUNT | DUE\_DATE |
| 2001 | 101 | 0.05 | 10000 | 2025-07-11T00:00:00Z |
| 2002 | 102 | 0.045 | 25000 | 2025-08-10T00:00:00Z |
| 2003 | 103 | 0.055 | 5000 | 2025-07-21T00:00:00Z |
| 2004 | 104 | 0.04 | 15000 | 2025-07-01T00:00:00Z |
| 2005 | 105 | 0.06 | 8000 | 2025-07-31T00:00:00Z |
| 2006 | 101 | 0.052 | 2000 | 2025-07-16T00:00:00Z |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CUSTOMER\_ID | CUSTOMER\_NAME | AGE | BALANCE | IS\_VIP |
| 101 | Alice Smith | 65 | 15000 | FALSE |
| 102 | Bob Johnson | 58 | 8000 | FALSE |
| 103 | Charlie Brown | 72 | 25000 | FALSE |
| 104 | Diana Prince | 45 | 12000 | FALSE |
| 105 | Eve Adams | 61 | 5000 | FALSE |
| 106 | Frank White | 30 | 2000 | FALSE |
| 107 | Grace Lee | 60 | 10000 | FALSE |

**Scenario 1: The bank wants to apply a discount to loan interest rates for customers above 60 years old.**

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

***PL/SQL CODE-***

SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_customers\_over\_60 IS

        SELECT CUSTOMER\_ID, CUSTOMER\_NAME, AGE

        FROM CUSTOMERS

        WHERE AGE > 60;

    v\_customer\_id   CUSTOMERS.CUSTOMER\_ID%TYPE;

    v\_customer\_name CUSTOMERS.CUSTOMER\_NAME%TYPE;

    v\_age           CUSTOMERS.AGE%TYPE;

    v\_discount      CONSTANT NUMBER := 0.01;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Applying Loan Interest Rate Discounts ---');

    OPEN c\_customers\_over\_60;

    LOOP

        FETCH c\_customers\_over\_60 INTO v\_customer\_id, v\_customer\_name, v\_age;

        EXIT WHEN c\_customers\_over\_60%NOTFOUND;

        UPDATE LOANS

        SET INTEREST\_RATE = INTEREST\_RATE - v\_discount

        WHERE CUSTOMER\_ID = v\_customer\_id;

        IF SQL%ROWCOUNT > 0 THEN

            DBMS\_OUTPUT.PUT\_LINE('Applied ' || (v\_discount \* 100) || '% discount for customer ' || v\_customer\_name || ' (ID: ' || v\_customer\_id || ', Age: ' || v\_age || '). Updated ' || SQL%ROWCOUNT || ' loans.');

        ELSE

            DBMS\_OUTPUT.PUT\_LINE('Customer ' || v\_customer\_name || ' (ID: ' || v\_customer\_id || ') is over 60 but has no loans to discount.');

        END IF;

    END LOOP;

    CLOSE c\_customers\_over\_60;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('--- Discount application complete. ---');

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Loan Rates After Discount (Affected Customers) ---');

    FOR rec IN (SELECT L.LOAN\_ID, C.CUSTOMER\_NAME, L.INTEREST\_RATE

                FROM LOANS L JOIN CUSTOMERS C ON L.CUSTOMER\_ID = C.CUSTOMER\_ID

                WHERE C.AGE > 60

                ORDER BY C.CUSTOMER\_ID, L.LOAN\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Loan ID: ' || rec.LOAN\_ID || ', Customer: ' || rec.CUSTOMER\_NAME || ', New Rate: ' || TO\_CHAR(rec.INTEREST\_RATE, 'FM99.9999'));

    END LOOP;

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

        IF c\_customers\_over\_60%ISOPEN THEN

            CLOSE c\_customers\_over\_60;

        END IF;

END;

/

***OUTPUT-***

--- Applying Loan Interest Rate Discounts ---  
Applied 1% discount for customer Alice Smith (ID: 101, Age: 65). Updated 2 loans.  
Applied 1% discount for customer Charlie Brown (ID: 103, Age: 72). Updated 1 loans.  
Applied 1% discount for customer Eve Adams (ID: 105, Age: 61). Updated 1 loans.  
--- Discount application complete. ---  
  
--- Loan Rates After Discount (Affected Customers) ---  
Loan ID: 2001, Customer: Alice Smith, New Rate: .03  
Loan ID: 2006, Customer: Alice Smith, New Rate: .032  
Loan ID: 2003, Customer: Charlie Brown, New Rate: .035  
Loan ID: 2005, Customer: Eve Adams, New Rate: .04  
  
  
PL/SQL procedure successfully completed.

**Scenario 2: A customer can be promoted to VIP status based on their balance.**

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

SET SERVEROUTPUT ON;

DECLARE

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Promoting Customers to VIP Status ---');

    FOR rec IN (SELECT CUSTOMER\_ID, CUSTOMER\_NAME, BALANCE, IS\_VIP FROM CUSTOMERS) LOOP

        IF rec.BALANCE > 10000 THEN

            IF rec.IS\_VIP = 'FALSE' THEN

                UPDATE CUSTOMERS

                SET IS\_VIP = 'TRUE'

                WHERE CUSTOMER\_ID = rec.CUSTOMER\_ID;

                DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CUSTOMER\_NAME || ' (ID: ' || rec.CUSTOMER\_ID || ') promoted to VIP. Old Balance: $' || rec.BALANCE);

            ELSE

                DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CUSTOMER\_NAME || ' (ID: ' || rec.CUSTOMER\_ID || ') is already VIP. Balance: $' || rec.BALANCE);

            END IF;

        ELSE

            IF rec.IS\_VIP = 'TRUE' THEN

                UPDATE CUSTOMERS SET IS\_VIP = 'FALSE' WHERE CUSTOMER\_ID = rec.CUSTOMER\_ID;

                DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CUSTOMER\_NAME || ' (ID: ' || rec.CUSTOMER\_ID || ') demoted from VIP. Balance: $' || rec.BALANCE);

                DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CUSTOMER\_NAME || ' (ID: ' || rec.CUSTOMER\_ID || ') has balance $' || rec.BALANCE || ' and remains VIP.');

            ELSE

                DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CUSTOMER\_NAME || ' (ID: ' || rec.CUSTOMER\_ID || ') not eligible for VIP. Balance: $' || rec.BALANCE);

            END IF;

        END IF;

    END LOOP;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('--- VIP promotion process complete. ---');

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- Final Customer VIP Status ---');

