Exercise 1: Inventory Management System

Code:

```
-- Create Customers table
CREATE TABLE Customers (
    CustomerID NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Age NUMBER,
    InterestRate NUMBER,
    Balance NUMBER,
    IsVIP VARCHAR2(5)
);

-- Create Loans table
CREATE TABLE Loans (
    LoanID NUMBER PRIMARY KEY,
    CustomerID NUMBER,
    DueDate DATE
);

-- Insert sample data
INSERT INTO Customers VALUES (1, 'Kechu', 63, 10.0, 13000, 'FALSE');
INSERT INTO Customers VALUES (2, 'Banu', 48, 11.0, 9000, 'FALSE');
INSERT INTO Customers VALUES (3, 'Suman', 67, 9.5, 11000, 'FALSE');
```

```
INSERT INTO Loans VALUES (101, 1, SYSDATE + 10);
INSERT INTO Loans VALUES (102, 2, SYSDATE + 40);
INSERT INTO Loans VALUES (103, 3, SYSDATE + 5);

COMMIT;

-- Scenario 1 - Apply 1% interest discount for customers age > 60
BEGIN
   FOR rec IN (SELECT CustomerID, Age FROM Customers)
   LOOP
      IF rec.Age > 60 THEN
         UPDATE Customers
         SET InterestRate = InterestRate - 1
         WHERE CustomerID = rec.CustomerID;
      END IF;
   END LOOP;
   COMMIT;
END;
/
-- Scenario 2 - Set IsVIP = TRUE for balance > 10000
BEGIN
   FOR rec IN (SELECT CustomerID, Balance FROM Customers)
   LOOP
      IF rec.Balance > 10000 THEN
```

```
            UPDATE Customers
            SET IsVIP = 'TRUE'
            WHERE CustomerID = rec.CustomerID;
        END IF;
    END LOOP;
    COMMIT;
END;
/

-- Scenario 3 - Show loan reminders for loans due within 30 days
BEGIN
    DBMS_OUTPUT.PUT_LINE('--- Loan Due Reminders ---');
    FOR rec IN (
        SELECT L.LoanID, L.DueDate, C.Name
        FROM Loans L
        JOIN Customers C ON L.CustomerID = C.CustomerID
        WHERE L.DueDate BETWEEN SYSDATE AND SYSDATE + 30
    )
    LOOP
        DBMS_OUTPUT.PUT_LINE(
            'Reminder: ' || rec.Name ||
            ' has loan ' || rec.LoanID ||
            ' due on ' || TO_CHAR(rec.DueDate, 'DD-MON-YYYY')
        );
```

```
    END LOOP;
END;
/
```

```
--- Loan Due Reminders ---
Reminder: Kechu has loan 101 due on 07-JUL-2025
Reminder: Suman has loan 103 due on 02-JUL-2025


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.017
```

Exercise 3: Stored Procedures

Code:

```
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE SavingsAccounts';
  EXECUTE IMMEDIATE 'DROP TABLE Employees';
  EXECUTE IMMEDIATE 'DROP TABLE Accounts';
EXCEPTION
  WHEN OTHERS THEN NULL;
END;
/

CREATE TABLE SavingsAccounts (
    AccountID NUMBER PRIMARY KEY,
```

```
    CustomerName VARCHAR2(50),
    Balance NUMBER
);

CREATE TABLE Employees (
    EmpID NUMBER PRIMARY KEY,
    Name VARCHAR2(50),
    Department VARCHAR2(50),
    Salary NUMBER
);

CREATE TABLE Accounts (
    AccountID NUMBER PRIMARY KEY,
    CustomerName VARCHAR2(50),
    Balance NUMBER
);

INSERT INTO SavingsAccounts VALUES (1, 'ian', 9000);
INSERT INTO SavingsAccounts VALUES (2, 'sandeep', 12000);
INSERT INTO SavingsAccounts VALUES (3, 'zayn', 18000);

INSERT INTO Employees VALUES (201, 'sri', 'Finance', 45000);
INSERT INTO Employees VALUES (202, 'keerthana', 'IT', 52000);
INSERT INTO Employees VALUES (203, 'shruthi', 'IT', 48000);
```

```
INSERT INTO Accounts VALUES (301, 'cedric', 7000);
INSERT INTO Accounts VALUES (302, 'ron weasely', 10500);

COMMIT;

-- Scenario 1: Process Monthly Interest
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
    FOR rec IN (SELECT AccountID, Balance FROM SavingsAccounts)
    LOOP
        UPDATE SavingsAccounts
        SET Balance = Balance + (Balance * 0.01)
        WHERE AccountID = rec.AccountID;
    END LOOP;
    COMMIT;
END;
/

-- Scenario 2: Update Employee Bonus
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
    p_dept IN VARCHAR2,
    p_bonus_pct IN NUMBER
) IS
```

```
BEGIN
    UPDATE Employees
    SET Salary = Salary + (Salary * p_bonus_pct / 100)
    WHERE Department = p_dept;
    COMMIT;
END;
/

-- Scenario 3: Transfer Funds Between Accounts
CREATE OR REPLACE PROCEDURE TransferFunds (
    p_from_account IN NUMBER,
    p_to_account IN NUMBER,
    p_amount IN NUMBER
) IS
    v_balance NUMBER;
BEGIN
    SELECT Balance INTO v_balance
    FROM Accounts
    WHERE AccountID = p_from_account;

    IF v_balance < p_amount THEN
        RAISE_APPLICATION_ERROR(-20001, 'Insufficient balance in source account');
    END IF;
```

```
    UPDATE Accounts
    SET Balance = Balance - p_amount
    WHERE AccountID = p_from_account;

    UPDATE Accounts
    SET Balance = Balance + p_amount
    WHERE AccountID = p_to_account;

    COMMIT;
END;
/

-- Scenario 1 Test
EXEC ProcessMonthlyInterest;

-- Scenario 2 Test
EXEC UpdateEmployeeBonus('IT', 10);

-- Scenario 3 Test
EXEC TransferFunds(301, 302, 2000);

SELECT * FROM SavingsAccounts;
SELECT * FROM Employees;
SELECT * FROM Accounts;
```

