WEEK-2_JavaFSE_HandsOn

PL/SQL

Exercise 1: Control Structures

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.

```
CREATE TABLE Customers (
    CustomerID NUMBER PRIMARY KEY,
    Name VARCHAR2(100),
    DOB DATE,
    Balance NUMBER,
    LastModified DATE
);

DECLARE

    TYPE id_list IS TABLE OF NUMBER;
    TYPE name_list IS TABLE OF VARCHAR2(100);
    TYPE dob_list IS TABLE OF DATE;


    v_ids   id_list;
    v_names name_list;
    v_dobs  dob_list;
    v_age   NUMBER;
BEGIN

    SELECT CustomerID, Name, DOB
    BULK COLLECT INTO v_ids, v_names, v_dobs
    FROM Customers;


    FOR i IN 1 .. v_ids.COUNT LOOP
        v_age := TRUNC(MONTHS_BETWEEN(SYSDATE, v_dobs(i))
/ 12);

        IF v_age > 60 THEN
            UPDATE Loans
            SET InterestRate = InterestRate - 1
            WHERE CustomerID = v_ids(i);
```

```
            DBMS_OUTPUT.PUT_LINE('Discount is applied for
' || v_names(i) ||
                                 ' Age: ' || v_age);
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('Discount process is finished.');
END;
```

OUTPUT:

```
Discount is applied for Preethi Age: 110
Discount process is finished.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.086
```

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.

```
ALTER TABLE Customers ADD IsVIP CHAR(1);

SET SERVEROUTPUT ON;

DECLARE
    TYPE id_list IS TABLE OF Customers.CustomerID%TYPE;
    TYPE balance_list IS TABLE OF Customers.Balance%TYPE;

    v_ids     id_list;
    v_balances balance_list;
BEGIN
    SELECT CustomerID, Balance
    BULK COLLECT INTO v_ids, v_balances
    FROM Customers;
    FOR i IN 1 .. v_ids.COUNT LOOP
        IF v_balances(i) > 10000 THEN
            UPDATE Customers
            SET IsVIP = 'Y'
            WHERE CustomerID = v_ids(i);
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('VIP status updated for eligible
customers.');
END;/
```

```
SQL> ALTER TABLE Customers ADD IsVIP CHAR(1)


Table CUSTOMERS altered.

Elapsed: 00:00:00.033


VIP status updated for eligible customers.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.012
```

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.

```
SET SERVEROUTPUT ON;

DECLARE
    TYPE loan_id_list IS TABLE OF Loans.LoanID%TYPE;
    TYPE customer_name_list IS TABLE OF
Customers.Name%TYPE;
    TYPE end_date_list IS TABLE OF Loans.EndDate%TYPE;

    v_loan_ids       loan_id_list;
    v_customer_names customer_name_list;
    v_due_dates      end_date_list;
BEGIN
    SELECT l.LoanID, c.Name, l.EndDate
    BULK COLLECT INTO v_loan_ids, v_customer_names,
v_due_dates
    FROM Loans l
    JOIN Customers c ON l.CustomerID = c.CustomerID
    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 2000
;

    FOR i IN 1 .. v_loan_ids.COUNT LOOP
        DBMS_OUTPUT.PUT_LINE(
            'Dear ' || v_customer_names(i) ||
```

```
            ', your loan with Loan ID ' || v_loan_ids(i)
||
            ' is due on ' || TO_CHAR(v_due_dates(i), 'DD-
Mon-YYYY') ||
            '. Kindly ensure timely repayment to avoid
penalties.'
        );
    END LOOP;

    IF v_loan_ids.COUNT = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No loans are due within the
next 30 days.');
    END IF;
END;
/
```

```
Dear Jane Smith, your loan with Loan ID 2 is due on 28-Jun-2028. Kindly ensure timely repayment to avoid penalties.
Dear Moorthi, your loan with Loan ID 3 is due on 28-Jun-2029. Kindly ensure timely repayment to avoid penalties.
Dear Murugan, your loan with Loan ID 4 is due on 28-Jun-2030. Kindly ensure timely repayment to avoid penalties.
Dear Muthuraj, your loan with Loan ID 6 is due on 28-Jun-2028. Kindly ensure timely repayment to avoid penalties.
Dear Sharavanan, your loan with Loan ID 7 is due on 28-Jun-2029. Kindly ensure timely repayment to avoid penalties.
Dear Kannan, your loan with Loan ID 8 is due on 28-Jun-2030. Kindly ensure timely repayment to avoid penalties.
Dear Preethi, your loan with Loan ID 9 is due on 28-Dec-2027. Kindly ensure timely repayment to avoid penalties.
Dear Manasa, your loan with Loan ID 10 is due on 28-Jun-2028. Kindly ensure timely repayment to avoid penalties.
```

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Exercise 2: Error Handling

Scenario 1: Handle exceptions during fund transfers between accounts.
Question: Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

```
CREATE OR REPLACE PROCEDURE SafeTransferFunds (
    p_from_account_id IN Accounts.AccountID%TYPE,
    p_to_account_id   IN Accounts.AccountID%TYPE,
    p_amount          IN NUMBER
) IS
```

```
    v_balance_from NUMBER;
BEGIN
    SELECT Balance INTO v_balance_from
    FROM Accounts
    WHERE AccountID = p_from_account_id;
    IF v_balance_from < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient
funds in source account.');
    END IF;
    UPDATE Accounts
    SET Balance = Balance - p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_from_account_id;
    UPDATE Accounts
    SET Balance = Balance + p_amount,
        LastModified = SYSDATE
    WHERE AccountID = p_to_account_id;

    COMMIT;
    DBMS_OUTPUT.PUT_LINE('Funds transferred
successfully.');
EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('Transfer failed: ' ||
SQLERRM);
END;
EXEC SafeTransferFunds(1, 2, 500);
```

```
Procedure SAFETRANSFERFUNDS compiled

Elapsed: 00:00:00.018
```

```
SQL> EXEC SafeTransferFunds(1, 2, 500)


Funds transferred successfully.
```

Scenario 2: Manage errors when updating employee salaries.
Question: Write a stored procedure **UpdateSalary** that increases the salary of and employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

```
CREATE OR REPLACE PROCEDURE UpdateSalary (
    p_emp_id      IN Employees.EmployeeID%TYPE,
    p_percentage  IN NUMBER  -- For example: 10 for 10%
) IS
    v_current_salary Employees.Salary%TYPE;
    v_updated_salary NUMBER;
BEGIN
    -- Try to fetch the employee's current salary
    SELECT Salary INTO v_current_salary
    FROM Employees
    WHERE EmployeeID = p_emp_id;

    -- Calculate the new salary
    v_updated_salary := v_current_salary +
(v_current_salary * p_percentage / 100);

    -- Update the employee's salary
    UPDATE Employees
    SET Salary = v_updated_salary
    WHERE EmployeeID = p_emp_id;

    DBMS_OUTPUT.PUT_LINE('Employee ID ' || p_emp_id || '
salary updated by ' || p_percentage ||
                         '%. New salary: ₹' ||
v_updated_salary);

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('Error: No employee found
with ID ' || p_emp_id);
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Unexpected error: ' ||
SQLERRM);
END;
/
```

```
SQL> CREATE OR REPLACE PROCEDURE UpdateSalary (
        p_emp_id      IN Employees.EmployeeID%TYPE,
        p_percentage  IN NUMBER  -- For example: 10 for 10%
    ) IS...
Show more...
```

```
Procedure UPDATESALARY compiled

Elapsed: 00:00:00.015
```

```
BEGIN
    UpdateSalary(2, 10);
END;
/
BEGIN
    UpdateSalary(99, 15); END;
/
```

```
Elapsed: 00:00:00.015
```

```
SQL> BEGIN
        UpdateSalary(99, 15);  -- No employee with ID 99
    END;


Error: No employee found with ID 99


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.085
```

```
SQL> BEGIN
        UpdateSalary(2, 10);  -- Increase salary of employee ID 2 by 10%
    END;


Employee ID 2 salary updated by 10%. New salary: ₹72600


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

Scenario 3: Ensure data integrity when adding a new customer.
Question: Write a stored procedure **AddNewCustomer** that inserts a new customer into the Customers table. If a customer with the same ID already exists, handle the exception by logging an error and preventing the insertion.

```
CREATE OR REPLACE PROCEDURE AddNewCustomer (
    p_customer_id   IN Customers.CustomerID%TYPE,
    p_name          IN Customers.Name%TYPE,
    p_dob           IN Customers.DOB%TYPE,
    p_balance       IN Customers.Balance%TYPE
) IS
BEGIN

    INSERT INTO Customers (
        CustomerID,
        Name,
        DOB,
        Balance,
        LastModified
    ) VALUES (
        p_customer_id,
        p_name,
        p_dob,
        p_balance,
        SYSDATE
    );

    DBMS_OUTPUT.PUT_LINE('Customer "' || p_name || '"
added successfully with ID: ' || p_customer_id);

EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Customer ID ' ||
p_customer_id || ' already exists. Insertion prevented.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Unexpected error while
adding customer: ' || SQLERRM);
END;
/
```

```
SQL> CREATE OR REPLACE PROCEDURE AddNewCustomer (
        p_customer_id   IN Customers.CustomerID%TYPE,
        p_name          IN Customers.Name%TYPE,
        p_dob           IN Customers.DOB%TYPE,...
Show more...
```

```
Procedure ADDNEWCUSTOMER compiled

Elapsed: 00:00:00.022
```

```
BEGIN
    AddNewCustomer(11, 'Ravi Kumar', TO_DATE('1989-03-20',
'YYYY-MM-DD'), 9000);
END;
/


BEGIN
    AddNewCustomer(14, 'Anand', TO_DATE('1990-01-01',
'YYYY-MM-DD'), 5000);
END;
/
```

```
SQL> BEGIN
         AddNewCustomer(11, 'Ravi Kumar', TO_DATE('1989-03-20', 'YYYY-MM-DD'), 9000);
     END;


Error: Customer ID 11 already exists. Insertion prevented.


PL/SQL procedure successfully completed.
```

```
SQL> BEGIN
         AddNewCustomer(14, 'Anand', TO_DATE('1990-01-01', 'YYYY-MM-DD'), 5000);
     END;


Customer "Anand" added successfully with ID: 14


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.011
```

Exercise 3: Stored Procedures

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.

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
    v_updated_count NUMBER := 0;
BEGIN
    UPDATE Accounts
    SET Balance = Balance + (Balance * 0.01),
```

```
        LastModified = SYSDATE
    WHERE AccountType = 'Savings';


    v_updated_count := SQL%ROWCOUNT;

    DBMS_OUTPUT.PUT_LINE('Monthly interest processed for
' || v_updated_count || ' savings account(s).');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error applying monthly
interest: ' || SQLERRM);
END;
/

BEGIN
    ProcessMonthlyInterest;
END;
/
```

```
SQL> BEGIN
        ProcessMonthlyInterest;
    END;


Monthly interest processed for 2 savings account(s).


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014
```

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.

