WEEK 2

Naren Karthick A

St. Joseph’s Institute of Technology

Superset ID: 6377326

JUnit Testing

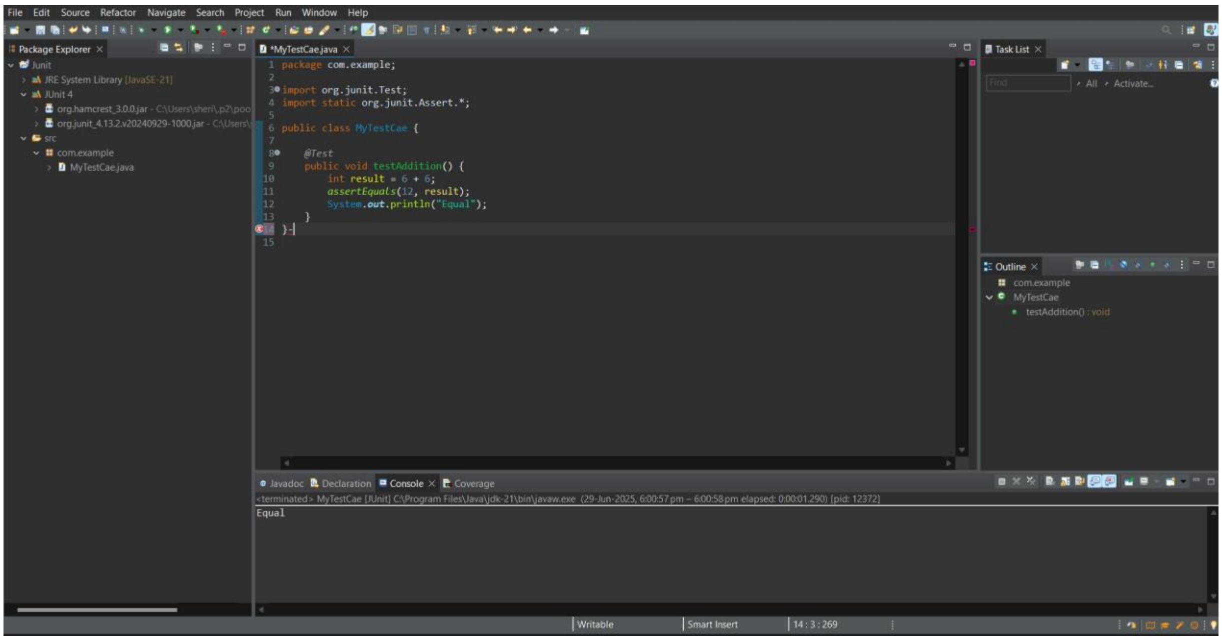
Exercise 1: Setting Up JUnit

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

Code:

MyTestCase.java:

```java
package com.example;
import org.junit.Test;
import static org.junit.Assert.*;
public class MyTestCae {
    @Test
    public void testAddition() {
        int result = 6 + 6;
        assertEquals(12, result);
        System.out.println("Equal");
    }
}
```



