WEEK 2

Kavya V.R

St. Joseph’s Institute of Technology

Superset ID: 6377320

JUnit Testing Exercises

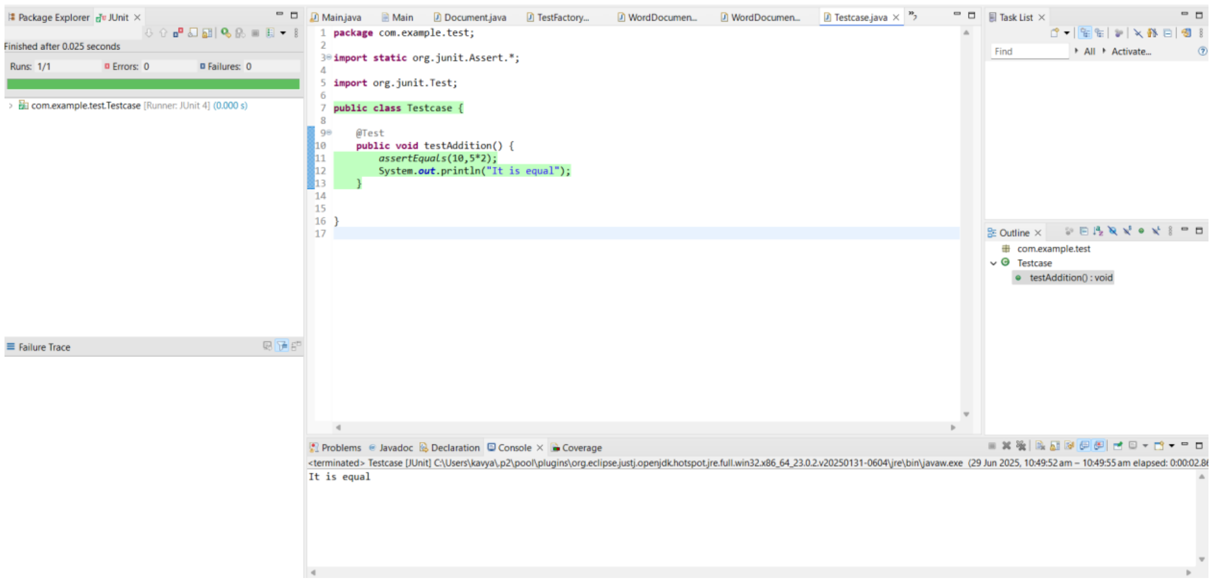
Exercise 1:

Setting Up JUnit Scenario: You need to set up JUnit in your Java project to start writing unit tests.

Testcase.java:

```
package com.example.test;

import static org.junit.Assert.*;

import org.junit.Test;

public class Testcase {

        @Test

        public void testAddition() {

            assertEquals(10,5*2);

        }

}
```

Output:



```
Finished after 0.025 seconds
Runs: 1/1    Errors: 0    Failures: 0
com.example.test.Testcase [Runner: JUnit 4] (0.000 s)
```

```
package com.example.test;

import static org.junit.Assert.*;

import org.junit.Test;

public class Testcase {

    @Test
    public void testAddition() {
        assertEquals(10,5*2);
        System.out.println("It is equal");
    }


}
```

```
<terminated> Testcase [JUnit] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 10:49:52 am – 10:49:55 am elapsed: 0:00:02.86
It is equal
```