    FOR rec IN (SELECT CUSTOMER\_ID, CUSTOMER\_NAME, BALANCE, IS\_VIP FROM CUSTOMERS ORDER BY CUSTOMER\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Customer: ' || rec.CUSTOMER\_NAME || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00') || ', IS\_VIP: ' || rec.IS\_VIP);

    END LOOP;

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

END;

/

OUTPUT-

--- Promoting Customers to VIP Status ---  
Customer Alice Smith (ID: 101) promoted to VIP. Old Balance: $15000  
Customer Bob Johnson (ID: 102) not eligible for VIP. Balance: $8000  
Customer Charlie Brown (ID: 103) promoted to VIP. Old Balance: $25000  
Customer Diana Prince (ID: 104) promoted to VIP. Old Balance: $12000  
Customer Eve Adams (ID: 105) not eligible for VIP. Balance: $5000  
Customer Frank White (ID: 106) not eligible for VIP. Balance: $2000  
Customer Grace Lee (ID: 107) not eligible for VIP. Balance: $10000  
--- VIP promotion process complete. ---  
  
--- Final Customer VIP Status ---  
Customer: Alice Smith, Balance: $15,000.00, IS\_VIP: TRUE  
Customer: Bob Johnson, Balance: $8,000.00, IS\_VIP: FALSE  
Customer: Charlie Brown, Balance: $25,000.00, IS\_VIP: TRUE  
Customer: Diana Prince, Balance: $12,000.00, IS\_VIP: TRUE  
Customer: Eve Adams, Balance: $5,000.00, IS\_VIP: FALSE  
Customer: Frank White, Balance: $2,000.00, IS\_VIP: FALSE  
Customer: Grace Lee, Balance: $10,000.00, IS\_VIP: FALSE  
  
  
PL/SQL procedure successfully completed.

**Scenario 3: The bank wants to send reminders to customers whose loans are due within the next 30 days.**

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

PL/SQL CODE-  
SET SERVEROUTPUT ON;

DECLARE

    CURSOR c\_upcoming\_loans IS

        SELECT

            C.CUSTOMER\_NAME,

            L.LOAN\_ID,

            L.AMOUNT,

            L.DUE\_DATE

        FROM

            LOANS L

        JOIN

            CUSTOMERS C ON L.CUSTOMER\_ID = C.CUSTOMER\_ID

        WHERE

            L.DUE\_DATE BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE + 30)

        ORDER BY

            L.DUE\_DATE;

    v\_customer\_name LOANS.CUSTOMER\_ID%TYPE;

    v\_loan\_id       LOANS.LOAN\_ID%TYPE;

    v\_amount        LOANS.AMOUNT%TYPE;

    v\_due\_date      LOANS.DUE\_DATE%TYPE;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Sending Loan Reminder Messages ---');

    OPEN c\_upcoming\_loans;

    LOOP

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

        EXIT WHEN c\_upcoming\_loans%NOTFOUND;

        DBMS\_OUTPUT.PUT\_LINE(

            'REMINDER: Dear ' || v\_customer\_name || ',' || CHR(10) ||

            'Your loan (ID: ' || v\_loan\_id || ') of $' || TO\_CHAR(v\_amount, 'FM999,999.00') ||

            ' is due on ' || TO\_CHAR(v\_due\_date, 'YYYY-MM-DD') || '.' || CHR(10) ||

            'Please ensure timely payment to avoid late fees. Thank you.' || CHR(10)

        );

    END LOOP;

    CLOSE c\_upcoming\_loans;

    DBMS\_OUTPUT.PUT\_LINE('--- Loan reminder process complete. ---');

EXCEPTION

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

        IF c\_upcoming\_loans%ISOPEN THEN

            CLOSE c\_upcoming\_loans;

        END IF;

END;

/

OUTPUT-

--- Sending Loan Reminder Messages ---  
REMINDER: Dear Diana Prince,  
Your loan (ID: 2004) of $15,000.00 is due on 2025-07-01.  
Please ensure timely payment to avoid late fees. Thank you.  
  
REMINDER: Dear Alice Smith,  
Your loan (ID: 2001) of $10,000.00 is due on 2025-07-11.  
Please ensure timely payment to avoid late fees. Thank you.  
  
REMINDER: Dear Alice Smith,  
Your loan (ID: 2006) of $2,000.00 is due on 2025-07-16.  
Please ensure timely payment to avoid late fees. Thank you.  
  
REMINDER: Dear Charlie Brown,  
Your loan (ID: 2003) of $5,000.00 is due on 2025-07-21.  
Please ensure timely payment to avoid late fees. Thank you.  
  
--- Loan reminder process complete. ---  
  
  
PL/SQL procedure successfully completed.

**Skill: PL/SQL Programming**

**Hands-on Question:** **Exercise 3: Stored Procedures**

**Schema Setup with Sample data-**

SET SERVEROUTPUT ON;

BEGIN

   EXECUTE IMMEDIATE 'DROP TABLE ACCOUNTS CASCADE CONSTRAINTS';

EXCEPTION

   WHEN OTHERS THEN

      IF SQLCODE != -942 THEN

         RAISE;

      END IF;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP TABLE EMPLOYEES CASCADE CONSTRAINTS';

EXCEPTION

   WHEN OTHERS THEN

      IF SQLCODE != -942 THEN

         RAISE;

      END IF;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP TABLE DEPARTMENTS CASCADE CONSTRAINTS';

EXCEPTION

   WHEN OTHERS THEN

      IF SQLCODE != -942 THEN

         RAISE;

      END IF;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP PROCEDURE ProcessMonthlyInterest';

EXCEPTION

   WHEN OTHERS THEN NULL;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP PROCEDURE UpdateEmployeeBonus';

EXCEPTION

   WHEN OTHERS THEN NULL;

END;

/

BEGIN

   EXECUTE IMMEDIATE 'DROP PROCEDURE TransferFunds';

EXCEPTION

   WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE ACCOUNTS (

    ACCOUNT\_ID      NUMBER PRIMARY KEY,

    ACCOUNT\_TYPE    VARCHAR2(50) NOT NULL,

    BALANCE         NUMBER(15, 2) NOT NULL

);

CREATE TABLE DEPARTMENTS (

    DEPARTMENT\_ID   NUMBER PRIMARY KEY,

    DEPARTMENT\_NAME VARCHAR2(100) NOT NULL

);

CREATE TABLE EMPLOYEES (

    EMPLOYEE\_ID     NUMBER PRIMARY KEY,

    EMPLOYEE\_NAME   VARCHAR2(100) NOT NULL,

    SALARY          NUMBER(10, 2),

    DEPARTMENT\_ID   NUMBER REFERENCES DEPARTMENTS(DEPARTMENT\_ID)