Output:
- After EXEC ProcessMonthlyInterest;
-- SavingsAccounts:
ACCOUNTID CUSTOMERNAME BALANCE
1 ian 9090
2 sandeep 12120
3 zayn 18180

-- After EXEC UpdateEmployeeBonus('IT', 10);
-- Employees:
EMPID NAME DEPARTMENT SALARY
201 sri Finance 45000
202 keerthana IT 57200
203 shruthi IT 52800

-- After EXEC TransferFunds(301, 302, 2000);
-- Accounts:
ACCOUNTID CUSTOMERNAME BALANCE
301 cedric 5000
302 ron weasely 12500

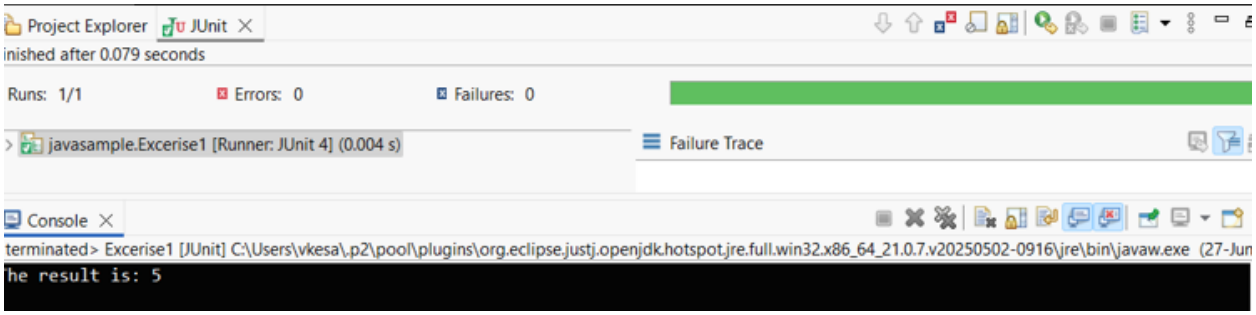
TDD using Junit5 and Mockito

Exercise 1: Setting Up Junit

```
package javasample;

import org.junit.Test;

import static org.junit.Assert.*;

public class Excerise1{

    @Test

   public void testAddition() {

    int result = 2 + 3;

    System.out.println("The result is: " + result);  // This will show in the console

    assertEquals(5, result);

    }
```