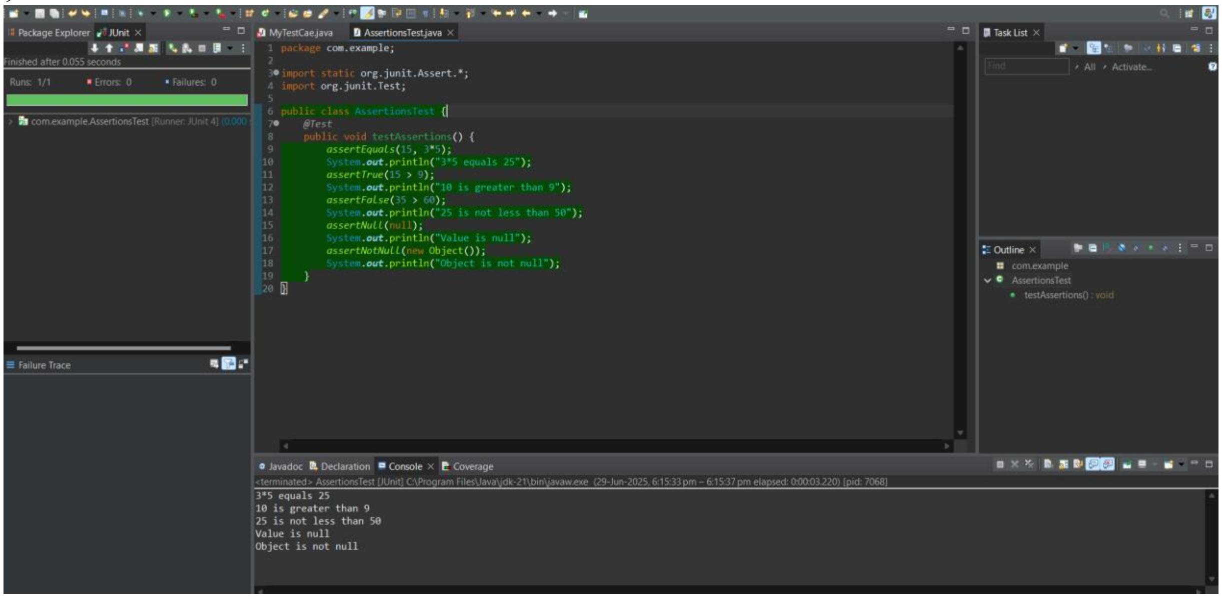
```java
package com.example;

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

public class MyTestCae {

    @Test
    public void testAddition() {
        int result = 6 + 6;
        assertEquals(12, result);
        System.out.println("Equal");
    }
}
```

```
<terminated> MyTestCae [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe  (29-Jun-2025, 6:00:57 pm – 6:00:58 pm elapsed: 0:00:01.290) [pid: 12372]
Equal
```

Exercise 3: Assertions in Junit
Scenario: You need to use different assertions in JUnit to validate your test results

AssertionsTest.java:

```java
package com.example;
import static org.junit.Assert.*;
import org.junit.Test;
public class AssertionsTest {
   @Test
   public void testAssertions() {
      assertEquals(15, 3*5);
      System.out.println("3*5 equals 25");
      assertTrue(15 > 9);
      System.out.println("10 is greater than 9");
      assertFalse(35 > 60);
      System.out.println("25 is not less than 50");
      assertNull(null);
      System.out.println("Value is null");
      assertNotNull(new Object());
      System.out.println("Object is not null");
   }
}
```



Finished after 0.055 seconds

Runs: 1/1   Errors: 0   Failures: 0

com.example.AssertionsTest [Runner: JUnit 4] (0.000

```java
package com.example;

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

public class AssertionsTest {
    @Test
    public void testAssertions() {
        assertEquals(15, 3*5);
        System.out.println("3*5 equals 25");
        assertTrue(15 > 9);
        System.out.println("10 is greater than 9");
        assertFalse(35 > 60);
        System.out.println("25 is not less than 50");
        assertNull(null);
        System.out.println("Value is null");
        assertNotNull(new Object());
        System.out.println("Object is not null");
    }
}
```