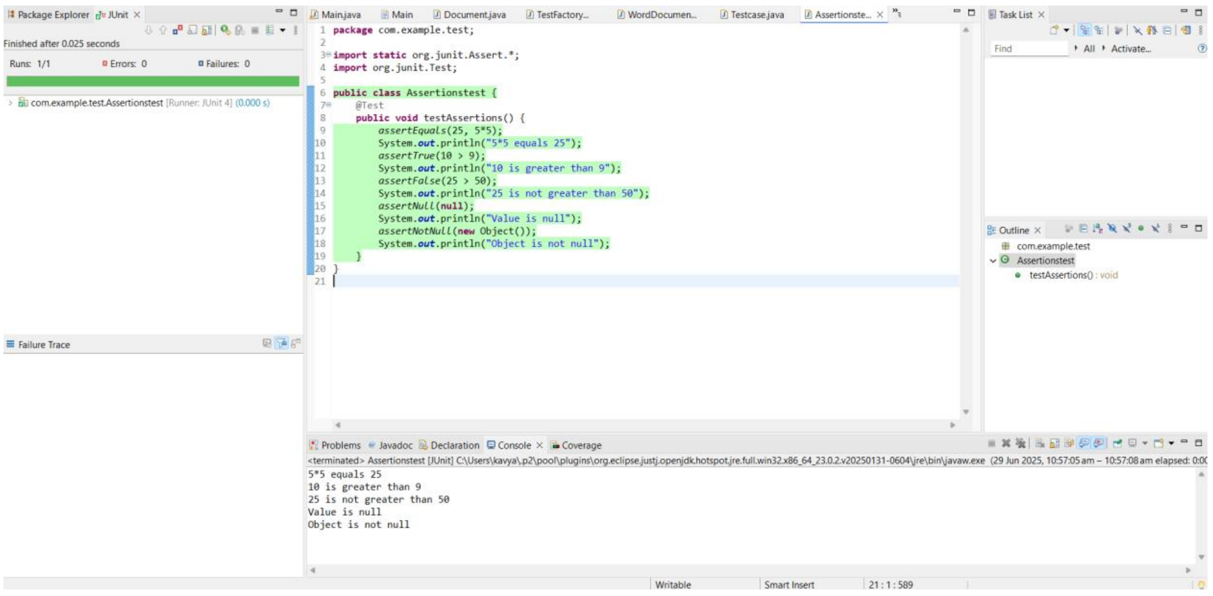
Exercise 3: Assertions in JUnit Scenario:

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

Assertionstest.java

```
package com.example.test;
import static org.junit.Assert.*;
import org.junit.Test;
public class Assertionstest {
    @Test
    public void testAssertions() {
        assertEquals(25, 5*5);
        System.out.println("5*5 equals 25");
        assertTrue(10 > 9);
        System.out.println("10 is greater than 9");
        assertFalse(25 > 50);
        System.out.println("25 is not greater than 50");
        assertNull(null);
        System.out.println("Value is null");
        assertNotNull(new Object());
        System.out.println("Object is not null");
    }
}
```

Output:



Finished after 0.025 seconds

Runs: 1/1   Errors: 0   Failures: 0

com.example.test.Assertionstest [Runner: JUnit 4] (0.000 s)

```java
package com.example.test;

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

public class Assertionstest {
    @Test
    public void testAssertions() {
        assertEquals(25, 5*5);
        System.out.println("5*5 equals 25");
        assertTrue(10 > 9);
        System.out.println("10 is greater than 9");
        assertFalse(25 > 50);
        System.out.println("25 is not greater than 50");
        assertNull(null);
        System.out.println("Value is null");
        assertNotNull(new Object());
        System.out.println("Object is not null");
    }
}
```

```
<terminated> Assertionstest [JUnit] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 10:57:05 am – 10:57:08 am elapsed: 0:0
5*5 equals 25
10 is greater than 9
25 is not greater than 50
Value is null
Object is not null
```

Exercise 4:

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

Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

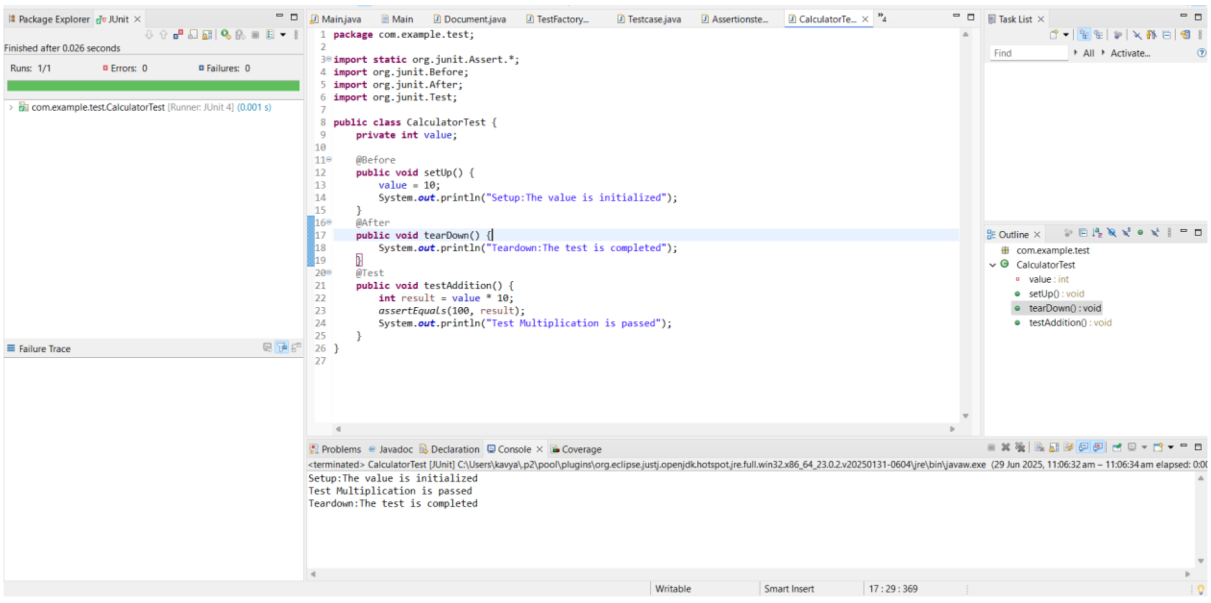
Steps:

1. Write tests using the AAA pattern.

2. Use @Before and @After annotations for setup and teardown methods

Code:

```java
package com.example.test;

import static org.junit.Assert.*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class CalculatorTest {

    private int value;

    @Before
    public void setUp() {
        value = 10;
        System.out.println("Setup:The value is initialized");
    }
    @After
    public void tearDown() {
        System.out.println("Teardown:The test is completed");
    }
    @Test
    public void testAddition() {
        int result = value * 10;
        assertEquals(100, result);
        System.out.println("Test Multiplication is passed");
    }
}
```



Finished after 0.026 seconds

Runs: 1/1   Errors: 0   Failures: 0

com.example.test.CalculatorTest [Runner: JUnit 4] (0.001 s)

```
package com.example.test;

import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;

public class CalculatorTest {
    private int value;

    @Before
    public void setUp() {
        value = 10;
        System.out.println("Setup:The value is initialized");
    }
    @After
    public void tearDown() {
        System.out.println("Teardown:The test is completed");
    }
    @Test
    public void testAddition() {
        int result = value * 10;
        assertEquals(100, result);
        System.out.println("Test Multiplication is passed");
    }
}
```

```
<terminated> CalculatorTest [JUnit] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 11:06:32 am – 11:06:34 am elapsed: 0:0
Setup:The value is initialized
Test Multiplication is passed
Teardown:The test is completed
```