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
    p_department IN Employees.Department%TYPE,
    p_bonus_percent IN NUMBER
) IS
    v_updated_count NUMBER := 0;
BEGIN
        UPDATE Employees
    SET Salary = Salary + (Salary * p_bonus_percent / 100)
    WHERE Department = p_department;
```

```
    v_updated_count := SQL%ROWCOUNT;

    IF v_updated_count > 0 THEN
        DBMS_OUTPUT.PUT_LINE(v_updated_count || '
employees in department "' ||
                                p_department || '" received
a ' || p_bonus_percent || '% bonus.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No employees found in
department "' || p_department || '".');
    END IF;

    COMMIT;

END;
/

BEGIN
    UpdateEmployeeBonus('HR', 5);
    UPDATEEMPLOYEEBONUS('IT', 10);
    UPDATEEMPLOYEEBONUS('Finance', 12);
END;
/
```

```
Procedure UPDATEEMPLOYEEBONUS compiled

Elapsed: 00:00:00.003
```

```
2 employees in department "HR" received a 5% bonus.
2 employees in department "IT" received a 10% bonus.
2 employees in department "Finance" received a 12% bonus.


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.

```
CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account_id IN Accounts.AccountID%TYPE,
    p_to_account_id   IN Accounts.AccountID%TYPE,
    p_amount          IN NUMBER
) IS
    v_from_balance   NUMBER;
    v_from_customer  NUMBER;
    v_to_customer    NUMBER;
BEGIN
    SELECT Balance, CustomerID
    INTO v_from_balance, v_from_customer
    FROM Accounts
    WHERE AccountID = p_from_account_id;
    SELECT CustomerID
    INTO v_to_customer
    FROM Accounts
    WHERE AccountID = p_to_account_id;
    IF v_from_customer = v_to_customer THEN
        IF v_from_balance >= p_amount THEN
            UPDATE Accounts
            SET Balance = Balance - p_amount,
                LastModified = SYSDATE
            WHERE AccountID = p_from_account_id;

            UPDATE Accounts
            SET Balance = Balance + p_amount,
                LastModified = SYSDATE
            WHERE AccountID = p_to_account_id;

            COMMIT;
            DBMS_OUTPUT.PUT_LINE('Transfer successful.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Insufficient balance.
Transfer not completed.');
        END IF;
    ELSE
        DBMS_OUTPUT.PUT_LINE('Transfer failed: Accounts
do not belong to the same customer.');
    END IF;
END;
/

BEGIN
```

```
    TransferFunds(1, 11, 200);
    TransferFunds(1, 11, 10000);  -- ✕ Not enough balance
in Account 1
    TransferFunds(1,3,200);
END;
/
```

```
Procedure TRANSFERFUNDS compiled

Elapsed: 00:00:00.022

Transfer successful.
Insufficient balance. Transfer not completed.
Transfer failed: Accounts do not belong to the same customer.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.014
```

Exercise 4: Functions

Scenario 1: Calculate the age of customers for eligibility checks.
Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

```
CREATE OR REPLACE FUNCTION CalculateAge (
    p_dob IN DATE
) RETURN NUMBER IS
    v_age NUMBER;
BEGIN
    v_age := TRUNC(MONTHS_BETWEEN(SYSDATE, p_dob) / 12);
    RETURN v_age;
END;
/
SELECT CalculateAge(TO_DATE('1990-06-20', 'YYYY-MM-DD'))
AS age FROM dual;
SELECT Name, DOB, CalculateAge(DOB) AS Age
FROM Customers;
```

	AGE
1	35

	NAME	DOB	AGE
1	John Doe	5/15/1985, 12:00:0(	40
2	Jane Smith	7/20/1990, 12:00:0(	34
3	Moorthi	9/12/1983, 12:00:0(	41
4	Murugan	4/28/1990, 12:00:0(	35
5	Muthuraj	6/20/1970, 12:00:0(	55
6	Sharavanan	1/15/1997, 12:00:0(	28
7	Kannan	6/12/1990, 12:00:0(	35
8	Preethi	3/11/1915, 12:00:0(	110
9	Manasa	10/3/1999, 12:00:0(	25

Scenario 2: The bank needs to compute the monthly installment for a loan.
Question: Write a function **CalculateMonthlyInstallment** that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

```
CREATE OR REPLACE FUNCTION
CalculateMonthlyInstallmentFlat (
    p_loan_amount      IN NUMBER,
    p_annual_interest IN NUMBER,
    p_duration_years   IN NUMBER
) RETURN NUMBER IS
    v_total_amount    NUMBER;
    v_num_months      NUMBER;
    v_simple_interest NUMBER;
BEGIN
    v_simple_interest := (p_loan_amount *
p_annual_interest * p_duration_years) / 100;


    v_total_amount := p_loan_amount + v_simple_interest;
```

```
    v_num_months := p_duration_years * 12;
    RETURN ROUND(v_total_amount / v_num_months, 2);
END;
/

SELECT
    LoanID,
    CustomerID,
    LoanAmount,
    InterestRate,
    StartDate,
    EndDate,
    CalculateMonthlyInstallmentFlat(
        LoanAmount,
        InterestRate,
        ROUND(MONTHS_BETWEEN(EndDate, StartDate) / 12, 2)
    ) AS Monthly_EMI
FROM Loans;
```

	LOANID	CUSTOMERID	LOANAMOUNT	INTERESTRATE	STARTDATE	ENDDATE	MONTHLY_EMI
1	2	2	10000	6.5	6/28/2025,	6/28/2028,	331.94
2	3	3	20000	7	6/28/2025,	6/28/2029,	533.33
3	4	4	50000	8.5	6/28/2025,	6/28/2030,	1187.5
4	6	6	12000	5.5	6/28/2025,	6/28/2028,	388.33
5	7	7	18000	7.2	6/28/2025,	6/28/2029,	483
6	8	8	22000	6.75	6/28/2025,	6/28/2030,	490.42
7	9	9	9000	0.8	6/28/2025,	12/28/2027,	306
8	10	10	16000	6.25	6/28/2025,	6/28/2028,	527.78

Scenario 3: Check if a customer has sufficient balance before making a transaction.
Question: Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

```
CREATE OR REPLACE FUNCTION HasSufficientBalance (
    p_account_id IN Accounts.AccountID%TYPE,
    p_amount     IN NUMBER
) RETURN BOOLEAN IS
    v_balance NUMBER := 0;
BEGIN
        SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_account_id;
    RETURN v_balance >= p_amount;
END;
/
BEGIN
    IF HasSufficientBalance(1, 1000) THEN
        DBMS_OUTPUT.PUT_LINE(' Balance is sufficient.');
    ELSE
        DBMS_OUTPUT.PUT_LINE(' Balance is NOT
sufficient.');
    END IF;
END;
/

BEGIN
    IF HasSufficientBalance(2, 1500) THEN
        DBMS_OUTPUT.PUT_LINE('Balance is sufficient.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Balance is NOT
sufficient.');
    END IF;
END;
/
```

OUTPUT:

```
SQL> BEGIN
        IF HasSufficientBalance(1, 1000) THEN
            DBMS_OUTPUT.PUT_LINE(' Balance is sufficient.');
        ELSE...
Show more...
```

```
Balance is NOT sufficient.
```

```
SQL> BEGIN
        IF HasSufficientBalance(2, 1500) THEN
            DBMS_OUTPUT.PUT_LINE('Balance is sufficient.');
        ELSE...
Show more...
```

```
Balance is sufficient.
```

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

Exercise 5: Triggers

Scenario 1: Automatically update the last modified date when a customer's record is updated.
Question: Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.

```
CREATE OR REPLACE TRIGGER UpdateCustomerLastModified
BEFORE UPDATE ON Customers
FOR EACH ROW
BEGIN
    :NEW.LastModified := SYSDATE;
END;
/
SELECT LastModified FROM Customers WHERE CustomerID = 1;
UPDATE Customers
SET Name = 'Aman'
WHERE CustomerID = 1;

SELECT LastModified FROM Customers WHERE CustomerID = 1;
```

OUTPUT:

```
SQL> UPDATE Customers
     SET Name = 'Aman'
     WHERE CustomerID = 1



1 row updated.
```

	LASTMODIFIED
1	6/28/2025, 2:58:57

Scenario 2: Maintain an audit log for all transactions.
Question: Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

```
CREATE TABLE AuditLog (
    AuditID NUMBER PRIMARY KEY,
    TransactionID NUMBER,
    AccountID NUMBER,
    ActionType VARCHAR2(20),
    LogDate DATE,
    FOREIGN KEY (TransactionID) REFERENCES
Transactions(TransactionID),
    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)
);
```

```
Table AUDITLOG created.

Elapsed: 00:00:00.021
```

```
CREATE OR REPLACE TRIGGER LogTransaction
AFTER INSERT ON Transactions
FOR EACH ROW
BEGIN
```

```
    INSERT INTO AuditLog (
        AuditID,
        TransactionID,
        AccountID,
        ActionType,
        LogDate
    ) VALUES (
        AuditLog_seq.NEXTVAL,
        :NEW.TransactionID,
        :NEW.AccountID,
        'INSERT',
        SYSDATE
    );
END;
/
```

```
Trigger LOGTRANSACTION compiled

Elapsed: 00:00:00.021
```

```
INSERT INTO Transactions (TransactionID, AccountID,
TransactionDate, Amount, TransactionType)
VALUES (10, 1, SYSDATE, 1000, 'Deposit');
```

```
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
     VALUES (10, 1, SYSDATE, 1000, 'Deposit')

1 row inserted.

Elapsed: 00:00:00.024
```

```
SELECT * FROM AuditLog WHERE TransactionID = 10;
```