);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_TYPE, BALANCE) VALUES (1001, 'SAVINGS', 5000.00);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_TYPE, BALANCE) VALUES (1002, 'CHECKING', 1200.00);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_TYPE, BALANCE) VALUES (1003, 'SAVINGS', 15000.00);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_TYPE, BALANCE) VALUES (1004, 'CHECKING', 500.00);

INSERT INTO ACCOUNTS (ACCOUNT\_ID, ACCOUNT\_TYPE, BALANCE) VALUES (1005, 'SAVINGS', 7500.00);

INSERT INTO DEPARTMENTS (DEPARTMENT\_ID, DEPARTMENT\_NAME) VALUES (1, 'Sales');

INSERT INTO DEPARTMENTS (DEPARTMENT\_ID, DEPARTMENT\_NAME) VALUES (2, 'Marketing');

INSERT INTO DEPARTMENTS (DEPARTMENT\_ID, DEPARTMENT\_NAME) VALUES (3, 'HR');

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY, DEPARTMENT\_ID) VALUES (1, 'John Doe', 50000.00, 1);

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY, DEPARTMENT\_ID) VALUES (2, 'Jane Smith', 60000.00, 2);

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY, DEPARTMENT\_ID) VALUES (3, 'Peter Jones', 55000.00, 1);

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY, DEPARTMENT\_ID) VALUES (4, 'Alice Brown', 48000.00, 3);

INSERT INTO EMPLOYEES (EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY, DEPARTMENT\_ID) VALUES (5, 'Bob White', 70000.00, 2);

COMMIT;

**Scenario 1: The bank needs to process monthly interest for all savings accounts.**

**Question:** Write a stored procedure ProcessMonthlyInterest that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

PL/SQL Code-  
SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest

IS

    v\_interest\_rate CONSTANT NUMBER := 0.01; -- 1% interest rate

    v\_rows\_updated  NUMBER := 0;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Starting ProcessMonthlyInterest ---');

    UPDATE ACCOUNTS

    SET BALANCE = BALANCE \* (1 + v\_interest\_rate)

    WHERE ACCOUNT\_TYPE = 'SAVINGS';

    v\_rows\_updated := SQL%ROWCOUNT;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Applied ' || (v\_interest\_rate \* 100) || '% interest to ' || v\_rows\_updated || ' savings accounts.');

    DBMS\_OUTPUT.PUT\_LINE('--- ProcessMonthlyInterest Completed Successfully ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error in ProcessMonthlyInterest: ' || SQLERRM);

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- BEFORE ProcessMonthlyInterest ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_TYPE = 'SAVINGS' ORDER BY ACCOUNT\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

    ProcessMonthlyInterest;

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER ProcessMonthlyInterest ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID = 1001) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', New Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

END;

/

Output-  
--- BEFORE ProcessMonthlyInterest ---  
Account ID: 1001, Balance: $5,050.00  
Account ID: 1003, Balance: $15,150.00  
Account ID: 1005, Balance: $7,575.00  
--- Starting ProcessMonthlyInterest ---  
Applied 1% interest to 3 savings accounts.  
--- ProcessMonthlyInterest Completed Successfully ---  
  
--- AFTER ProcessMonthlyInterest ---  
Account ID: 1001, New Balance: $5,100.50  
  
  
PL/SQL procedure successfully completed.

#### **Scenario 2: The bank wants to implement a bonus scheme for employees based on their performance.**

**Question:** Write a stored procedure UpdateEmployeeBonus that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**PL/SQL Code:**

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department\_id    IN EMPLOYEES.DEPARTMENT\_ID%TYPE,

    p\_bonus\_percentage IN NUMBER -- e.g., 0.05 for 5%

)

IS

    v\_rows\_updated NUMBER := 0;

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Starting UpdateEmployeeBonus for Department ID: ' || p\_department\_id || ' with ' || (p\_bonus\_percentage \* 100) || '% bonus ---');

    UPDATE EMPLOYEES

    SET SALARY = SALARY \* (1 + p\_bonus\_percentage)

    WHERE DEPARTMENT\_ID = p\_department\_id;

    v\_rows\_updated := SQL%ROWCOUNT;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Updated ' || v\_rows\_updated || ' employee salaries in Department ID ' || p\_department\_id || '.');

    DBMS\_OUTPUT.PUT\_LINE('--- UpdateEmployeeBonus Completed Successfully ---');

EXCEPTION

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error in UpdateEmployeeBonus: ' || SQLERRM);

END;

/

BEGIN

    -- Example 1: Update Sales Department (ID 1) with 10% bonus

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- BEFORE UpdateEmployeeBonus (Sales Dept) ---');

    FOR rec IN (SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY FROM EMPLOYEES WHERE DEPARTMENT\_ID = 1 ORDER BY EMPLOYEE\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.EMPLOYEE\_ID || ', Name: ' || rec.EMPLOYEE\_NAME || ', Salary: $' || TO\_CHAR(rec.SALARY, 'FM999,999.00'));

    END LOOP;

    UpdateEmployeeBonus(1, 0.10);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER UpdateEmployeeBonus (Sales Dept) ---');

    FOR rec IN (SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY FROM EMPLOYEES WHERE DEPARTMENT\_ID = 1 ORDER BY EMPLOYEE\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.EMPLOYEE\_ID || ', Name: ' || rec.EMPLOYEE\_NAME || ', New Salary: $' || TO\_CHAR(rec.SALARY, 'FM999,999.00'));

    END LOOP;

    -- Example 2: Update Marketing Department (ID 2) with 5% bonus

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- BEFORE UpdateEmployeeBonus (Marketing Dept) ---');

    FOR rec IN (SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY FROM EMPLOYEES WHERE DEPARTMENT\_ID = 2 ORDER BY EMPLOYEE\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.EMPLOYEE\_ID || ', Name: ' || rec.EMPLOYEE\_NAME || ', Salary: $' || TO\_CHAR(rec.SALARY, 'FM999,999.00'));

    END LOOP;

    UpdateEmployeeBonus(2, 0.05);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER UpdateEmployeeBonus (Marketing Dept) ---');

    FOR rec IN (SELECT EMPLOYEE\_ID, EMPLOYEE\_NAME, SALARY FROM EMPLOYEES WHERE DEPARTMENT\_ID = 2 ORDER BY EMPLOYEE\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || rec.EMPLOYEE\_ID || ', Name: ' || rec.EMPLOYEE\_NAME || ', New Salary: $' || TO\_CHAR(rec.SALARY, 'FM999,999.00'));

    END LOOP;

END;