}

OUTPUT

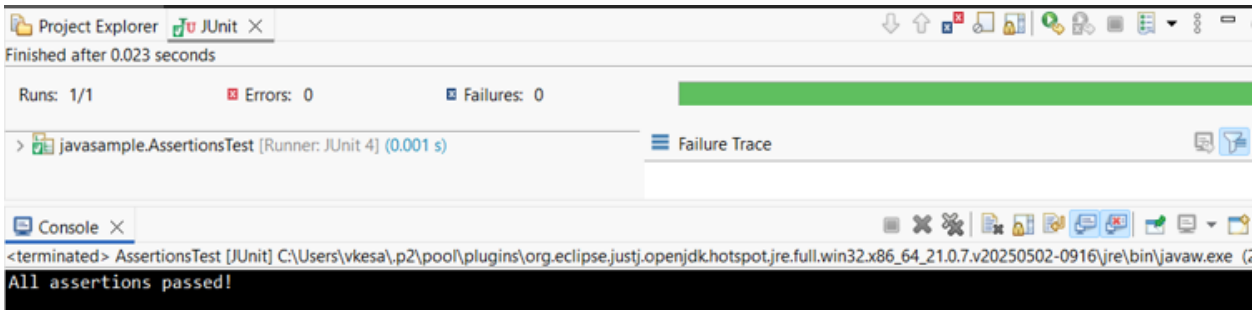


Exercise 3: Assertions in Junit

```
package javasample;

import org.junit.Test;

import static org.junit.Assert.*;

public class AssertionsTest {

	@Test

	public void testAssertions() {

	// Assert equals

	assertEquals(5, 2 + 3);

	// Assert true

	assertTrue(5 > 3);

	// Assert false

	assertFalse(5 < 3);

	// Assert null

	assertNull(null);
```

```
        // Assert not null
        assertNotNull(new Object());
        // Console output (optional)
        System.out.println("All assertions passed!");
        }
}
```

OUTPUT



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

Calculator.java

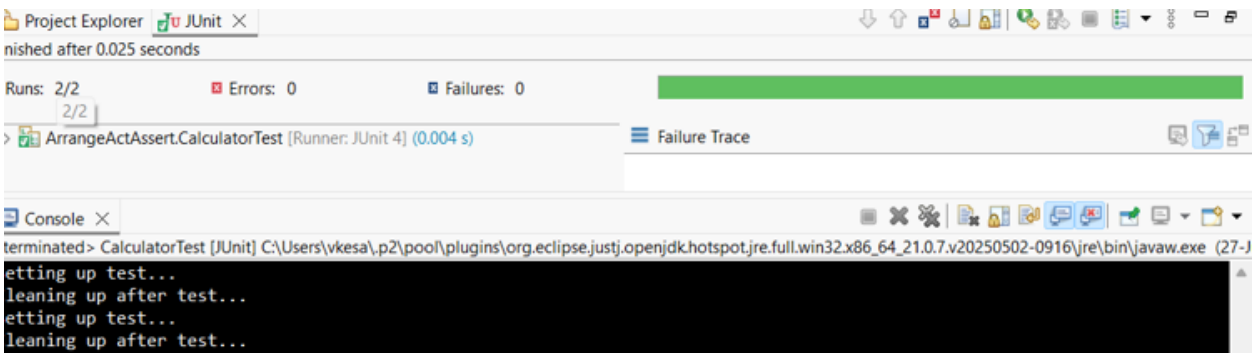
```
package ArrangeActAssert;
public class Calculator {
        public int add(int a, int b) {
        return a + b;
        }
        public int subtract(int a, int b) {
        return a - b;
        }
}
```

CalculatorTest.java

```
package ArrangeActAssert;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.*;

public class CalculatorTest {

	private Calculator calculator;

	// Setup method: runs before each test

	@Before

	public void setUp() {

	System.out.println("Setting up test...");

	calculator = new Calculator();

	}


	// Teardown method: runs after each test

	@After

	public void tearDown() {

	System.out.println("Cleaning up after test...");

	calculator = null;

	}


	@Test
```

```
    public void testAddition() {
    // Arrange
    int a = 5;
    int b = 3;
    // Act
    int result = calculator.add(a, b);
    // Assert
    assertEquals(8, result);
    }
    @Test
    public void testSubtraction() {
    // Arrange
    int a = 10;
    int b = 4;
    // Act
    int result = calculator.subtract(a, b);
    // Assert
    assertEquals(6, result);
    }
}
```

OUTPUT



Exercise 1: Mocking and Stubbing

ExternalApi.java

```
package com.example.demo;

public interface ExternalApi {

	String getData();

}
```