	AUDITID	TRANSACTIONID	ACCOUNTID	ACTIONTYPE	LOGDATE
1	1	10	1	INSERT	6/28/2025, 3:04:46

Scenario 3: Enforce business rules on deposits and withdrawals.
Question: Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.
.

```oracle_plsql
CREATE OR REPLACE TRIGGER CheckTransactionRules
BEFORE INSERT ON Transactions
FOR EACH ROW
DECLARE
    v_balance NUMBER;
BEGIN

    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = :NEW.AccountID;
    IF UPPER(:NEW.TransactionType) = 'WITHDRAWAL' THEN
        IF :NEW.Amount > v_balance THEN
            RAISE_APPLICATION_ERROR(-20001, 'Withdrawal
amount exceeds available balance.');
        ELSIF :NEW.Amount <= 0 THEN
            RAISE_APPLICATION_ERROR(-20002, 'Withdrawal
amount must be greater than zero.');
        END IF;
    ELSIF UPPER(:NEW.TransactionType) = 'DEPOSIT' THEN
        IF :NEW.Amount <= 0 THEN
            RAISE_APPLICATION_ERROR(-20003, 'Deposit
amount must be greater than zero.');
        END IF;
    ELSE
        RAISE_APPLICATION_ERROR(-20004, 'Invalid
transaction type. Must be DEPOSIT or WITHDRAWAL.');
    END IF;
END;
/
```

```
Trigger CHECKTRANSACTIONRULES compiled

Elapsed: 00:00:00.019
```

```oracle_plsql
INSERT INTO Transactions (TransactionID, AccountID,
TransactionDate, Amount, TransactionType)
VALUES (20, 1, SYSDATE, 200, 'Deposit');
```

```
SQL> INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)
     VALUES (20, 1, SYSDATE, 200, 'Deposit')


1 row inserted.

Elapsed: 00:00:00.010
```

```
INSERT INTO Transactions (TransactionID, AccountID,
TransactionDate, Amount, TransactionType)
VALUES (21, 1, SYSDATE, 50000, 'Withdrawal');
```

```
ORA-20001: Withdrawal amount exceeds available balance.
ORA-06512: at "SQL_9H5C04KWJSJ6TS87BKEREI8HSD.CHECKTRANSACTIONRULES", line 12
ORA-04088: error during execution of trigger 'SQL_9H5C04KWJSJ6TS87BKEREI8HSD.CHECKTRANSACTIONRULES'

https://docs.oracle.com/error-help/db/ora-20001/
Error at Line: 4 Column: 0
```

Exercise 6: Cursors

Scenario 1: Generate monthly statements for all customers.
Question: Write a PL/SQL block using an explicit cursor **GenerateMonthlyStatements** that retrieves all transactions for the current month and prints a statement for each customer.

```
DECLARE
       CURSOR monthly_transactions_cursor IS
        SELECT
            t.TransactionID,
            t.TransactionDate,
            t.AccountID,
            t.Amount,
            t.TransactionType,
            c.CustomerID,
            c.Name
        FROM Transactions t
        JOIN Accounts a ON t.AccountID = a.AccountID
        JOIN Customers c ON a.CustomerID = c.CustomerID
```

```
        WHERE TO_CHAR(t.TransactionDate, 'YYYY-MM') =
TO_CHAR(SYSDATE, 'YYYY-MM')
        ORDER BY c.CustomerID, t.TransactionDate;
    transaction_record
monthly_transactions_cursor%ROWTYPE;
    previous_customer_id NUMBER := NULL;
BEGIN
    DBMS_OUTPUT.PUT_LINE('===== Monthly Transaction
Statements =====');

    OPEN monthly_transactions_cursor;

    LOOP
        FETCH monthly_transactions_cursor INTO
transaction_record;
        EXIT WHEN monthly_transactions_cursor%NOTFOUND;

        IF previous_customer_id IS NULL OR
previous_customer_id != transaction_record.CustomerID
THEN
            previous_customer_id :=
transaction_record.CustomerID;

            DBMS_OUTPUT.PUT_LINE(CHR(10) || 'Customer
ID   : ' || transaction_record.CustomerID);
            DBMS_OUTPUT.PUT_LINE('Customer Name : ' ||
transaction_record.Name);

        END IF;
        DBMS_OUTPUT.PUT_LINE(
            TO_CHAR(transaction_record.TransactionDate,
'DD-MON-YYYY') || ' | ' ||
            RPAD(transaction_record.TransactionType, 12)
|| ' | ₹' || transaction_record.Amount
        );
    END LOOP;

    CLOSE monthly_transactions_cursor;

    DBMS_OUTPUT.PUT_LINE(CHR(10) || 'End of Statements');
END;
/
```

```
Customer ID   : 1
Customer Name : Aman
23-JUN-2025 | Withdrawal   | ₹500
26-JUN-2025 | Withdrawal   | ₹150
28-JUN-2025 | Deposit      | ₹200
28-JUN-2025 | Deposit      | ₹200
28-JUN-2025 | Deposit      | ₹1000
28-JUN-2025 | Deposit      | ₹200

Customer ID   : 2
Customer Name : Jane Smith
25-JUN-2025 | Deposit      | ₹1000
28-JUN-2025 | Withdrawal   | ₹300

Customer ID   : 3
Customer Name : Moorthi
27-JUN-2025 | Deposit      | ₹700
28-JUN-2025 | Withdrawal   | ₹300

Customer ID   : 4
Customer Name : Murugan
18-JUN-2025 | Deposit      | ₹1200

End of Statements


PL/SQL procedure successfully completed.
```

Scenario 2: Apply annual fee to all accounts.
Question: Write a PL/SQL block using an explicit cursor **ApplyAnnualFee** that deducts an annual maintenance fee from the balance of all accounts.

```
DECLARE

    v_annual_fee CONSTANT NUMBER := 500;

        CURSOR account_cursor IS
        SELECT AccountID, Balance
        FROM Accounts;


    account_rec account_cursor%ROWTYPE;
BEGIN
    OPEN account_cursor;

    LOOP
        FETCH account_cursor INTO account_rec;
```

```
        EXIT WHEN account_cursor%NOTFOUND;


        IF account_rec.Balance >= v_annual_fee THEN
                      UPDATE Accounts
            SET Balance = Balance - v_annual_fee,
                LastModified = SYSDATE
            WHERE AccountID = account_rec.AccountID;

            DBMS_OUTPUT.PUT_LINE('Annual fee of ₹' ||
v_annual_fee || ' applied to Account ID: ' ||
account_rec.AccountID);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Skipped Account ID ' ||
account_rec.AccountID || ': Insufficient balance.');
        END IF;
    END LOOP;

    CLOSE account_cursor;
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error applying annual fees:
' || SQLERRM);
        ROLLBACK;
END;
/
```

```
Skipped Account ID 1: Insufficient balance.
Annual fee of ₹500 applied to Account ID: 2
Annual fee of ₹500 applied to Account ID: 3
Annual fee of ₹500 applied to Account ID: 4
Annual fee of ₹500 applied to Account ID: 11


PL/SQL procedure successfully completed.
```

Scenario 3: Update the interest rate for all loans based on a new policy.
Question: Write a PL/SQL block using an explicit cursor **UpdateLoanInterestRates** that fetches all loans and updates their interest rates based on the new policy.

```
DECLARE
        CURSOR loan_cursor IS
        SELECT LoanID, InterestRate
        FROM Loans;

    loan_rec loan_cursor%ROWTYPE;


    new_rate NUMBER;
BEGIN
    OPEN loan_cursor;

    LOOP
        FETCH loan_cursor INTO loan_rec;
        EXIT WHEN loan_cursor%NOTFOUND;
        IF loan_rec.InterestRate < 6 THEN
            new_rate := loan_rec.InterestRate + 0.5;
        ELSIF loan_rec.InterestRate >= 6 AND
loan_rec.InterestRate <= 10 THEN
            new_rate := loan_rec.InterestRate + 0.25;
        ELSE
            new_rate := loan_rec.InterestRate;
        END IF;
        UPDATE Loans
        SET InterestRate = new_rate
        WHERE LoanID = loan_rec.LoanID;

        DBMS_OUTPUT.PUT_LINE('Loan ID ' ||
loan_rec.LoanID ||
                             ' updated to new interest
rate: ' || new_rate || '%');
    END LOOP;

    CLOSE loan_cursor;
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error updating loan
interest rates: ' || SQLERRM);
        ROLLBACK;
END;
/
```

```
Loan ID 2 updated to new interest rate: 6.75%
Loan ID 3 updated to new interest rate: 7.25%
Loan ID 4 updated to new interest rate: 8.75%
Loan ID 6 updated to new interest rate: 6%
Loan ID 7 updated to new interest rate: 7.45%
Loan ID 8 updated to new interest rate: 7%
Loan ID 9 updated to new interest rate: 1.3%
Loan ID 10 updated to new interest rate: 6.5%
```

Exercise 7: Packages

Scenario 1: Group all customer-related procedures and functions into a package.
Question: Create a package **CustomerManagement** with procedures for adding a new customer, updating customer details, and a function to get customer balance.

```oracle
--Package specification

CREATE OR REPLACE PACKAGE CustomerManagement IS
    PROCEDURE AddCustomer(
        p_customer_id IN Customers.CustomerID%TYPE,
        p_name        IN Customers.Name%TYPE,
        p_dob         IN Customers.DOB%TYPE
    );
    PROCEDURE UpdateCustomerDetails(
        p_customer_id IN Customers.CustomerID%TYPE,
        p_name        IN Customers.Name%TYPE,
        p_dob         IN Customers.DOB%TYPE
    );
    FUNCTION GetCustomerBalance(
        p_customer_id IN Customers.CustomerID%TYPE
    ) RETURN NUMBER;
END CustomerManagement;
/
```

```
Package CUSTOMERMANAGEMENT compiled

Elapsed: 00:00:00.016
```

```oracle_plsql
--Package BODY
CREATE OR REPLACE PACKAGE BODY CustomerManagement IS

    PROCEDURE AddCustomer(
        p_customer_id IN Customers.CustomerID%TYPE,
        p_name        IN Customers.Name%TYPE,
        p_dob         IN Customers.DOB%TYPE
    ) IS
    BEGIN
        INSERT INTO Customers (CustomerID, Name, DOB,
Balance, LastModified)
        VALUES (p_customer_id, p_name, p_dob, 0, SYSDATE);

        DBMS_OUTPUT.PUT_LINE('Customer ' || p_name || '
added successfully.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Customer ID
already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Unexpected error: ' ||
SQLERRM);
    END AddCustomer;


     PROCEDURE UpdateCustomerDetails(
        p_customer_id IN Customers.CustomerID%TYPE,
        p_name        IN Customers.Name%TYPE,
        p_dob         IN Customers.DOB%TYPE
    ) IS
    BEGIN
        UPDATE Customers
        SET Name = p_name,
            DOB = p_dob,
            LastModified = SYSDATE
        WHERE CustomerID = p_customer_id;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No customer found with
the given ID.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Customer details
updated successfully.');
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error updating customer:
' || SQLERRM);
    END UpdateCustomerDetails;
```

```
    FUNCTION GetCustomerBalance(
        p_customer_id IN Customers.CustomerID%TYPE
    ) RETURN NUMBER IS
        v_total_balance NUMBER;
    BEGIN
        SELECT NVL(SUM(Balance), 0)
        INTO v_total_balance
        FROM Accounts
        WHERE CustomerID = p_customer_id;

        RETURN v_total_balance;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RETURN 0;
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error retrieving
balance: ' || SQLERRM);
            RETURN -1;
    END GetCustomerBalance;

END CustomerManagement;
/
```

```
Package Body CUSTOMERMANAGEMENT compiled

Elapsed: 00:00:00.017
```

```
-- Add a new customer
BEGIN
    CustomerManagement.AddCustomer(11, 'Nandhini',
TO_DATE('1995-08-18', 'YYYY-MM-DD'));
END;
/
```

```
Customer Nandhini added successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.007
```

```
-- Update a customer's name and DOB
BEGIN
    CustomerManagement.UpdateCustomerDetails(11,
'Nandhini R.', TO_DATE('1995-08-20', 'YYYY-MM-DD'));
END;
/
```

```
Customer details updated successfully.

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.013
```

```
-- Get a customer's total balance
DECLARE
    v_balance NUMBER;
BEGIN
    v_balance := CustomerManagement.GetCustomerBalance(1);
    DBMS_OUTPUT.PUT_LINE('Total Balance: ₹' || v_balance);
END;
/
```

```
Total Balance: ₹500

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.008
```