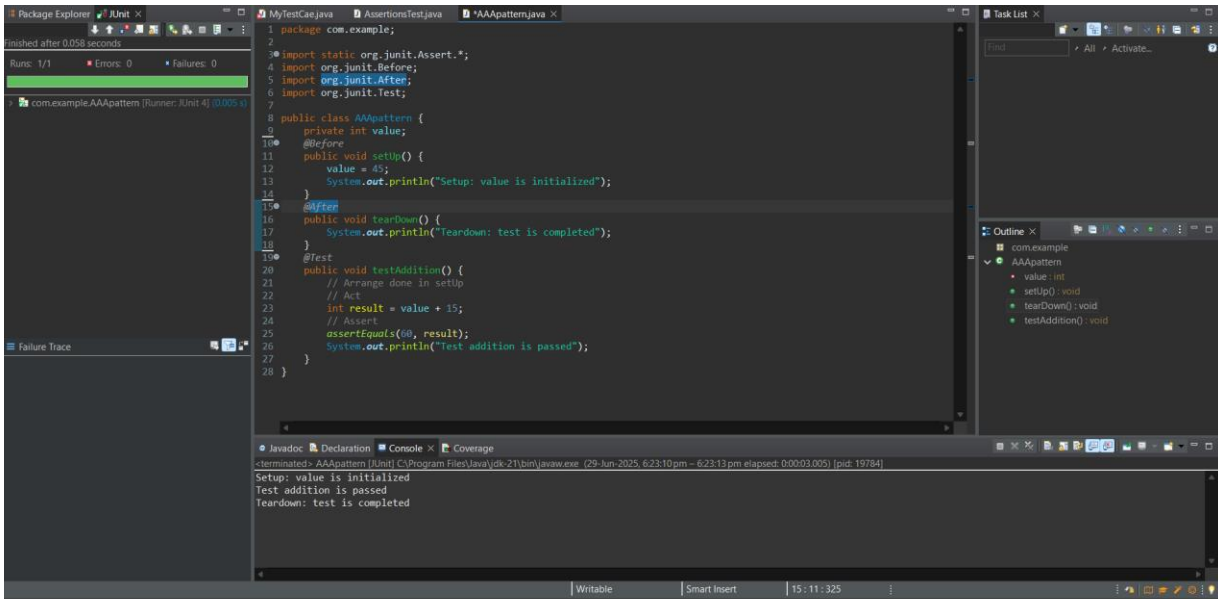
```
<terminated> AssertionsTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe  (29-Jun-2025, 6:15:33 pm – 6:15:37 pm elapsed: 0:00:03.220) [pid: 7068]
3*5 equals 25
10 is greater than 9
25 is not less than 50
Value is null
Object is not null
```

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.

AAApattern.java:

```
package com.example;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
public class AAApattern {
    private int value;
    @Before
    public void setUp() {
        value = 45;
        System.out.println("Setup: value is initialized");
    }
    @After
    public void tearDown() {
        System.out.println("Teardown: test is completed");
    }
    @Test
    public void testAddition() {
        int result = value + 15;
        assertEquals(60, result);
        System.out.println("Test addition is passed");
    }
}
```

Output:



```java
package com.example;

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

public class AAApattern {
    private int value;
    @Before
    public void setUp() {
        value = 45;
        System.out.println("Setup: value is initialized");
    }
    @After
    public void tearDown() {
        System.out.println("Teardown: test is completed");
    }
    @Test
    public void testAddition() {
        // Arrange done in setUp
        // Act
        int result = value + 15;
        // Assert
        assertEquals(60, result);
        System.out.println("Test addition is passed");
    }
}
```

```
<terminated> AAApattern [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe  (29-Jun-2025, 6:23:10 pm – 6:23:13 pm elapsed: 0:00:03.005) [pid: 19784]
Setup: value is initialized
Test addition is passed
Teardown: 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.LogginExample;

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) {

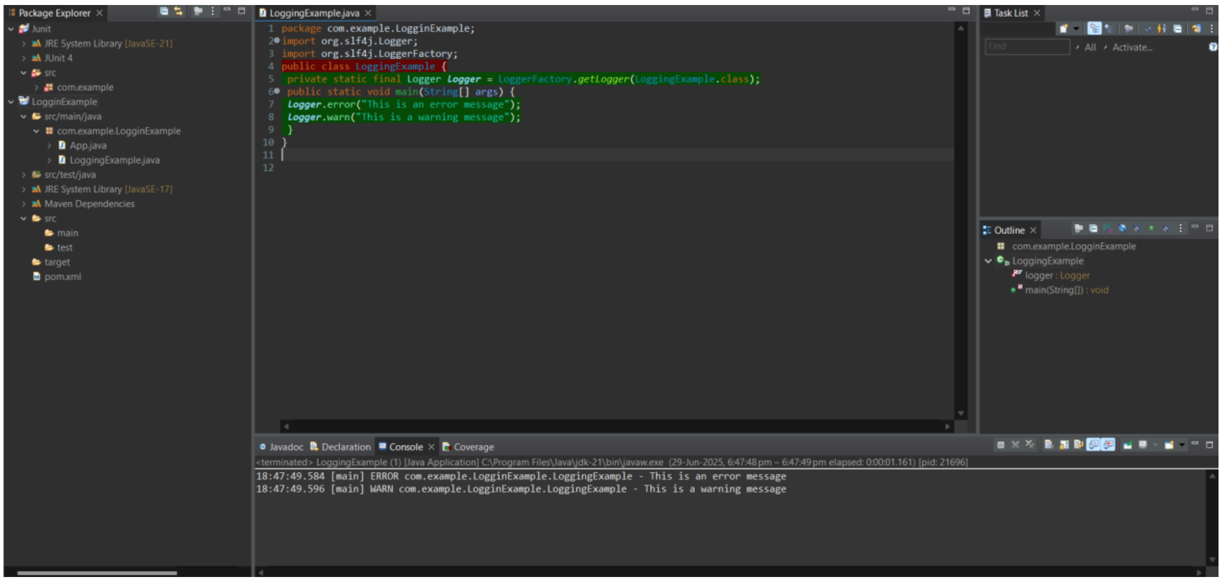
logger.error("This is an error message");

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

}

}

Output:



```
package com.example.LogginExample;
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");
 }
}
```

```
<terminated> LoggingExample (1) [Java Application] C:\Program Files\Java\jdk-21\bin\javaw.exe  (29-Jun-2025, 6:47:48 pm – 6:47:49 pm elapsed: 0:00:01.161) [pid: 21696]
18:47:49.584 [main] ERROR com.example.LogginExample.LoggingExample - This is an error message
18:47:49.596 [main] WARN com.example.LogginExample.LoggingExample - This is a warning message
```

Mockito

Exercise 1: Mocking and Stubbing

Code:

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.

ExternalApi.java

```
package com.example.mockito;

public interface ExternalApi {

String getData();

}
```

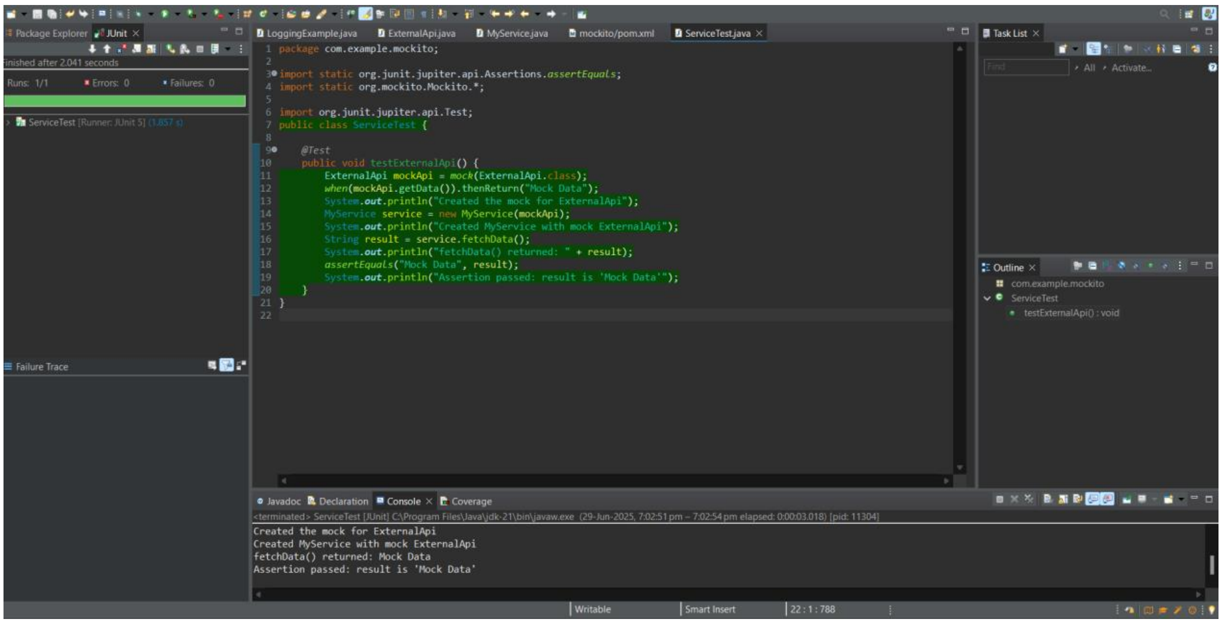
MyService.java

```
package com.example.mockito;

public class MyService {

    private ExternalApi api;

    public MyService(ExternalApi api) {

        this.api = api;