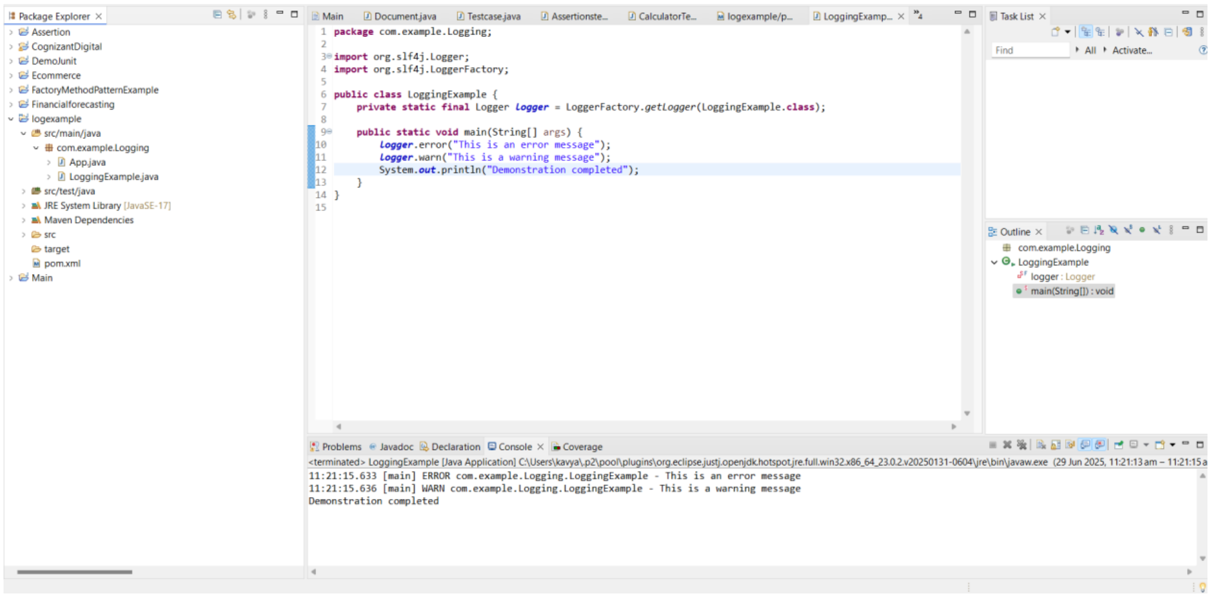
Logging using SLF4J

Exercise 1:

Logging Error Messages and Warning Levels Task:

Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

```
package com.example.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");
      System.out.println("Demonstration completed");
   }
}
```



```
package com.example.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");
        System.out.println("Demonstration completed");
    }
}
```

```
<terminated> LoggingExample [Java Application] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 11:21:13 am – 11:21:15 a
11:21:15.633 [main] ERROR com.example.Logging.LoggingExample - This is an error message
11:21:15.636 [main] WARN com.example.Logging.LoggingExample - This is a warning message
Demonstration completed
```

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.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

Code:

ExternalApi.java

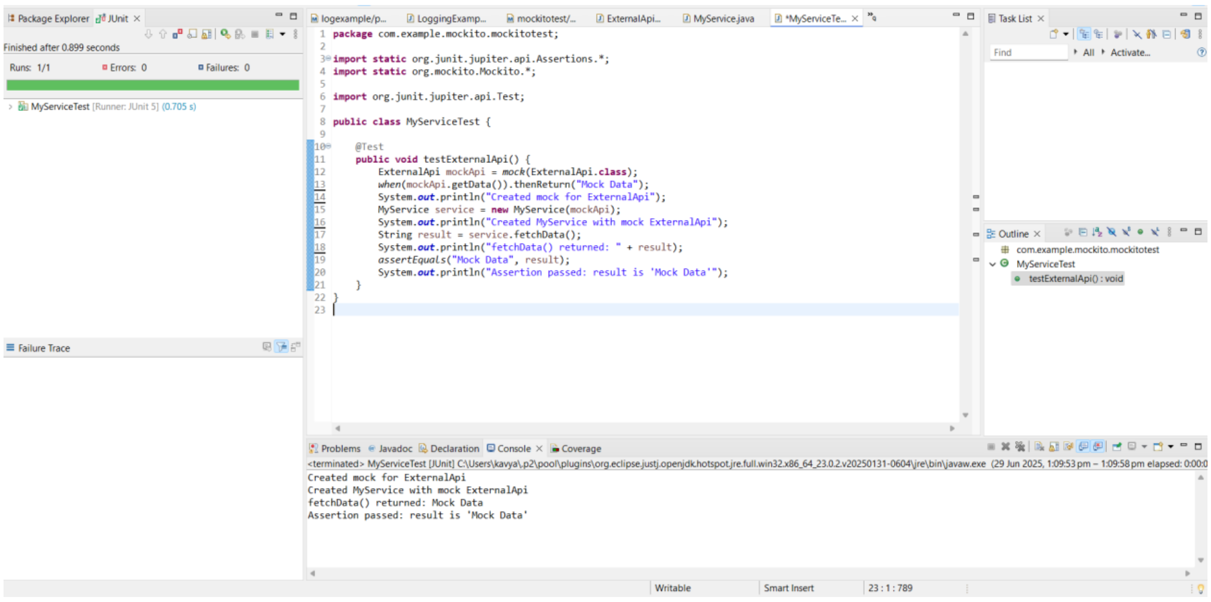
```
package com.example.mockito;

public interface ExternalApi {

            String getData();

}
```