Scenario 2: Create a package to manage employee data.
Question: Write a package **EmployeeManagement** with procedures to hire new employees, update employee details, and a function to calculate annual salary.

```
--Package specification
CREATE OR REPLACE PACKAGE EmployeeManagement IS

    PROCEDURE HireEmployee(
        p_emp_id      IN Employees.EmployeeID%TYPE,
        p_name        IN Employees.Name%TYPE,
        p_position    IN Employees.Position%TYPE,
        p_salary      IN Employees.Salary%TYPE,
        p_department IN Employees.Department%TYPE,
        p_hiredate    IN Employees.HireDate%TYPE
```

```
    );


    PROCEDURE UpdateEmployeeDetails(
        p_emp_id      IN Employees.EmployeeID%TYPE,
        p_name        IN Employees.Name%TYPE,
        p_position    IN Employees.Position%TYPE,
        p_salary      IN Employees.Salary%TYPE,
        p_department IN Employees.Department%TYPE,
        p_hiredate    IN Employees.HireDate%TYPE
    );

      FUNCTION CalculateAnnualSalary(
        p_emp_id IN Employees.EmployeeID%TYPE
    ) RETURN NUMBER;
END EmployeeManagement;
/
```

Package EMPLOYEEMANAGEMENT compiled

Elapsed: 00:00:00.015

```
--Package BODY
CREATE OR REPLACE PACKAGE BODY EmployeeManagement IS

    -- Hire a new employee
    PROCEDURE HireEmployee(
        p_emp_id      IN Employees.EmployeeID%TYPE,
        p_name        IN Employees.Name%TYPE,
        p_position    IN Employees.Position%TYPE,
        p_salary      IN Employees.Salary%TYPE,
        p_department IN Employees.Department%TYPE,
        p_hiredate    IN Employees.HireDate%TYPE
    ) IS
    BEGIN
        INSERT INTO Employees (EmployeeID, Name, Position,
Salary, Department, HireDate)
        VALUES (p_emp_id, p_name, p_position, p_salary,
p_department, p_hiredate);

        DBMS_OUTPUT.PUT_LINE('Employee ' || p_name || '
hired successfully.');
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
```

```
            DBMS_OUTPUT.PUT_LINE('Error: Employee ID
already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Unexpected error while
hiring: ' || SQLERRM);
    END HireEmployee;


    PROCEDURE UpdateEmployeeDetails(
        p_emp_id     IN Employees.EmployeeID%TYPE,
        p_name       IN Employees.Name%TYPE,
        p_position   IN Employees.Position%TYPE,
        p_salary     IN Employees.Salary%TYPE,
        p_department IN Employees.Department%TYPE,
        p_hiredate   IN Employees.HireDate%TYPE
    ) IS
    BEGIN
        UPDATE Employees
        SET Name = p_name,
            Position = p_position,
            Salary = p_salary,
            Department = p_department,
            HireDate = p_hiredate
        WHERE EmployeeID = p_emp_id;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No employee found with
the given ID.');
        ELSE
            DBMS_OUTPUT.PUT_LINE('Employee ID ' ||
p_emp_id || ' details updated successfully.');
        END IF;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error updating employee:
' || SQLERRM);
    END UpdateEmployeeDetails;

       FUNCTION CalculateAnnualSalary(
        p_emp_id IN Employees.EmployeeID%TYPE
    ) RETURN NUMBER IS
        v_monthly_salary NUMBER;
        v_annual_salary NUMBER;
    BEGIN
        SELECT Salary INTO v_monthly_salary
        FROM Employees
        WHERE EmployeeID = p_emp_id;

        v_annual_salary := v_monthly_salary * 12;
```

```
        RETURN v_annual_salary;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('Employee not found.');
            RETURN -1;
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error calculating
annual salary: ' || SQLERRM);
            RETURN -1;
    END CalculateAnnualSalary;

END EmployeeManagement;
/
```

```
Package Body EMPLOYEEMANAGEMENT compiled

Elapsed: 00:00:00.023
```

```
-- Hire a new employee
BEGIN
    EmployeeManagement.HireEmployee(10, 'Meera Raj',
'Analyst', 45000, 'Finance', TO_DATE('2022-09-01', 'YYYY-
MM-DD'));
END;
/
```

```
Employee Meera Raj hired successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

```
-- Update employee details
BEGIN
    EmployeeManagement.UpdateEmployeeDetails(10, 'Meera
Raj', 'Sr. Analyst', 48000, 'Finance', TO_DATE('2022-09-
01', 'YYYY-MM-DD'));
END;
/
```

```
Employee ID 10 details updated successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.012
```

```
DECLARE
    v_annual NUMBER;
BEGIN
    v_annual :=
EmployeeManagement.CalculateAnnualSalary(10);
    DBMS_OUTPUT.PUT_LINE('Annual Salary: ₹' || v_annual);
END;
/
```

```
Annual Salary: ₹576000


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.008
```

Scenario 3: Group all account-related operations into a package.
Question: Create a package **AccountOperations** with procedures for opening a new account, closing an account, and a function to get the total balance of a customer across all accounts.

```
--Package Specification
CREATE OR REPLACE PACKAGE AccountOperations IS
    PROCEDURE OpenAccount(
        p_account_id   IN Accounts.AccountID%TYPE,
        p_customer_id  IN Accounts.CustomerID%TYPE,
        p_account_type IN Accounts.AccountType%TYPE,
        p_balance      IN Accounts.Balance%TYPE
    );
    PROCEDURE CloseAccount(
        p_account_id IN Accounts.AccountID%TYPE
    );

      FUNCTION GetTotalBalance(
        p_customer_id IN Accounts.CustomerID%TYPE
    ) RETURN NUMBER;
```

```
END AccountOperations;
/
```

```
Package ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.016
```

```
CREATE OR REPLACE PACKAGE BODY AccountOperations IS


    PROCEDURE OpenAccount(
        p_account_id   IN Accounts.AccountID%TYPE,
        p_customer_id  IN Accounts.CustomerID%TYPE,
        p_account_type IN Accounts.AccountType%TYPE,
        p_balance      IN Accounts.Balance%TYPE
    ) IS
    BEGIN
        INSERT INTO Accounts (AccountID, CustomerID,
AccountType, Balance, LastModified)
        VALUES (p_account_id, p_customer_id,
p_account_type, p_balance, SYSDATE);

        DBMS_OUTPUT.PUT_LINE('Account ID ' ||
p_account_id || ' opened successfully for Customer ID '
|| p_customer_id);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Account ID
already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Unexpected error while
opening account: ' || SQLERRM);
    END OpenAccount;

    -- Close an account
    PROCEDURE CloseAccount(
        p_account_id IN Accounts.AccountID%TYPE
    ) IS
    BEGIN
        DELETE FROM Accounts
        WHERE AccountID = p_account_id;

        IF SQL%ROWCOUNT = 0 THEN
            DBMS_OUTPUT.PUT_LINE('No account found with
Account ID: ' || p_account_id);
        ELSE
            DBMS_OUTPUT.PUT_LINE('Account ID ' ||
p_account_id || ' closed successfully.');
        END IF;
    EXCEPTION
```

```
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error closing account:
' || SQLERRM);
    END CloseAccount;


    FUNCTION GetTotalBalance(
        p_customer_id IN Accounts.CustomerID%TYPE
    ) RETURN NUMBER IS
        v_total_balance NUMBER;
    BEGIN
        SELECT NVL(SUM(Balance), 0)
        INTO v_total_balance
        FROM Accounts
        WHERE CustomerID = p_customer_id;

        RETURN v_total_balance;
    EXCEPTION
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error retrieving total
balance: ' || SQLERRM);
            RETURN -1;
    END GetTotalBalance;

END AccountOperations;
/
```

```
Package Body ACCOUNTOPERATIONS compiled

Elapsed: 00:00:00.017
```

```
-- Open a new account
BEGIN
    AccountOperations.OpenAccount(101, 3, 'Savings',
7500);
END;
/
```

```
Account ID 101 opened successfully for Customer ID 3


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.016
```

```
-- Close an account
BEGIN
    AccountOperations.CloseAccount(101);
END;
/
```

```
Account ID 101 closed successfully.