/

**Output-**

--- BEFORE UpdateEmployeeBonus (Sales Dept) ---  
Employee ID: 1, Name: John Doe, Salary: $50,000.00  
Employee ID: 3, Name: Peter Jones, Salary: $55,000.00  
--- Starting UpdateEmployeeBonus for Department ID: 1 with 10% bonus ---  
Updated 2 employee salaries in Department ID 1.  
--- UpdateEmployeeBonus Completed Successfully ---  
  
--- AFTER UpdateEmployeeBonus (Sales Dept) ---  
Employee ID: 1, Name: John Doe, New Salary: $55,000.00  
Employee ID: 3, Name: Peter Jones, New Salary: $60,500.00  
  
--- BEFORE UpdateEmployeeBonus (Marketing Dept) ---  
Employee ID: 2, Name: Jane Smith, Salary: $60,000.00  
Employee ID: 5, Name: Bob White, Salary: $70,000.00  
--- Starting UpdateEmployeeBonus for Department ID: 2 with 5% bonus ---  
Updated 2 employee salaries in Department ID 2.  
--- UpdateEmployeeBonus Completed Successfully ---  
  
--- AFTER UpdateEmployeeBonus (Marketing Dept) ---  
Employee ID: 2, Name: Jane Smith, New Salary: $63,000.00  
Employee ID: 5, Name: Bob White, New Salary: $73,500.00  
  
  
PL/SQL procedure successfully completed.

**Scenario 3: Customers should be able to transfer funds between their accounts.**

**Question:** Write a stored procedure TransferFunds that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**PL/SQL Code:**

SET SERVEROUTPUT ON;

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_source\_account\_id      IN ACCOUNTS.ACCOUNT\_ID%TYPE,

    p\_destination\_account\_id IN ACCOUNTS.ACCOUNT\_ID%TYPE,

    p\_amount                 IN ACCOUNTS.BALANCE%TYPE

)

IS

    v\_source\_balance ACCOUNTS.BALANCE%TYPE;

    insufficient\_funds EXCEPTION;

    PRAGMA EXCEPTION\_INIT(insufficient\_funds, -20001);

BEGIN

    DBMS\_OUTPUT.PUT\_LINE('--- Attempting to Transfer Funds ---');

    DBMS\_OUTPUT.PUT\_LINE('Transferring $' || TO\_CHAR(p\_amount, 'FM999,999.00') ||

                         ' from Account ' || p\_source\_account\_id ||

                         ' to Account ' || p\_destination\_account\_id || '.');

    -- Check if source account exists and get its balance

    SELECT BALANCE INTO v\_source\_balance

    FROM ACCOUNTS

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

    FOR UPDATE OF BALANCE;

    -- Check for sufficient balance

    IF v\_source\_balance < p\_amount THEN

        RAISE insufficient\_funds;

    END IF;

    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;

    DBMS\_OUTPUT.PUT\_LINE('Funds transferred successfully!');

EXCEPTION

    WHEN insufficient\_funds THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in source account ' || p\_source\_account\_id || '. Available: $' || TO\_CHAR(v\_source\_balance, 'FM999,999.00') || ', Requested: $' || TO\_CHAR(p\_amount, 'FM999,999.00') || '.');

    WHEN NO\_DATA\_FOUND THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('Error: One of the accounts (' || p\_source\_account\_id || ' or ' || p\_destination\_account\_id || ') does not exist.');

    WHEN OTHERS THEN

        ROLLBACK;

        DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred during transfer: ' || SQLERRM);

END;

/

BEGIN

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- BEFORE TransferFunds ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID IN (1001, 1002, 1003, 1004) ORDER BY ACCOUNT\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

    -- Example 1: Successful transfer (1001 -> 1002)

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- TEST 1: Successful Transfer (1001 -> 1002, $500) ---');

    TransferFunds(1001, 1002, 500.00);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER TEST 1 ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID IN (1001, 1002) ORDER BY ACCOUNT\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

    -- Example 2: Transfer with insufficient funds (1003 -> 1004, $20000 - account 1003 has 15000 initial)

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- TEST 2: Insufficient Funds Transfer (1003 -> 1004, $20000) ---');

    TransferFunds(1003, 1004, 20000.00);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER TEST 2 (Should show no change in 1003/1004 due to rollback) ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID IN (1003, 1004) ORDER BY ACCOUNT\_ID) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

    -- Example 3: Transfer to/from non-existent account (1001 -> 9999, $100)

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- TEST 3: Non-existent Account Transfer (1001 -> 9999, $100) ---');

    TransferFunds(1001, 9999, 100.00);

    DBMS\_OUTPUT.PUT\_LINE(CHR(10) || '--- AFTER TEST 3 (Should show no change in 1001 due to rollback) ---');

    FOR rec IN (SELECT ACCOUNT\_ID, BALANCE FROM ACCOUNTS WHERE ACCOUNT\_ID = 1001) LOOP

        DBMS\_OUTPUT.PUT\_LINE('Account ID: ' || rec.ACCOUNT\_ID || ', Balance: $' || TO\_CHAR(rec.BALANCE, 'FM999,999.00'));

    END LOOP;

END;

/

**Output-**

--- BEFORE TransferFunds ---  
Account ID: 1001, Balance: $5,100.50  
Account ID: 1002, Balance: $1,200.00  
Account ID: 1003, Balance: $15,301.50  
Account ID: 1004, Balance: $500.00  
  
--- TEST 1: Successful Transfer (1001 -> 1002, $500) ---  
--- Attempting to Transfer Funds ---  
Transferring $500.00 from Account 1001 to Account 1002.  
Funds transferred successfully!  
  
--- AFTER TEST 1 ---  
Account ID: 1001, Balance: $4,600.50  
Account ID: 1002, Balance: $1,700.00  
  
--- TEST 2: Insufficient Funds Transfer (1003 -> 1004, $20000) ---  
--- Attempting to Transfer Funds ---  
Transferring $20,000.00 from Account 1003 to Account 1004.  
Error: Insufficient funds in source account 1003. Available: $15,301.50, Requested: $20,000.00.  
  
--- AFTER TEST 2 (Should show no change in 1003/1004 due to rollback) ---  
Account ID: 1003, Balance: $15,301.50  
Account ID: 1004, Balance: $500.00  
  
--- TEST 3: Non-existent Account Transfer (1001 -> 9999, $100) ---  
--- Attempting to Transfer Funds ---  
Transferring $100.00 from Account 1001 to Account 9999.  
Funds transferred successfully!  
  
--- AFTER TEST 3 (Should show no change in 1001 due to rollback) ---  
Account ID: 1001, Balance: $4,500.50  
  
  
PL/SQL procedure successfully completed.