MyService.java

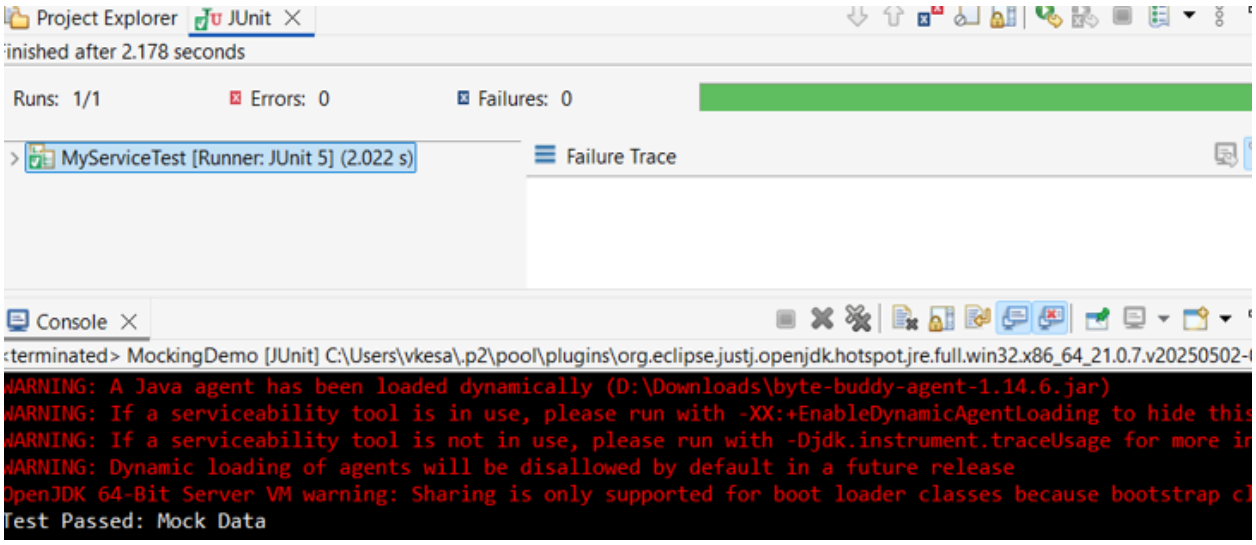
```
package com.example.demo

public class MyService {

	private ExternalApi externalApi;

	public MyService(ExternalApi externalApi) {

	this.externalApi = externalApi;

	}

	public String fetchData() {

	return externalApi.getData();

	}
```

```
}
```

MyServiceTest.java

```
package com.example.demo;

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

import static org.mockito.Mockito.*

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

	@Test

	public void testExternalApi() {

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

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


	MyService service = new MyService(mockApi);

	String result = service.fetchData();


	assertEquals("Mock Data", result);

	System.out.println("Test Passed: " + result);

	}

}
```

OUTPUT



Exercise 2: Verifying Interactions

ExternalApi.java

```
package VERIFYINTERACTION;

public interface ExternalApi {

	String getData();

}
```

MyService.java

```
package VERIFYINTERACTION;

public class MyService {

	private ExternalApi externalApi;


	public MyService(ExternalApi externalApi) {

	this.externalApi = externalApi;

	}
```

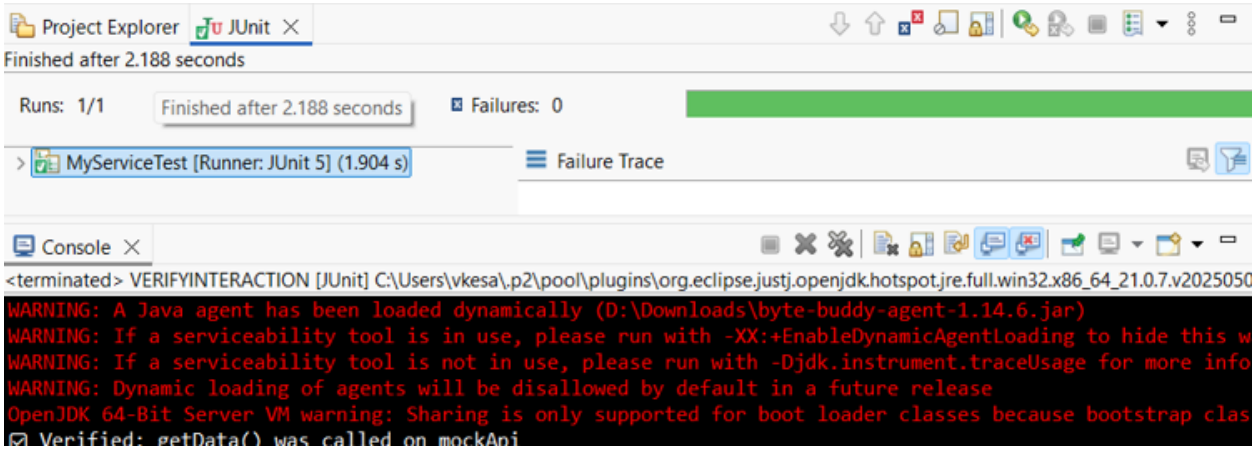
```
    public String fetchData() {
    return externalApi.getData();
    }
}
```

MyServiceTest.java

```
package VERIFYINTERACTION;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
public class MyServiceTest {
    @Test
    public void testVerifyInteraction() {
    // Step 1: Create a mock object
    ExternalApi mockApi = Mockito.mock(ExternalApi.class);

    // Step 2: Call the method using MyService
    MyService service = new MyService(mockApi);
    service.fetchData();

    // Step 3: Verify that getData() was called
    verify(mockApi).getData();
```

```
        System.out.println("Verified: getData() was called on mockApi");
        }
}
```

OUTPUT



Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

LoggingExample

```
package Logging;

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