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.010
```

```
-- Get total balance across all accounts for a customer
DECLARE
    v_total NUMBER;
BEGIN
    v_total := AccountOperations.GetTotalBalance(1);
    DBMS_OUTPUT.PUT_LINE('Total Balance for Customer 1:
₹' || v_total);
END;
/
```

```
Total Balance for Customer 1: ₹500


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.008
```

Unit Testing Mandatory Hands-On

1.JUnit_Basic Testing

Exercise 1: Setting Up JUnit

Project name: testunit1
Group ID: com.example1
ArtifactID: testunit1
Junit version used: JUnit 5

Pom.xml dependency

```
<dependencies>
      <dependency>
        <groupId>org.junit</groupId>
        <artifactId>junit-bom</artifactId>
        <version>5.11.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>

    </dependencies>
```

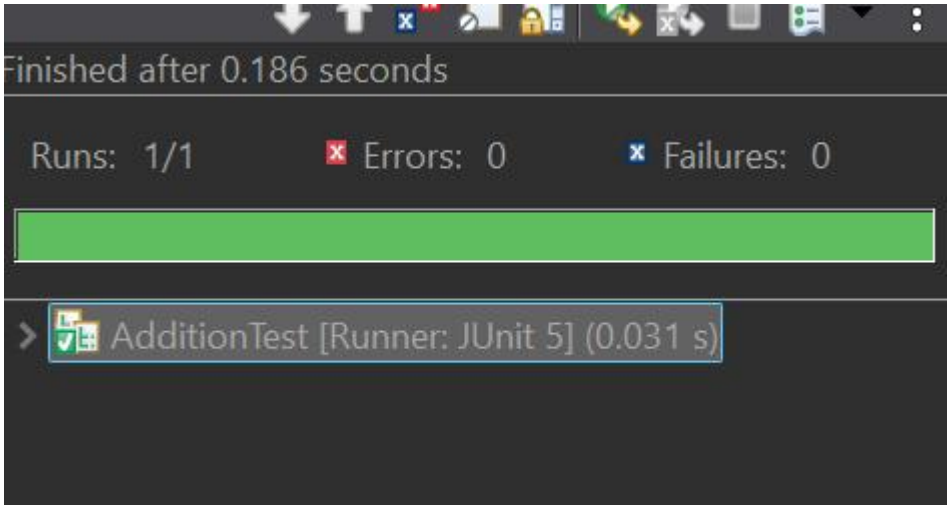
Java class(Addition.java)

```
package com.example1.testunit1;

public class Addition {
       public int add(int a, int b) {
         return a + b;
    }
}
```

AdditionTest.java(JUnit Test class)

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class AdditionTest {
       @Test
    public void testAdd() {
        Addition addition= new Addition();
        assertEquals(7, addition.add(3, 4));
    }
}
```

OUTPUT

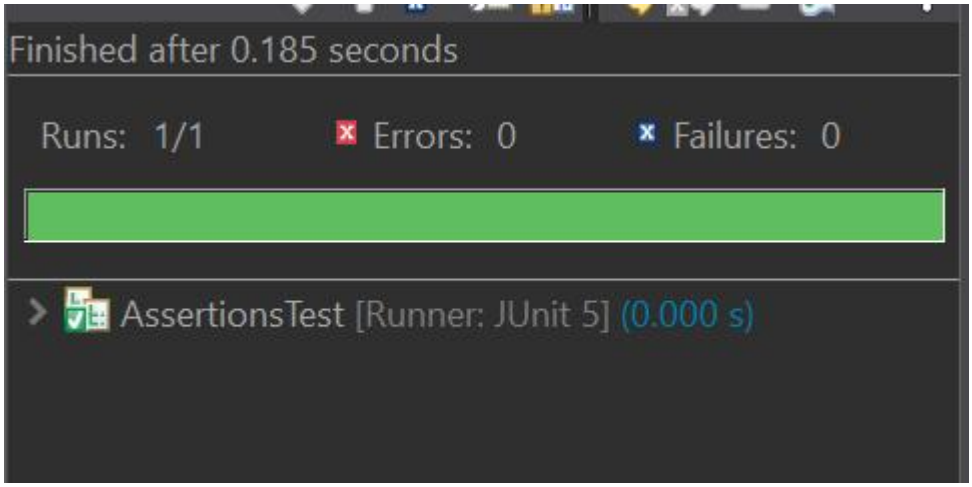


Exercise 3: Assertions in JUnit

AssertionTest.java

```
package com.example1.testunit1;

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

public class AssertionsTest {

    @Test
    public void testAssertions() {
        // Assert equality
        assertEquals(5, 2 + 3);

        // Assert true
        assertTrue(5 > 3);

        // Assert false
        assertFalse(5 < 3);

        // Assert null
        assertNull(null);

        // Assert not null
        assertNotNull(new Object());
    }
}
```

OUTPUT:



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

```
package com.example1.testunit1;

public class Calculator {
        public int add(int a, int b) {
                return a + b;
            }

            public int subtract(int a, int b) {
                return a - b;
            }
}
```

AAAPattern

```
package com.example1.testunit1;

import org.junit.jupiter.api.*;

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

public class CalculatorTest {

    Calculator calculator;

    @BeforeEach
    public void setUp() {
        System.out.println("setUp() method used");
        calculator = new Calculator();  // Arrange
    }

    @AfterEach
    public void tearDown() {
        System.out.println("tearDown() method used");
        calculator = null;
    }
```

```
    @Test
    public void testAdd() {
        // Act
        int result = calculator.add(10, 5);
        // Assert
        assertEquals(15, result);
    }

    @Test
    public void testSubtract() {
        // Act
        int result = calculator.subtract(10, 5);
        // Assert
        assertEquals(5, result);
    }
}
```

OUTPUT:

```
<terminated> CalculatorTest [JUnit] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (29-Jun-2025, 6
setUp() method used
tearDown() method used
setUp() method used
tearDown() method used
```

Mockito Exercises

Exercise - 1: Mocking and Stubbing

ExternalApi.java

```
package com.example1.testunit1;

public interface ExternalApi {

      String getData();
}
```

MyService.java

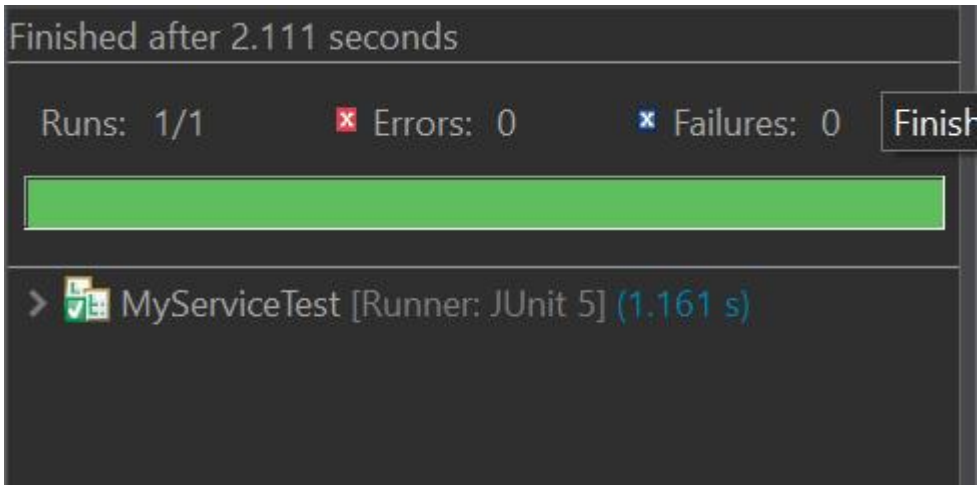
```
package com.example1.testunit1;

public class MyService {
    private ExternalApi api;
```

```java
    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData();
    }
}
```

MyServiceTest.java

```java
package com.example1.testunit1;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import org.mockito.Mockito;

public class MyServiceTest {
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = Mockito.mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");

        MyService service = new MyService(mockApi);
        String result = service.fetchData();

        assertEquals("Mock Data", result);
    }
}
```

OUTPUT:



Exercise - 2: Verifying Interactions

ExternalApi.java

```
package com.example1.testunit1;

public interface ExternalApi {

      String getData();
}
```

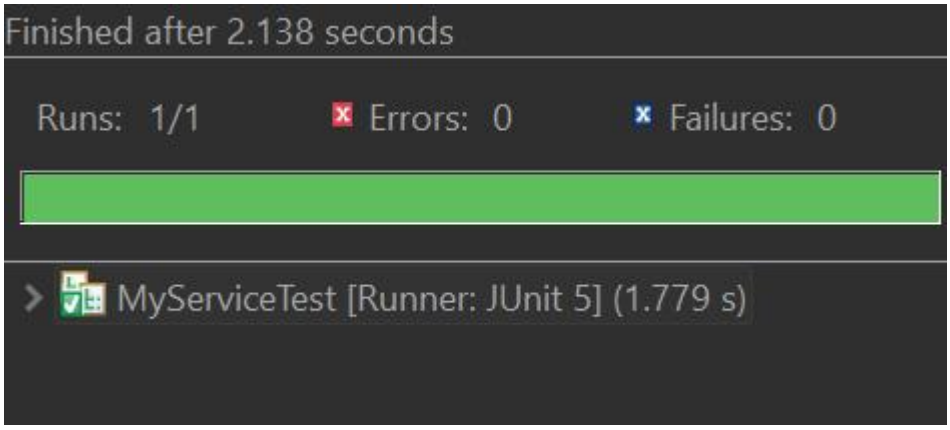
MyService.java

```
package com.example1.testunit1;

public class MyService {
    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchData() {
        return api.getData();
    }
}
```

MyServiceTest.java

```
package com.example1.testunit1;

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();
    }
}
```

OUTPUT:



SL4J Logging Exercises

Exercise - 1: Logging Error messages and Warning levels

LoggingExample.java

```
package com.example1.testunit1;

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:

```
<terminated> LoggingExample [Java Application] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20
04:32:08.601 [main] ERROR com.example1.testunit1.LoggingExample -- This is an error message
04:32:08.622 [main] WARN com.example1.testunit1.LoggingExample -- This is a warning message
```

Additional Hands-On Exercises

JUnit Testing Exercises

Exercise 2: Writing Basic JUnit Test

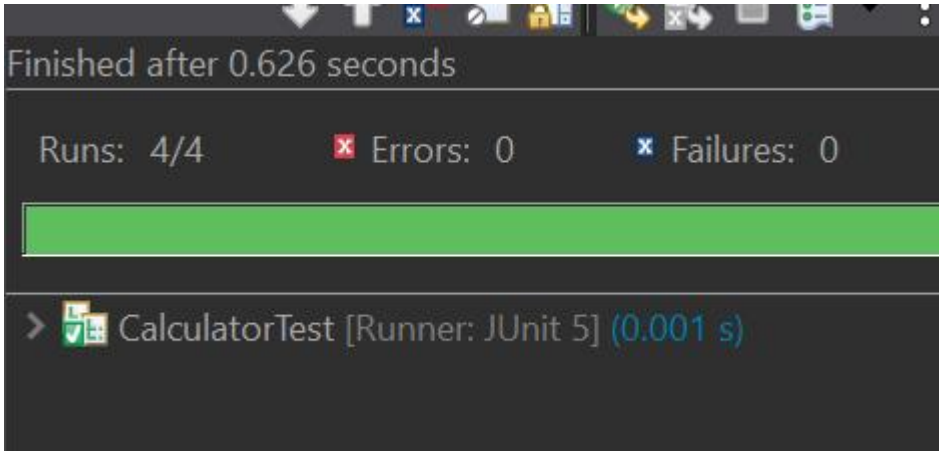
Calculator.java

```java
package com.example1.testunit1;

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) {
                return a / b; // Note: Assumes b is not zero
            }
}
```

CalculatorTest.java

```java
package com.example1.testunit1;

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

public class CalculatorTest {

    Calculator calculator = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(10, calculator.add(6, 4));
    }