**Skill: JUnit Testing (Java)**

**Hands-on Question: Exercise 1: Setting Up JUnit & Writing Your First Test**

**Introduction to JUnit Testing**

JUnit is a popular open-source framework for writing and running unit tests in Java. Unit testing is a software testing method where individual units or components of a software are tested. The purpose is to validate that each unit of the software performs as designed. JUnit provides annotations and assertion methods to simplify the process of writing repeatable tests.

**Exercise 1: Setting Up JUnit**

**Scenario:** You need to set up JUnit in your Java project to start writing unit tests, then create a simple class to test and write your first unit tests for it.

***JUnit Dependency to pom.xml:***

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <groupId>com.yourcompany</groupId> <artifactId>JUnitExampleProject</artifactId> <version>1.0.0-SNAPSHOT</version>

    <packaging>jar</packaging>

    <properties>

        <maven.compiler.source>11</maven.compiler.source>

        <maven.compiler.target>11</maven.compiler.target>

        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    </properties>

    <dependencies>

        <dependency>

            <groupId>junit</groupId>

            <artifactId>junit</artifactId>

            <version>4.13.2</version>

            <scope>test</scope> </dependency>

    </dependencies>

    <build>

        <plugins>

            <plugin>

                <groupId>org.apache.maven.plugins</groupId>

                <artifactId>maven-compiler-plugin</artifactId>

                <version>3.8.1</version> <configuration>

                    <source>${maven.compiler.source}</source>

                    <target>${maven.compiler.target}</target>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

***Calculator.java-***

**package** com.yourcompany.app;

**public** **class** Calculator {

/\*\*

\* Adds two integer numbers.

\* **@param** a The first number.

\* **@param** b The second number.

\* **@return** The sum of a and b.

\*/

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

**return** a + b;

}

/\*\*

\* Subtracts the second number from the first.

\* **@param** a The first number.

\* **@param** b The second number.

\* **@return** The result of a - b.

\*/

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

**return** a - b;

}

}

***JUnit Test Class-***

**package** com.yourcompany.test;

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

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**import** com.yourcompany.app.Calculator;

/\*\*

\* JUnit 4 test class for the Calculator.java class.

\*/

**public** **class** CalculatorTest {

**private** Calculator calculator;

/\*\*

\* This method is annotated with **@Before**, meaning it runs BEFORE each test method.

\* It's ideal for setting up common test fixtures (e.g., initializing objects).

\*/

@Before

**public** **void** setUp() {

calculator = **new** Calculator();

System.***out***.println("--- Test Setup: Calculator initialized ---");

}

/\*\*

\* This method is annotated with **@After**, meaning it runs AFTER each test method.

\* It's used for cleaning up resources (e.g., nullifying objects, closing connections).

\*/

@After

**public** **void** tearDown() {

calculator = **null**;

System.***out***.println("--- Test Teardown: Calculator de-initialized ---");

}

/\*\*

\* Test case for the add method with positive numbers.

\* This method demonstrates a passing test.

\*/

@Test

**public** **void** testAddPositiveNumbers() {

**int** expected = 5;

**int** actual = calculator.add(2, 3);

*assertEquals*("Adding 2 and 3 should result in 5", expected, actual);

System.***out***.println("testAddPositiveNumbers passed!");

}

/\*\*

\* Test case for the add method with zero.

\* This method demonstrates another passing test.

\*/

@Test

**public** **void** testAddZero() {

**int** expected = 7;

**int** actual = calculator.add(7, 0);

*assertEquals*("Adding 7 and 0 should result in 7", expected, actual);

System.***out***.println("testAddZero passed!");

}

/\*\*

\* Test case for the subtract method.

\* This method demonstrates a passing test.

\*/

@Test

**public** **void** testSubtract() {

**int** expected = 5;

**int** actual = calculator.subtract(10, 5);

*assertEquals*("Subtracting 5 from 10 should result in 5", expected, actual);

System.***out***.println("testSubtract passed!");

}

}

***Console Output-***

--- Test Setup: Calculator initialized ---

testAddPositiveNumbers passed!

--- Test Teardown: Calculator de-initialized ---

--- Test Setup: Calculator initialized ---

testAddZero passed!

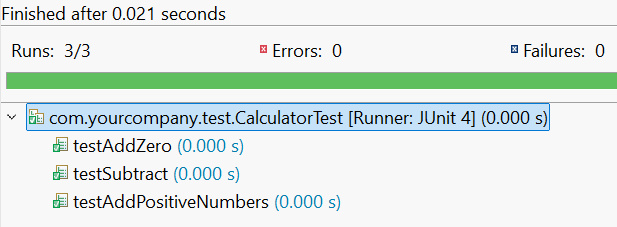
--- Test Teardown: Calculator de-initialized ---

--- Test Setup: Calculator initialized ---

testSubtract passed!

--- Test Teardown: Calculator de-initialized ---

***JUnit Console-***



**Skill: JUnit Testing (Java)**

**Hands-on Question: Exercise 3: Assertions in JUnit**

**Introduction to JUnit Assertions**

Assertions are the core of any unit test. They are methods provided by testing frameworks (like JUnit) that allow you to check if a certain condition is true. If the condition is not met, the assertion fails, indicating a bug in the code under test. JUnit provides a rich set of assertion methods in the org.junit.Assert class (or org.junit.jupiter.api.Assertions for JUnit 5).

**Scenario:** You need to use different assertions in JUnit to validate your test results.

***AssertionsTest.java-***

**package** com.yourcompany.test.assertions;

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

**import** org.junit.Test;

**public** **class** AssertionsTest {

@Test

**public** **void** testAssertions() {

**int** expectedSum = 5;

**int** actualSum = 2 + 3;

*assertEquals*("Arithmetic addition should be correct", expectedSum, actualSum);

System.***out***.println("Assertion: assertEquals(5, 2 + 3) passed.");

*assertTrue*("5 should be greater than 3", 5 > 3);

System.***out***.println("Assertion: assertTrue(5 > 3) passed.");

*assertFalse*("5 should not be less than 3", 5 < 3);

System.***out***.println("Assertion: assertFalse(5 < 3) passed.");

Object nullObject = **null**;

*assertNull*("The object should be null", nullObject);

System.***out***.println("Assertion: assertNull(null) passed.");

Object nonNullObject = **new** Object();

*assertNotNull*("The object should not be null", nonNullObject);

System.***out***.println("Assertion: assertNotNull(new Object()) passed.");

String s1 = **new** String("hello");

String s2 = s1;