MyService.java

```
package com.example.mockito;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

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

MyServiceTest.java

```
package com.example.mockito.mockitotest;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class MyServiceTest {
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        System.out.println("Created mock for ExternalApi");
        MyService service = new MyService(mockApi);
        System.out.println("Created MyService with mock ExternalApi");
        String result = service.fetchData();
        System.out.println("fetchData() returned: " + result);
        assertEquals("Mock Data", result);
        System.out.println("Assertion passed: result is 'Mock Data'");
    }
}
```



```java
package com.example.mockito.mockitotest;

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

import org.junit.jupiter.api.Test;

public class MyServiceTest {

    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        System.out.println("Created mock for ExternalApi");
        MyService service = new MyService(mockApi);
        System.out.println("Created MyService with mock ExternalApi");
        String result = service.fetchData();
        System.out.println("fetchData() returned: " + result);
        assertEquals("Mock Data", result);
        System.out.println("Assertion passed: result is 'Mock Data'");
    }
}
```

```
<terminated> MyServiceTest [JUnit] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 1:09:53 pm – 1:09:58 pm elapsed: 0:00:0
Created mock for ExternalApi
Created MyService with mock ExternalApi
fetchData() returned: Mock Data
Assertion passed: result is 'Mock Data'
```

Exercise 2: Verifying Interactions Scenario:

You need to ensure that a method is called with specific arguments.

Steps:

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

Code:

ExternalApi.java

```java
package com.example.mockito;

public interface ExternalApi {

        String getData();

}
```

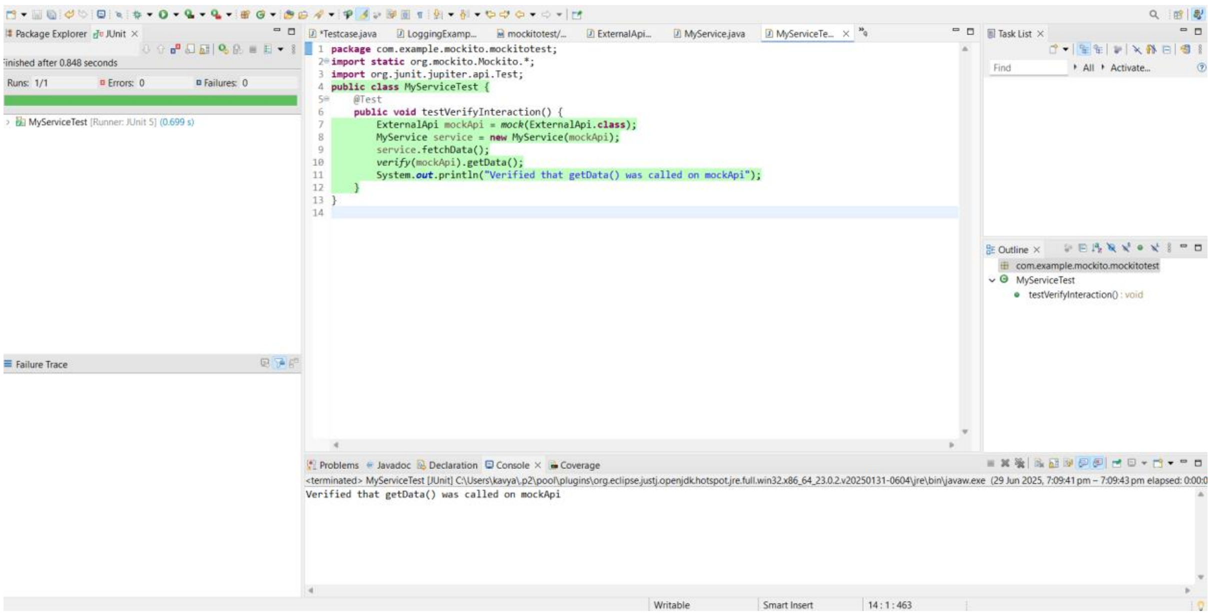
MyService.java

```java
package com.example.mockito;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