    @Test
    public void testSubtract() {
        assertEquals(2, calculator.subtract(6, 4));
    }

    @Test
    public void testMultiply() {
        assertEquals(24, calculator.multiply(6, 4));
    }

    @Test
    public void testDivide() {
        assertEquals(2, calculator.divide(8, 4));
    }
}
```

OUTPUT:



Advanced JUnit Testing Exercises

Exercise 1: Parameterized Tests

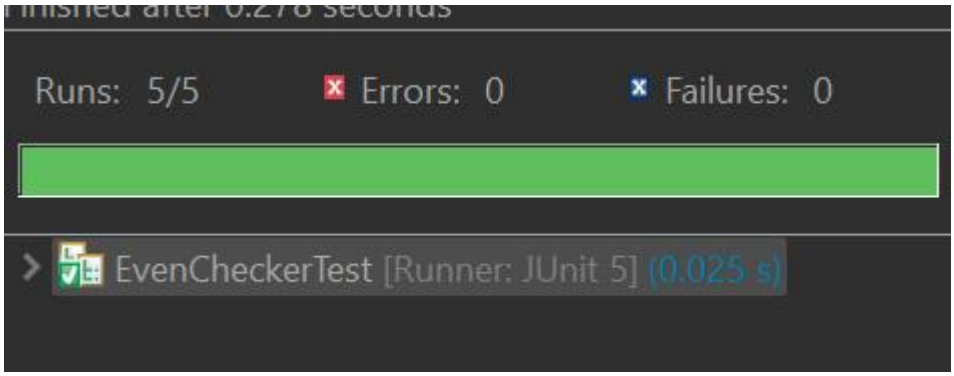
EvenChecker.java

```java
package com.example1.testunit1;

public class EvenChecker {
    public boolean isEven(int number) {
        return number % 2 == 0;
    }
}
```

EvenCheckerTest.java

```java
package com.example1.testunit1;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

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

public class EvenCheckerTest {

    EvenChecker checker = new EvenChecker();

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6, 8, 10})
    public void testIsEven(int number) {
        assertTrue(checker.isEven(number));
    }
}
```

OUTPUT:



Exercise 2: Test Suites and Categories

CalculatorTest.java

```
package com.example1.testunit1;

import org.junit.jupiter.api.*;

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

public class CalculatorTest {

    Calculator calculator = new Calculator();


    @Test
    public void testAdd() {
        // Act
        int result = calculator.add(10, 5);
        // Assert
        assertEquals(15, result);
    }

    @Test
    public void testSubtract() {
        // Act
        int result = calculator.subtract(10, 5);
        // Assert
        assertEquals(5, result);
    }
}
```

EvenCheckerTest.java

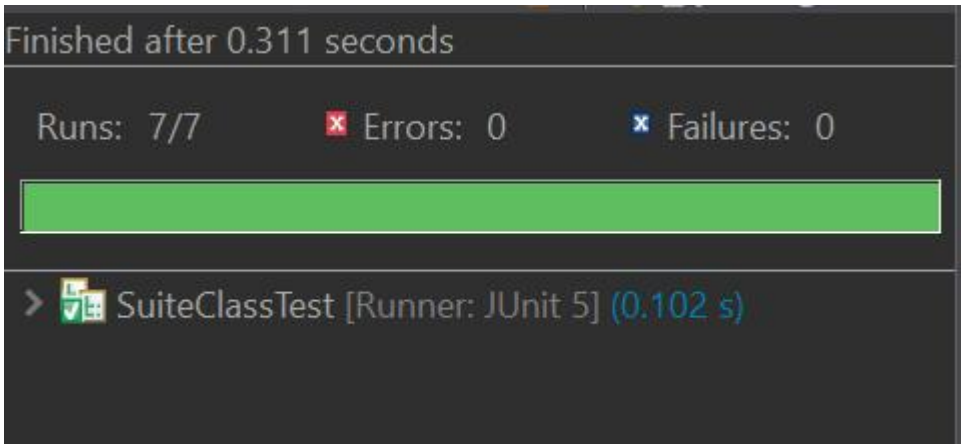
```
package com.example1.testunit1;

import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
```

```java
import static org.junit.jupiter.api.Assertions.assertTrue;

public class EvenCheckerTest {

    EvenChecker checker = new EvenChecker();

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6, 8, 10})
    public void testIsEven(int number) {
        assertTrue(checker.isEven(number));
    }
}
```

SuiteClassTest.java

```java
package com.example1.testunit1;

import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.Suite;

@Suite
@SelectClasses({
    CalculatorTest.class,
    EvenCheckerTest.class
})
public class SuiteClassTest {}
```

OUTPUT:



Exercise - 3: Test Execution Order

OrderedStudentsTests.java

```java
package com.example1.testunit1;

import org.junit.jupiter.api.*;

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

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
```

```java
public class OrderedStudentsTests {

    @Test
    @Order(1)
    void testRegisterStudent() {
        System.out.println("Student registered:");
        String studentName = "Jane Smith";
        assertNotNull(studentName);
    }

    @Test
    @Order(2)
    void testAssignCourse() {
        System.out.println("Course assignment");
        String course = "Programming in java";
        assertEquals("Programming in java", course);
    }

    @Test
    @Order(3)
    void testGenerateIDCard() {
        System.out.println("ID Card Generation");
        int studentId = 927282;
        assertTrue(studentId > 0);
    }
}
```

OUTPUT:

```
<terminated> OrderedStudentsTests (1) [JUnit] C:\Users\admin\.p2\pool\pl
Student registered:
Course assignment
ID Card Generation
```

Finished after 0.193 seconds

Runs: 3/3 Errors: 0 Failures: 0

OrderedStudentsTests [Runner: JUnit 5] (0.033 s)

Exercise - 4: Exception Testing

ExceptionThrower.java

```java
package com.example1.testunit1;

public class ExceptionThrower {
```

```
    public void throwException(String input) {
        if (input == null || input.isEmpty()) {
            throw new IllegalArgumentException("Input cannot be null or
empty!");
        }

        System.out.println("Valid input: " + input);
    }
}
```

ExceptionThrowerTest.java

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;

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

public class ExceptionThrowerTest {

    @Test
    void testThrowsExceptionWhenInputIsNull() {
        ExceptionThrower et = new ExceptionThrower();

        // Assert that an IllegalArgumentException is thrown
        assertThrows(IllegalArgumentException.class, () -> {
            et.throwException(null);
        });
    }

    @Test
    void testThrowsExceptionWhenInputIsEmpty() {
        ExceptionThrower et = new ExceptionThrower();

        assertThrows(IllegalArgumentException.class, () -> {
            et.throwException("");
        });
    }

    @Test
    void testDoesNotThrowExceptionForValidInput() {
        ExceptionThrower et = new ExceptionThrower();

        // This should not throw anything
        assertDoesNotThrow(() -> et.throwException("...."));
    }
}
```

OUTPUT:

```
Finished after 0.218 seconds

Runs: 3/3        Errors: 0        Failures: 0

> ExceptionThrowerTest [Runner: JUnit 5] (0.024 s)

<terminated> ExceptionThrowerTest [JUnit] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.
Valid input: ....
```

Mockito HandsOn Exercises

Exercise - 3: Argument Matching

ExternalApi.java

```java
package com.example1.testunit1;

public interface ExternalApi {

      String getData(String id);
}
```

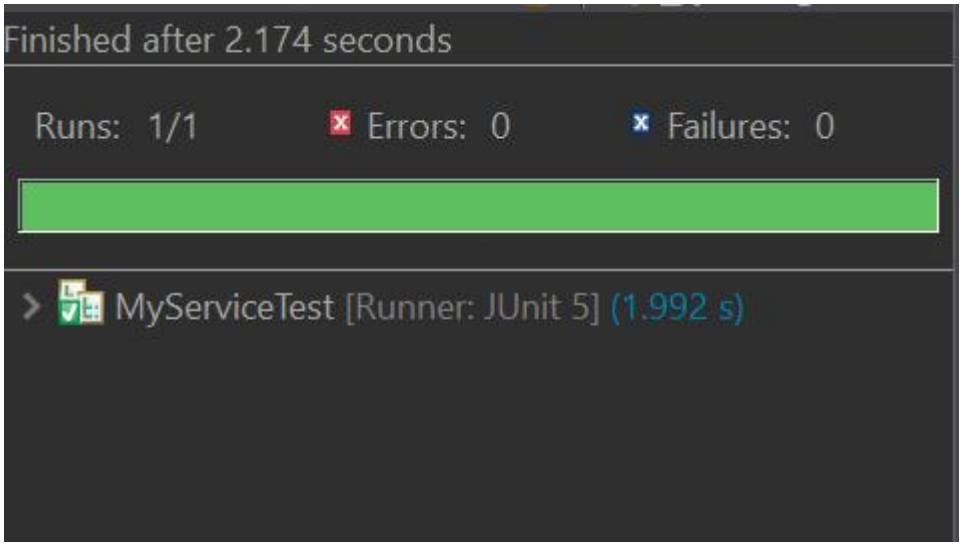
MyService.java

```java
package com.example1.testunit1;

public class MyService {
    private ExternalApi api;

    public MyService(ExternalApi api) {
        this.api = api;
    }

    public String fetchById(String id) {
        return api.getData(id);
    }
}
```

MyServiceTest.java

```java
// File: MyServiceTest.java
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
import static org.mockito.ArgumentMatchers.*;

public class MyServiceTest {

    @Test
    void testFetchById_CalledWithCorrectArgument() {

        ExternalApi mockApi = mock(ExternalApi.class);


        when(mockApi.getData(anyString())).thenReturn("Mocked Data");


        MyService service = new MyService(mockApi);
        String result = service.fetchById("123");


        verify(mockApi).getData(eq("123"));


        assertEquals("Mocked Data", result);
    }
}
```

OUTPUT



Exercise - 4: Handling void methods

Notifier.java

```
package com.example1.testunit1;

public interface Notifier {
    void sendNotification(String message);
}
```