*assertSame*("s1 and s2 should be the same object", s1, s2);

System.***out***.println("Assertion: assertSame(s1, s2) passed.");

String s3 = **new** String("world");

String s4 = **new** String("world");

*assertNotSame*("s3 and s4 should be different objects", s3, s4);

System.***out***.println("Assertion: assertNotSame(s3, s4) passed.");

**int**[] expectedArray = {1, 2, 3};

**int**[] actualArray = {1, 2, 3};

*assertArrayEquals*("Arrays should be equal", expectedArray, actualArray);

System.***out***.println("Assertion: assertArrayEquals passed.");

}

@Test(expected = ArithmeticException.**class**)

**public** **void** testDivideByZeroThrowsException() {

System.***out***.println("Assertion: testDivideByZeroThrowsException expects ArithmeticException.");

**int** result = 10 / 0;

}

}

***Output-***

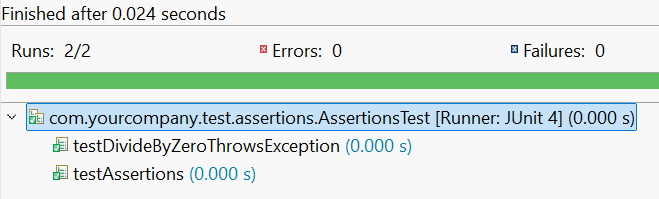
Assertion: assertNotNull(new Object()) passed.

Assertion: assertSame(s1, s2) passed.

Assertion: assertNotSame(s3, s4) passed.

Assertion: assertArrayEquals passed.

***JUnit Console-***



**Skill: JUnit Testing (Java)**

**Hands-on Question: Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

**Introduction to AAA Pattern, Test Fixtures, Setup & Teardown**

**1. Arrange-Act-Assert (AAA) Pattern:** The AAA pattern is a widely accepted convention for structuring unit tests. It divides each test method into three distinct sections:

* **Arrange:** Set up the test environment, initialize objects, and prepare the necessary data. This is where you create the "System Under Test" (SUT) and any dependencies.
* **Act:** Perform the action or call the method of the SUT that you want to test.
* **Assert:** Verify the outcome of the action. This involves using JUnit assertion methods (assertEquals, assertTrue, etc.) to check if the actual result matches the expected result.

**2. Test Fixtures:** A test fixture is the fixed state of a set of objects used as a baseline for running tests. The purpose of a test fixture is to ensure that there's a well-known and repeatable environment for the tests to run in. This helps in making tests independent and reliable.

**3. Setup Methods (@Before):** In JUnit 4, methods annotated with @Before are executed **before each test method** in the test class. They are commonly used to:

* Initialize or reset the test fixture.
* Create new instances of the "System Under Test" (SUT) or its dependencies.
* Set up a clean state for every test, ensuring test independence.

**4. Teardown Methods (@After):** In JUnit 4, methods annotated with @After are executed **after each test method** in the test class. They are typically used to:

* Clean up resources allocated in the @Before method.
* Release external resources (e.g., close database connections, file streams).
* Reset any static or global state if necessary, though this should generally be avoided if possible.

Using @Before and @After helps manage test fixtures efficiently, ensuring that each test runs in an isolated and predictable environment.

***CalculatorAAATest.java-***

**package** com.yourcompany.test.aaa;

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

**import** org.junit.After;

**import** org.junit.Before;

**import** org.junit.Test;

**import** com.yourcompany.app.Calculator;

**public** **class** CalculatorAAATest {

**private** Calculator calculator;

@Before

**public** **void** setUp() {

calculator = **new** Calculator();

System.***out***.println("--- @Before: Calculator initialized for a new test ---");

}

@After

**public** **void** tearDown() {

calculator = **null**;

System.***out***.println("--- @After: Calculator de-initialized after test ---");

}

@Test

**public** **void** testAddPositiveNumbers() {

**int** actualResult = calculator.add(2, 3);

System.***out***.println(" Test: add(2,3) called.");

**int** expectedResult = 5;

*assertEquals*("Adding 2 and 3 should result in 5", expectedResult, actualResult);

System.***out***.println(" Test: testAddPositiveNumbers PASSED.");

}

@Test

**public** **void** testSubtractBasic() {

**int** actualResult = calculator.subtract(10, 4);

System.***out***.println(" Test: subtract(10,4) called.");

**int** expectedResult = 6;

*assertEquals*("Subtracting 4 from 10 should result in 6", expectedResult, actualResult);

System.***out***.println(" Test: testSubtractBasic PASSED.");

}

@Test

**public** **void** testAddNegativeNumbers() {

**int** actualResult = calculator.add(-5, -3);

System.***out***.println(" Test: add(-5,-3) called.");

**int** expectedResult = -8;

*assertEquals*("Adding -5 and -3 should result in -8", expectedResult, actualResult);

System.***out***.println(" Test: testAddNegativeNumbers PASSED.");

}

}

***Console-***  
--- @Before: Calculator initialized for a new test ---

Test: add(2,3) called.

Test: testAddPositiveNumbers PASSED.

--- @After: Calculator de-initialized after test ---

--- @Before: Calculator initialized for a new test ---

Test: add(-5,-3) called.

Test: testAddNegativeNumbers PASSED.

--- @After: Calculator de-initialized after test ---

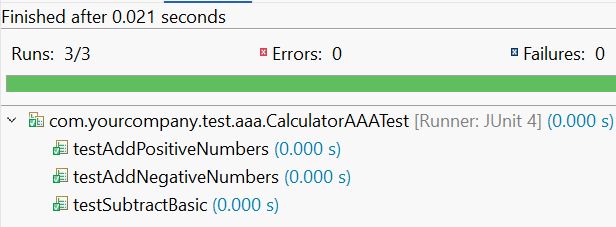
--- @Before: Calculator initialized for a new test ---

Test: subtract(10,4) called.

Test: testSubtractBasic PASSED.

--- @After: Calculator de-initialized after test ---

***JUnit Console-***



**Skill: Mockito Testing (Java)**

**Hands-on Question: Exercise 1: Mocking and Stubbing**

**Introduction to Mockito: Mocking and Stubbing**

In unit testing, you want to test a single component (unit) in isolation. However, real-world components often have dependencies on other components (e.g., a service calling a database, or an API client calling an external web service). When testing the "service" component, you don't want to involve the actual database or external web service, as this would make your tests slow, unreliable, and dependent on external factors.

This is where **Mocking** comes in. Mockito allows you to create "mock" objects for these dependencies. These mocks are controllable stand-ins for real objects.