    }

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

ServiceTest.java

```
package com.example.mockito;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class ServiceTest {
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        System.out.println("Created the 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'");
    }
}
```

Output:



```java
package com.example.mockito;

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

import org.junit.jupiter.api.Test;
public class ServiceTest {

    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        System.out.println("Created the 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> ServiceTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe (29-Jun-2025, 7:02:51 pm – 7:02:54 pm elapsed: 0:00:03.018) [pid: 11304]
Created the 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

```
package com.example.mockito;
public interface ExternalApi {
String getData();
}
```

MyService.java

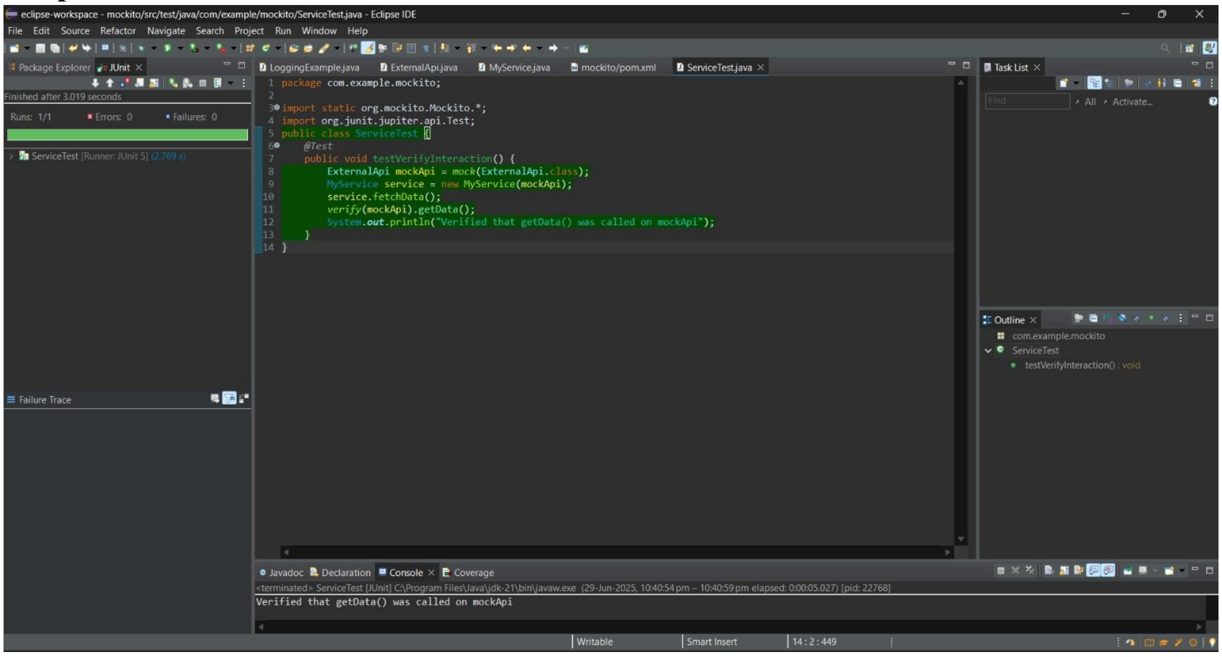
```
package com.example.mockito;
public class MyService {
    private ExternalApi api;
    public MyService(ExternalApi api) {
        this.api = api;
    }
    public String fetchData() {
        return api.getData();
    }
}
```

MyServiceTest.java

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

```
import static org.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:



```
eclipse-workspace - mockito/src/test/java/com/example/mockito/ServiceTest.java - Eclipse IDE
```

```
Finished after 3.019 seconds
Runs: 1/1    Errors: 0    Failures: 0
ServiceTest [Runner: JUnit 5] (2.769 s)
```

```
package com.example.mockito;

import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
public class ServiceTest {
    @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> ServiceTest [JUnit] C:\Program Files\Java\jdk-21\bin\javaw.exe  (29-Jun-2025, 10:40:54 pm – 10:40:59 pm elapsed: 0:00:05.027) [pid: 22768]
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);
```