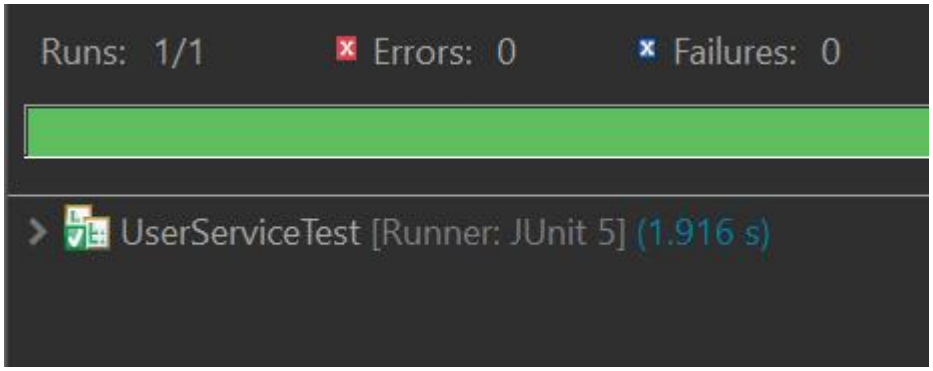
UserService.java

```
package com.example1.testunit1;

public class UserService {
    private Notifier notifier;

    public UserService(Notifier notifier) {
        this.notifier = notifier;
    }

    public void registerUser(String username) {
        notifier.sendNotification("User " + username + " registered
successfully.");
    }
}
```

UserServiceTest.java

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;

public class UserServiceTest {

    @Test
    void testSendNotificationCalled() {
        Notifier mockNotifier = mock(Notifier.class);

        UserService service = new UserService(mockNotifier);

        service.registerUser("John Doe.");

        verify(mockNotifier).sendNotification("User John Doe. registered
successfully.");
    }
}
```

OUTPUT



Exercise - 5: Mocking and Stubbing with Multiple Returns

WeatherApi.java

```
package com.example1.testunit1;

public interface WeatherApi {
    int getTemperature();
}
```

WeatherService.java

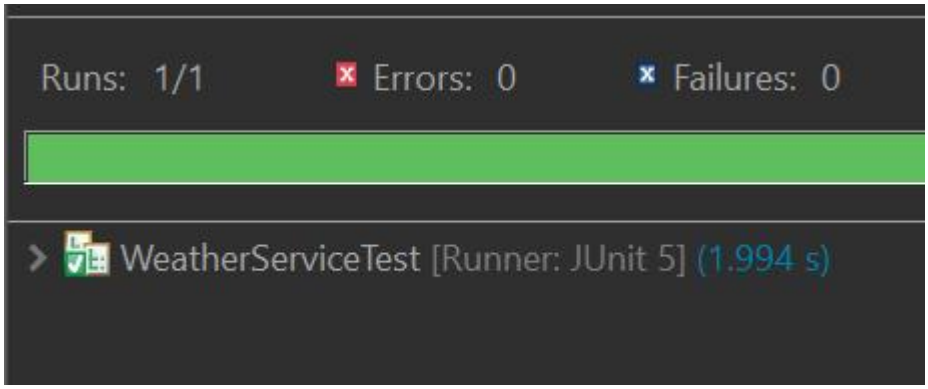
```
package com.example1.testunit1;

public class WeatherService {
    private WeatherApi weatherApi;

    public WeatherService(WeatherApi weatherApi) {
        this.weatherApi = weatherApi;
    }

    public int[] getThreeDayForecast() {
        return new int[] {
            weatherApi.getTemperature(),
            weatherApi.getTemperature(),
            weatherApi.getTemperature()
        };
    }
}
```

WeatherServiceTest.java

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

public class WeatherServiceTest {

    @Test
```

```
    void testGetThreeDayForecast() {
        WeatherApi mockApi = mock(WeatherApi.class);

        when(mockApi.getTemperature()).thenReturn(30, 32, 28);

        WeatherService service = new WeatherService(mockApi);
        int[] forecast = service.getThreeDayForecast();

        assertArrayEquals(new int[]{30, 32, 28}, forecast);
    }
}
```

OUTPUT:



Exercise - 6: Verifying Interaction Order

OrderService.java

```
package com.example1.testunit1;

public interface OrderService {
    void placeOrder(String item);
}
```

PaymentService.java

```
package com.example1.testunit1;

public interface PaymentService {
    void processPayment(double amount);
}
```

FoodOrderingApp.java

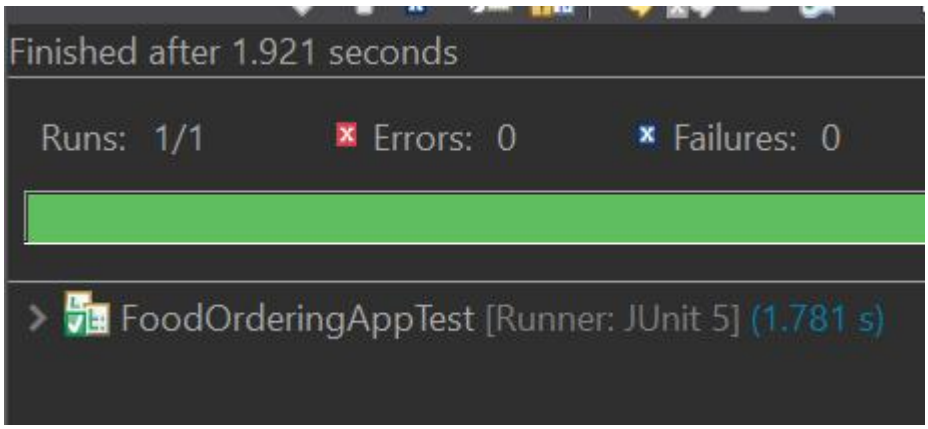
```
package com.example1.testunit1;

public class FoodOrderingApp {
    private OrderService orderService;
    private PaymentService paymentService;

    public FoodOrderingApp(OrderService orderService, PaymentService
paymentService) {
        this.orderService = orderService;
```

```
        this.paymentService = paymentService;
    }

    public void orderFood(String item, double amount) {
        orderService.placeOrder(item);
        paymentService.processPayment(amount);

    }
}
```

FoodOrderingTest.java

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import org.mockito.InOrder;

public class FoodOrderingAppTest {

    @Test
    void testOrderProcessSequence() {
        OrderService orderService = mock(OrderService.class);
        PaymentService paymentService = mock(PaymentService.class);

        FoodOrderingApp app = new FoodOrderingApp(orderService,
paymentService);


        app.orderFood("Pizza", 299.99);


        InOrder inOrder = inOrder(orderService, paymentService);
        inOrder.verify(orderService).placeOrder("Pizza");
        inOrder.verify(paymentService).processPayment(299.99);
    }
}
```

OUTPUT:



Exercise - 7: Handling void methods with Exceptions

```java
package com.example1.testunit1;

public interface Printer {
    void print(String documentName);
}
```

```java
package com.example1.testunit1;

public class PrintManager {
    private Printer printer;

    public PrintManager(Printer printer) {
        this.printer = printer;
    }

    public void printDocument(String name) {
        try {
            printer.print(name);
        } catch (RuntimeException e) {
            System.out.println("Print failed: " + e.getMessage());
        }
    }
}
```

```java
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;

public class PrintManagerTest {

    @Test
    void testPrintThrowsException() {
        Printer mockPrinter = mock(Printer.class);

        doThrow(new RuntimeException("Printer is offline"))
            .when(mockPrinter).print("error");

        PrintManager manager = new PrintManager(mockPrinter);
        manager.printDocument("error");

        verify(mockPrinter).print("error");
    }
}
```

OUTPUT:

```
Finished after 2.009 seconds

Runs: 1/1        Errors: 0        Failures: 0

> PrintManagerTest [Runner: JUnit 5] (1.847 s)

<terminated> PrintManagerTest [JUnit] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (29-Jun-
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
OpenJDK 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been append
Print failed: Printer is offline
```

MOCKITO ADVANCED HANDS-ON EXERCISES

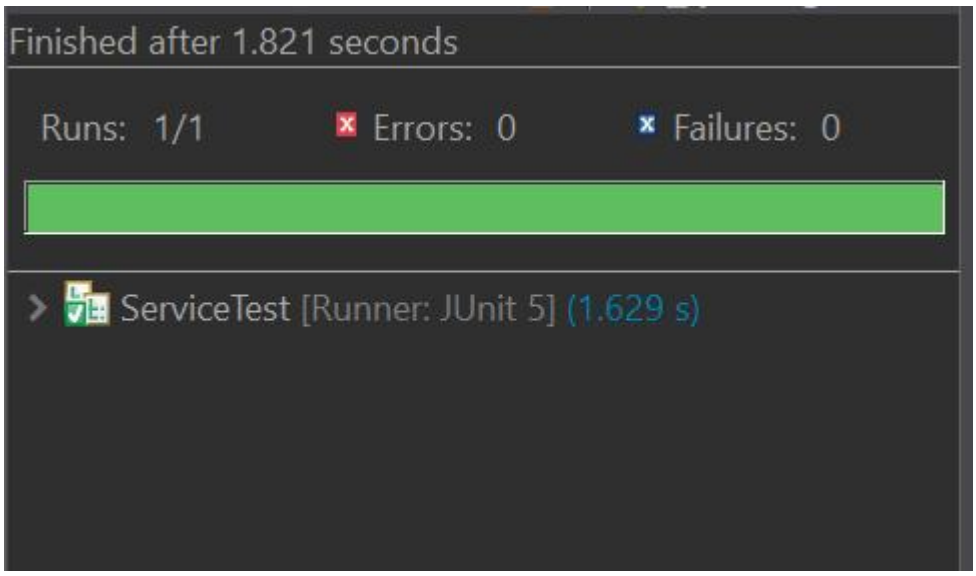
Exercise - 1: Mocking Databases and Repositories

```java
package com.example1.testunit1;

public interface Repository {
    String getData();
}
```

```java
package com.example1.testunit1;

public class Service {
    private Repository repository;

    public Service(Repository repository) {
        this.repository = repository;
    }

    public String processData() {
        String data = repository.getData();
        return "Processed " + data;
    }
}
```

```java
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class ServiceTest {

    @Test
    public void testServiceWithMockRepository() {
        Repository mockRepository = mock(Repository.class);

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

        Service service = new Service(mockRepository);
        String result = service.processData();

        assertEquals("Processed Mock Data", result);
    }
}
```

OUTPUT:



Exercise - 2: Mocking External Services(Restful APIs)

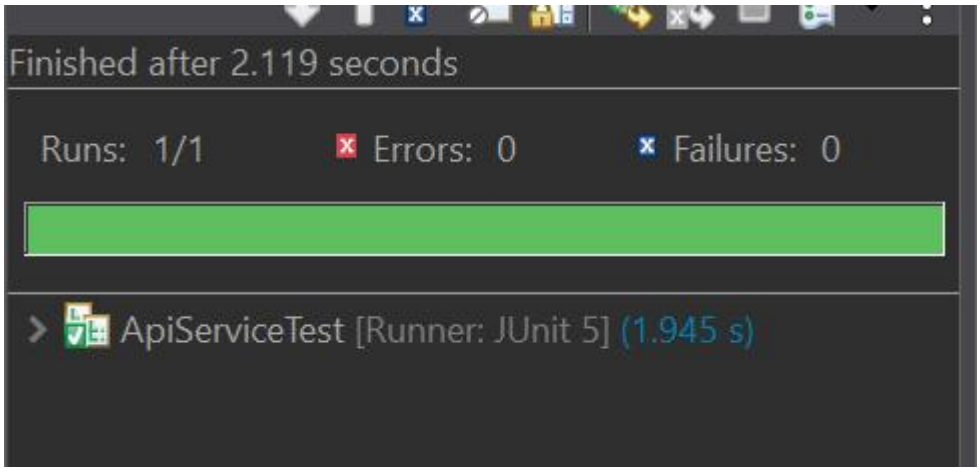
```java
package com.example1.testunit1;

public interface RestClient {
    String getResponse();
}
```

```java
package com.example1.testunit1;
```

```java
public class ApiService {
    private RestClient restClient;

    public ApiService(RestClient restClient) {
        this.restClient = restClient;
    }

    public String fetchData() {
        return "Fetched " + restClient.getResponse();
    }
}
```