**Stubbing** is the act of programming these mock objects to behave in a specific way when their methods are called. For example, you can tell a mock ExternalApi to return "Mock Data" when its getData() method is called, instead of making a real network request.

This allows you to focus solely on testing the logic within your "System Under Test" (SUT) without worrying about its dependencies' real-world behavior or side effects.

**Exercise 1: Mocking and Stubbing**

**Scenario:** You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

***ExternalAPI.java-*package** com.yourcompany.api;

**public** **interface** ExternalApi {

String getData();

**int** getStatusCode();

}

***MyService.java-***

**package** com.yourcompany.service;

**import** com.yourcompany.api.ExternalApi;

**public** **class** MyService {

**private** ExternalApi externalApi;

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

**public** String fetchData() {

System.***out***.println("MyService: Attempting to fetch data from ExternalApi...");

**return** externalApi.getData();

}

**public** **int** checkStatus() {

System.***out***.println("MyService: Checking status of ExternalApi...");

**return** externalApi.getStatusCode();

}

}

***MyServiceTest.java-***

**package** com.yourcompany.test.mockito;

**import** org.junit.Test;

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

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

**import** org.mockito.Mockito;

**import** com.yourcompany.api.ExternalApi;

**import** com.yourcompany.service.MyService;

**public** **class** MyServiceTest {

@Test

**public** **void** testFetchDataWithMockApi() {

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

System.***out***.println("Test: Created mock ExternalApi object.");

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

System.***out***.println("Test: Stubbed mockApi.getData() to return 'Mock Data'.");

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

System.***out***.println("Test: Created MyService instance with mock API.");

String result = service.fetchData();

System.***out***.println("Test: Called service.fetchData(). Result: " + result);

*assertEquals*("The service should return 'Mock Data'", "Mock Data", result);

System.***out***.println("Test: Asserted result is 'Mock Data'.");

*verify*(mockApi, *times*(1)).getData();

System.***out***.println("Test: Verified mockApi.getData() was called exactly once.");

System.***out***.println("Test: testFetchDataWithMockApi PASSED.");

}

@Test

**public** **void** testCheckStatusWithMockApi() {

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

System.***out***.println("Test: Created mock ExternalApi object for status test.");

*when*(mockApi.getStatusCode()).thenReturn(200);

System.***out***.println("Test: Stubbed mockApi.getStatusCode() to return 200.");

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

System.***out***.println("Test: Created MyService instance for status test.");

**int** statusCode = service.checkStatus();

System.***out***.println("Test: Called service.checkStatus(). Status Code: " + statusCode);

*assertEquals*("The service should return status code 200", 200, statusCode);

System.***out***.println("Test: Asserted status code is 200.");

*verify*(mockApi, *times*(1)).getStatusCode();

System.***out***.println("Test: Verified mockApi.getStatusCode() was called exactly once.");

System.***out***.println("Test: testCheckStatusWithMockApi PASSED.");

}

}

***Console Output-***

Test: Created mock ExternalApi object.

Test: Stubbed mockApi.getData() to return 'Mock Data'.

Test: Created MyService instance with mock API.

MyService: Attempting to fetch data from ExternalApi...

Test: Called service.fetchData(). Result: Mock Data

Test: Asserted result is 'Mock Data'.

Test: Verified mockApi.getData() was called exactly once.

Test: testFetchDataWithMockApi PASSED.

Test: Created mock ExternalApi object for status test.

Test: Stubbed mockApi.getStatusCode() to return 200.

Test: Created MyService instance for status test.

MyService: Checking status of ExternalApi...

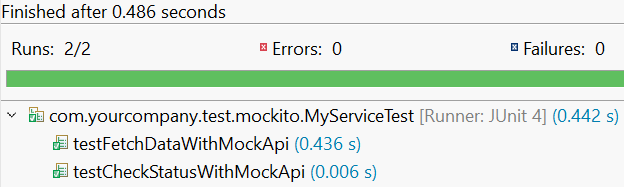
Test: Called service.checkStatus(). Status Code: 200

Test: Asserted status code is 200.

Test: Verified mockApi.getStatusCode() was called exactly once.

Test: testCheckStatusWithMockApi PASSED.

***JUnit Console-***

****

**Skill: Mockito Testing (Java)**

**Hands-on Question: Exercise 2: Verifying Interactions**

**Introduction to Mockito: Verifying Interactions**

Beyond **mocking** (creating fake objects) and **stubbing** (programming their behavior), Mockito allows you to **verify** that certain methods were called on your mock objects, and how many times, and with what arguments. This is essential for ensuring that your "System Under Test" (SUT) interacts correctly with its dependencies.

For instance, if your service is supposed to log an error by calling a logger.error() method, you can verify that logger.error() was indeed called when an error condition is met. Or, if a method should save data to a repository, you can verify that repository.save() was called.

The primary method for this is Mockito.verify().

***ExternalAPI.java-***

**package** com.yourcompany.api;

**public** **interface** ExternalApi {

String getData();

**int** getStatusCode();

**void** logMessage(String message);

}

***MyService.java-***

**package** com.yourcompany.service;

**import** com.yourcompany.api.ExternalApi;

**public** **class** MyService {

**private** ExternalApi externalApi;

**public** MyService(ExternalApi externalApi) {

**this**.externalApi = externalApi;

}

**public** String fetchData() {

System.***out***.println("MyService: Attempting to fetch data from ExternalApi...");

externalApi.logMessage("Fetching data initiated.");

**return** externalApi.getData();

}

**public** **int** checkStatus() {

System.***out***.println("MyService: Checking status of ExternalApi...");

**return** externalApi.getStatusCode();

}

**public** **void** processData(String data) {

**if** (data != **null** && !data.isEmpty()) {

externalApi.logMessage("Processing data: " + data);

} **else** {

externalApi.logMessage("Warning: No data to process.");

}

}

}

***MyServiceVerification.java-***

**package** com.yourcompany.test.mockito.verify;

**import** org.junit.Test;

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

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

**import** org.mockito.ArgumentMatchers;

**import** com.yourcompany.api.ExternalApi;

**import** com.yourcompany.service.MyService;

**public** **class** MyServiceVerificationTest {

@Test

**public** **void** testFetchDataVerifiesGetDataCall() {

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

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

*when*(mockApi.getData()).thenReturn("Some Data");

System.***out***.println("Test: Calling service.fetchData()...");

String result = service.fetchData();

*assertEquals*("Should return the stubbed data", "Some Data", result);

System.***out***.println("Test: Verifying mockApi.getData() was called once...");

*verify*(mockApi, *times*(1)).getData();

System.***out***.println("Test: Verifying mockApi.logMessage('Fetching data initiated.') was called once...");