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

MyServiceTest.java

```
package com.example.mockito.mockitotest;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class MyServiceTest {
    @Test
    public void testVerifyInteraction() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService service = new MyService(mockApi);
        service.fetchData();
        verify(mockApi).getData();
        System.out.println("Verified that getData() was called on mockApi");
    }
}
```

Output:



```java
package com.example.mockito.mockitotest;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class MyServiceTest {
    @Test
    public void testVerifyInteraction() {
        ExternalApi mockApi = mock(ExternalApi.class);
        MyService service = new MyService(mockApi);
        service.fetchData();
        verify(mockApi).getData();
        System.out.println("Verified that getData() was called on mockApi");
    }
}
```

```
<terminated> MyServiceTest [JUnit] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (29 Jun 2025, 7:09:41 pm – 7:09:43 pm elapsed: 0:00:0
Verified that getData() was called on mockApi
```

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.

Code:

```
DECLARE
        v_age NUMBER;
BEGIN
        FOR rec IN (SELECT CustomerID, InterestRate FROM Loans l)
        JOIN Customers c ON l.CustomerID = c.CustomerID
        LOOP
            SELECT FLOOR(MONTHS_BETWEEN(SYSDATE, c.DOB) / 12) INTO v_age
      FROM Customers c WHERE c.CustomerID = rec.CustomerID;
        IF v_age > 60 THEN
            UPDATE Loans
            SET InterestRate = InterestRate * 0.99
            WHERE CustomerID = rec.CustomerID;
        END IF;
        END LOOP;
        COMMIT;
 END;
```

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.

Code:

ALTER TABLE first: ALTER TABLE Customers ADD (IsVIP VARCHAR2(3));

```
UPDATE Customers SET IsVIP = 'FALSE';
BEGIN
        FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP
        IF rec.Balance > 10000 THEN
        UPDATE Customers SET IsVIP = 'TRUE' WHERE CustomerID = rec.CustomerID;
        ELSE
            UPDATE Customers SET IsVIP = 'FALSE' WHERE CustomerID = rec.CustomerID;
        END IF;
        END LOOP;
        COMMIT;
END;
```

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.

Code:

```
BEGIN
        FOR rec IN (
            SELECT c.Name, l.EndDate
            FROM Loans l
            JOIN Customers c ON l.CustomerID = c.CustomerID
        WHERE l.EndDate BETWEEN SYSDATE AND ADD_MONTHS(SYSDATE, 1)
        ) LOOP
                DBMS_OUTPUT.PUT_LINE('Reminder: ' || rec.Name || ', your loan is due
      on ' || TO_CHAR(rec.EndDate, 'YYYY-MM-DD'));
        END LOOP;
END;
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS
BEGIN
        UPDATE Accounts
        SET Balance = Balance * 1.01
        WHERE AccountType = 'Savings';
        COMMIT;
```

```
END;
EXECUTE ProcessMonthlyInterest;
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (
        p_Department VARCHAR2,
        p_BonusPercentage NUMBER
) IS
BEGIN
        UPDATE Employees
        SET Salary = Salary + (Salary * p_BonusPercentage / 100)
        WHERE Department = p_Department;
        COMMIT;
END;
EXECUTE UpdateEmployeeBonus('IT', 10);
```

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.

Code:

```
CREATE OR REPLACE PROCEDURE TransferFunds (
p_SourceAccountID NUMBER,
p_DestAccountID NUMBER,
p_Amount NUMBER
) IS
v_SourceBalance NUMBER;
```

```
BEGIN
                SELECT Balance INTO v_SourceBalance FROM Accounts WHERE AccountID =
p_SourceAccountID;
IF v_SourceBalance >= p_Amount THEN
                    UPDATE Accounts
                    SET Balance = Balance - p_Amount
                    WHERE AccountID = p_SourceAccountID;

                    UPDATE Accounts
                    SET Balance = Balance + p_Amount
                    WHERE AccountID = p_DestAccountID;
                    COMMIT;
         ELSE
                    RAISE_APPLICATION_ERROR(-20001, 'Insufficient funds in source     account.');
END IF;
END;
EXECUTE TransferFunds(1, 2, 500);
```