```java
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class ApiServiceTest {

    @Test
    public void testServiceWithMockRestClient() {
        RestClient mockRestClient = mock(RestClient.class);

        when(mockRestClient.getResponse()).thenReturn("Mock Response");

        ApiService apiService = new ApiService(mockRestClient);
        String result = apiService.fetchData();

        assertEquals("Fetched Mock Response", result);
    }
}
```

OUTPUT:



Exercise - 3: Mocking File I/O

```java
package com.example1.testunit1;

public interface FileReader {
    String read();
```

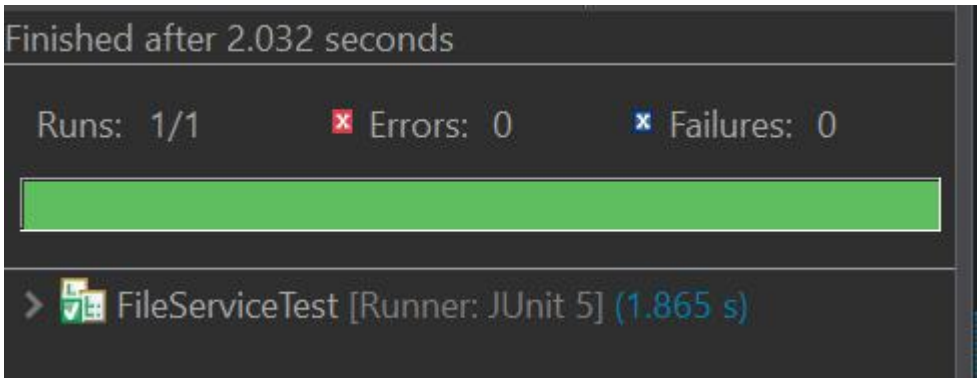
```
}
```

```java
package com.example1.testunit1;

public interface FileWriter {
    void write(String content);
}
```

```java
package com.example1.testunit1;

public class FileService {
    private FileReader fileReader;
    private FileWriter fileWriter;

    public FileService(FileReader fileReader, FileWriter fileWriter) {
        this.fileReader = fileReader;
        this.fileWriter = fileWriter;
    }

    public String processFile() {
        String content = fileReader.read();
        String processed = "Processed " + content;
        fileWriter.write(processed);
        return processed;
    }
}
```

```java
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class FileServiceTest {

    @Test
    public void testServiceWithMockFileIO() {
        FileReader mockFileReader = mock(FileReader.class);
        FileWriter mockFileWriter = mock(FileWriter.class);

        when(mockFileReader.read()).thenReturn("Mock File Content");

        FileService fileService = new FileService(mockFileReader,
mockFileWriter);

        String result = fileService.processFile();
        assertEquals("Processed Mock File Content", result);

        verify(mockFileWriter).write("Processed Mock File Content");
    }
}
```

OUTPUT:



Exercise - 4: Mocking Network Interactions

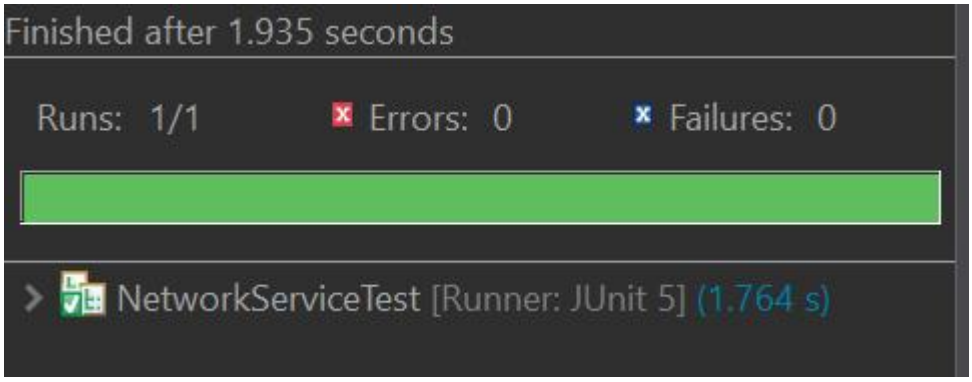
```
package com.example1.testunit1;

public interface NetworkClient {
    String connect();
}
```

```
package com.example1.testunit1;

public class NetworkService {
    private NetworkClient networkClient;

    public NetworkService(NetworkClient networkClient) {
        this.networkClient = networkClient;
    }

    public String connectToServer() {
        String response = networkClient.connect();
        return "Connected to " + response;
    }
}
```

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class NetworkServiceTest {

    @Test
    public void testServiceWithMockNetworkClient() {
        NetworkClient mockNetworkClient = mock(NetworkClient.class);

        when(mockNetworkClient.connect()).thenReturn("Mock Connection");
```

```
        NetworkService networkService = new
NetworkService(mockNetworkClient);
        package com.example1.testunit1;

        public class NetworkService {
            private NetworkClient networkClient;

            public NetworkService(NetworkClient networkClient) {
                this.networkClient = networkClient;
            }

            public String connectToServer() {
                String response = networkClient.connect();
                return "Connected to " + response;
            }
        }


        String result = networkService.connectToServer();
        assertEquals("Connected to Mock Connection", result);
    }
}
```

OUTPUT:



Exercise - 5: Mocking Multiple Return Values

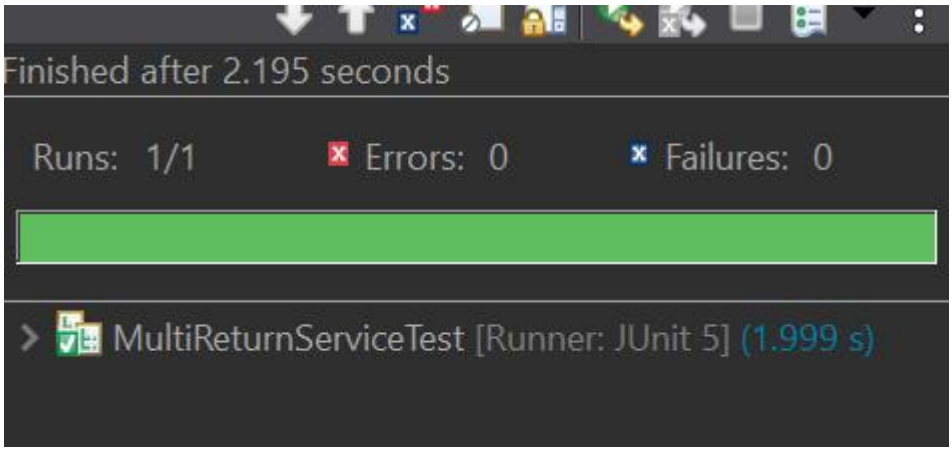
```
package com.example1.testunit1;

public interface Repository {
    String getData();
}
```

```
package com.example1.testunit1;

public class Service {
    private Repository repository;

    public Service(Repository repository) {
        this.repository = repository;
    }

    public String processData() {
```

```
        return "Processed " + repository.getData();
    }
}
```

```
package com.example1.testunit1;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;

public class MultiReturnServiceTest {

    @Test
    public void testServiceWithMultipleReturnValues() {
        Repository mockRepository = mock(Repository.class);

        when(mockRepository.getData())
            .thenReturn("First Mock Data")
            .thenReturn("Second Mock Data");

        Service service = new Service(mockRepository);

        String firstResult = service.processData();
        String secondResult = service.processData();

        assertEquals("Processed First Mock Data", firstResult);
        assertEquals("Processed Second Mock Data", secondResult);
    }
}
```

OUTPUT:



Logging using SLF4J

Exercise - 2: Parameterized Logging

ParameterizedLoggingExample.java

```java
package com.example1.testunit1;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class ParameterizedLoggingExample {

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

    public static void main(String[] args) {
        String username = "John Doe.";
        int loginAttempts = 3;

        logger.info("User {} has logged in successfully.", username);
        logger.warn("User {} has attempted to login {} times
unsuccessfully.", username, loginAttempts);
        logger.error("User {} failed to login after {} attempts. Account
may be locked.", username, loginAttempts);
    }
}
```

OUTPUT:

```
<terminated> ParameterizedLoggingExample [Java Application] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw.exe (30-Jun-2025, 4:35
04:35:36.958 [main] INFO com.example1.testunit1.ParameterizedLoggingExample -- User John Doe. has logged in successfully.
04:35:36.968 [main] WARN com.example1.testunit1.ParameterizedLoggingExample -- User John Doe. has attempted to login 3 times unsuccessfully.
04:35:36.968 [main] ERROR com.example1.testunit1.ParameterizedLoggingExample -- User John Doe. failed to login after 3 attempts. Account may be locked.
```

Exercise - 3: Using Different Appenders

logback.xml

```
<configuration>
  <appender name="console" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
    </encoder>
  </appender>

  <appender name="file" class="ch.qos.logback.core.FileAppender">
    <file>logs/app.log</file>
    <append>true</append>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} %-5level %logger{36} -
%msg%n</pattern>
    </encoder>
  </appender>

  <root level="debug">
    <appender-ref ref="console" />
    <appender-ref ref="file" />
  </root>
</configuration>
```

LoggingExample.java

```
package com.example1.testunit1;

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.info("Logging from SLF4J - Hello, User.");
        logger.warn("This is a warning message...");
        logger.error("This is an error message...");
    }
}
```

OUTPUT:

```
<terminated> LoggingExample [Java Application] C:\Users\admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\javaw
04:47:47.152 [main] INFO  c.example1.testunit1.LoggingExample - Logging from SLF4J - Hello, User.
04:47:47.156 [main] WARN  c.example1.testunit1.LoggingExample - This is a warning message...
04:47:47.157 [main] ERROR c.example1.testunit1.LoggingExample - This is an error message...
```