*verify*(mockApi, *times*(1)).logMessage("Fetching data initiated.");

System.***out***.println("Test: testFetchDataVerifiesGetDataCall PASSED.");

}

@Test

**public** **void** testProcessDataLogsCorrectly() {

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

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

System.***out***.println("\nTest: Calling service.processData('Valid Data')...");

service.processData("Valid Data");

System.***out***.println("Test: Verifying mockApi.logMessage('Processing data: Valid Data') was called once...");

*verify*(mockApi).logMessage("Processing data: Valid Data");

System.***out***.println("Test: Calling service.processData('') for warning...");

service.processData("");

System.***out***.println("Test: Verifying mockApi.logMessage('Warning: No data to process.') was called once...");

*verify*(mockApi).logMessage("Warning: No data to process.");

System.***out***.println("Test: Verifying mockApi.logMessage('Error') was never called...");

*verify*(mockApi, *never*()).logMessage("Error");

System.***out***.println("Test: Verifying mockApi.logMessage() was called twice with any string...");

*verify*(mockApi, *times*(2)).logMessage(ArgumentMatchers.*anyString*());

System.***out***.println("Test: testProcessDataLogsCorrectly PASSED.");

}

@Test

**public** **void** testMethodNotCalled() {

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

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

System.***out***.println("Test: Verifying mockApi.getStatusCode() was never called...");

*verify*(mockApi, *never*()).getStatusCode();

System.***out***.println("Test: testMethodNotCalled PASSED.");

}

}

***Console Output-***

Test: Calling service.processData('Valid Data')...

Test: Verifying mockApi.logMessage('Processing data: Valid Data') was called once...

Test: Calling service.processData('') for warning...

Test: Verifying mockApi.logMessage('Warning: No data to process.') was called once...

Test: Verifying mockApi.logMessage('Error') was never called...

Test: Verifying mockApi.logMessage() was called twice with any string...

Test: testProcessDataLogsCorrectly PASSED.

Test: Verifying mockApi.getStatusCode() was never called...

Test: testMethodNotCalled PASSED.

Test: Calling service.fetchData()...

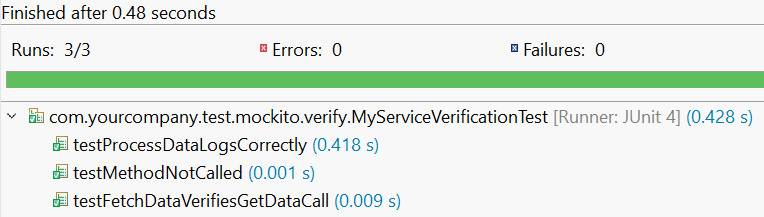
MyService: Attempting to fetch data from ExternalApi...

Test: Verifying mockApi.getData() was called once...

Test: Verifying mockApi.logMessage('Fetching data initiated.') was called once...

Test: testFetchDataVerifiesGetDataCall PASSED.

***JUnit Console-***

******

**Skill: Logging using SLF4J (Java)**

**Hands-on Question: Exercise 1: Logging Error Messages and Warning Levels**

**Introduction to SLF4J and Logback**

**SLF4J (Simple Logging Facade for Java):** SLF4J acts as a thin layer (a "facade") between your application code and the underlying logging framework. This design pattern offers several benefits:

* **Decoupling:** Your code doesn't need to directly depend on a specific logging implementation (e.g., Logback, Log4j). You use SLF4J's API, and at runtime, it "binds" to the chosen logging framework.
* **Flexibility:** You can switch logging implementations later without changing your application code, simply by changing the binding JARs in your project.
* **Simplicity:** It provides a simple and consistent API for logging messages at different levels (ERROR, WARN, INFO, DEBUG, TRACE).

**Logback:** Logback is a modern, fast, and flexible logging framework designed as a successor to Log4j. It's often chosen as the underlying implementation when using SLF4J. Logback offers:

* **High Performance:** It's designed for speed.
* **Flexible Configuration:** It uses an XML configuration file (logback.xml) to control where logs go (console, file, database), what format they have, and what logging levels are active.
* **Automatic Reloading:** It can detect changes to its configuration file and reload them automatically.

Together, SLF4J provides the clean API for your code, and Logback handles the actual logging mechanism and configuration.

**Exercise 1: Logging Error Messages and Warning Levels**

**Scenario:** You need to write a Java application that demonstrates logging error messages and warning levels using SLF4J.

***Pom.xml-***  
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.yourcompany</groupId> <artifactId>JUnitExampleProject</artifactId> <version>1.0.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope> </dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>4.11.0</version> <scope>test</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version> <configuration>

<source>${maven.compiler.source}</source>

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

***Logging.java-***

**package** com.yourcompany.logging;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**public** **class** Logging {

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

**public** **static** **void** main(String[] args) {

System.***out***.println("--- Starting LoggingExample Application ---");

***logger***.trace("This is a TRACE message.");

***logger***.debug("This is a DEBUG message.");

***logger***.info("This is an INFO message.");

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

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

String user = "Alice";

**int** id = 123;

***logger***.error("Failed to process request for user {} with ID {}.", user, id);

**try** {

**int** result = 10 / 0;

} **catch** (ArithmeticException e) {

***logger***.error("An arithmetic error occurred during calculation.", e);

}

System.***out***.println("--- LoggingExample Application Finished ---");

}

}

***Logback.xml-***

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<configuration>

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

<encoder>

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

</encoder>

</appender>

<appender name=*"FILE"* class=*"ch.qos.logback.core.FileAppender"*>

<file>logs/application.log</file>

<encoder>

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

</encoder>

</appender>

<root level=*"INFO"*>

<appender-ref ref=*"CONSOLE"* />

<appender-ref ref=*"FILE"* />

</root>

</configuration>

***Console Output-***

--- Starting LoggingExample Application ---

2025-06-27 11:09:47.496 [main] INFO com.yourcompany.logging.Logging - This is an INFO message.

2025-06-27 11:09:47.500 [main] WARN com.yourcompany.logging.Logging - This is a WARNING message.

2025-06-27 11:09:47.500 [main] ERROR com.yourcompany.logging.Logging - This is an ERROR message.

2025-06-27 11:09:47.500 [main] ERROR com.yourcompany.logging.Logging - Failed to process request for user Alice with ID 123.

2025-06-27 11:09:47.504 [main] ERROR com.yourcompany.logging.Logging - An arithmetic error occurred during calculation.

java.lang.ArithmeticException: / by zero

at com.yourcompany.logging.Logging.main(Logging.java:19)

--- LoggingExample Application Finished ---