**SUPER SET ID : 6389675**

**MODULE 2**

**PL/SQL :**

***NOTE: Sir/Mam, I have used online PL/SQL platfrom sql.live by Oracale.***

**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.**

**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.**

**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.**

**--------------------------------------------------------------------------------------------------------------------------------------**

**SOLUTION:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**PL/SQL**

**DATABASE NAME : BANK**

**TABLES : (I)CUSTOMERS (II) CUSTOMERS LOAN**

**CUSTOMERS:**

**-----CREATION OF TABLE:----**

**CREATE TABLE customers (**

**customer\_id     NUMBER PRIMARY KEY,**

**name            VARCHAR2(50),**

**age             NUMBER,**

**balance         NUMBER(10, 2),**

**IsVIP           VARCHAR2(5) DEFAULT 'FALSE' -- TRUE or FALSE**

**);**

**-----VALUE INSERTION:------**

**INSERT INTO customers VALUES (1,'RAM RAJ',33,30000,'FALSE');**

**INSERT INTO customers VALUES (2,'RANI RAJ',30,10000,'FALSE');**

**INSERT INTO customers VALUES (3,'LAXMAN RAJ',65,60000,'FALSE');**

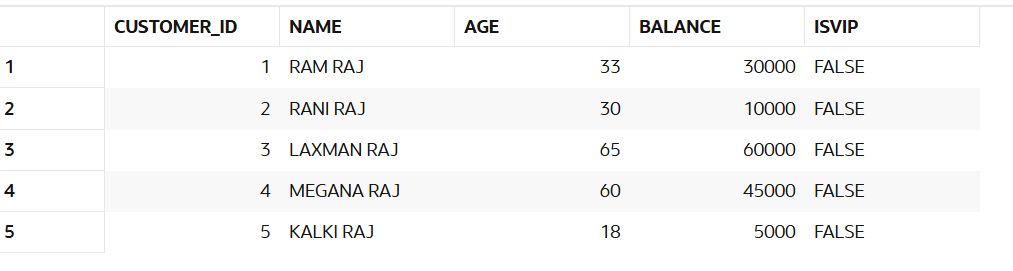
**INSERT INTO customers VALUES (4,'MEGANA RAJ',60,45000,'FALSE');**

**INSERT INTO customers VALUES (5,'KALKI RAJ',18,5000,'FALSE');**

**COMMIT;**

**SELECT \* FROM CUSTOMERS;**

**OUTPUT:**

****

**//LOAN TABLE**

**-----CREATION OF TABLE:----**

**CREATE TABLE loans (**

**loan\_id NUMBER PRIMARY KEY,**

**customer\_id  NUMBER,**

**interest\_rate NUMBER(5,2),**

**due\_date DATE,**

**FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)**

**);**

**-----VALUE INSERTION:------**

**INSERT INTO loans VALUES (101, 1, 10.5, SYSDATE + 20);**

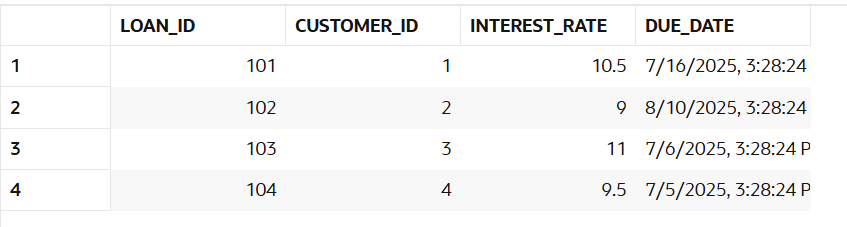
**INSERT INTO loans VALUES (102, 2, 9.0, SYSDATE + 45);**

**INSERT INTO loans VALUES (103, 3, 11.0, SYSDATE + 10);**

**INSERT INTO loans VALUES(104,4,9.5,SYSDATE + 9);**

**COMMIT;**

**OUTPUT:**

****

**PL/SQL QUERY :**

**Scenario 1:** ***Apply 1% discount on interest for customers above 60***

**BEGIN**

**FOR cust IN (SELECT customer\_id, age FROM customers) LOOP**

**IF cust.age > 60 THEN**

**UPDATE loans**

**SET interest\_rate = interest\_rate - 1**

**WHERE customer\_id = cust.customer\_id;**

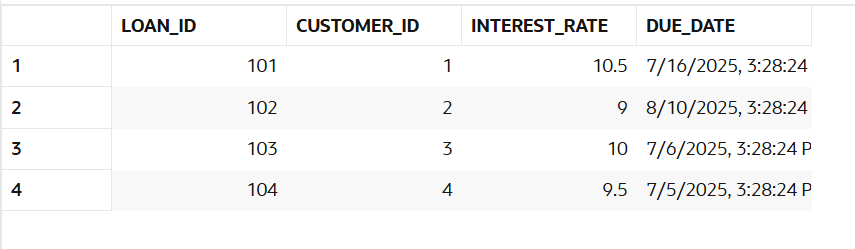
**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**OUTPUT:**

****

**--- HERE FOR CUSTOMER NAMED LAXMAN RAJ INTESER\_RATE HAS GIVEN 1% DISCOUNT.------**

**--------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 2:** ***Mark customers as VIP if balance > 10,000***

**BEGIN**

**FOR cust IN (SELECT customer\_id, balance FROM customers) LOOP**

**IF cust.balance > 10000 THEN**

**UPDATE customers**

**SET IsVIP = 'TRUE'**

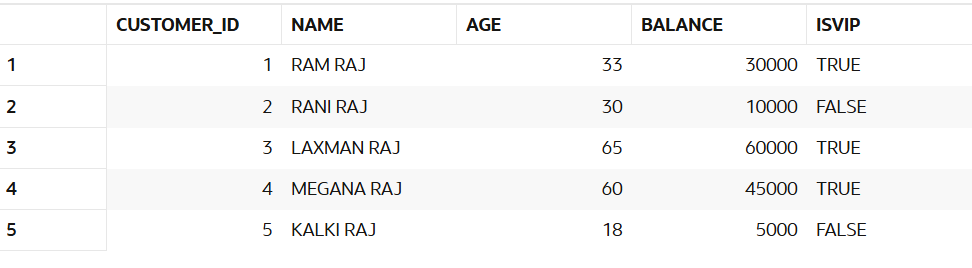
**WHERE customer\_id = cust.customer\_id;**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**//OUTPUT:**

**Scenario 3*:******Remind customers whose loan is due in next 30 days***

**DECLARE**

**v\_due\_date DATE := SYSDATE + 30;**

**BEGIN**

**FOR loan IN (**

**SELECT loan\_id, customer\_id, due\_date**

**FROM loans**

**WHERE due\_date BETWEEN SYSDATE AND v\_due\_date**

**) LOOP**

**DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || loan.loan\_id ||**

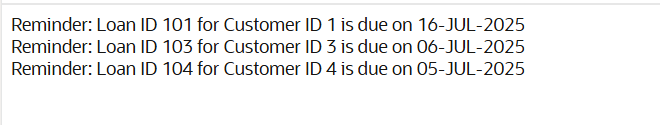
**' for Customer ID ' || loan.customer\_id ||**

**' is due on ' || TO\_CHAR(loan.due\_date, 'DD-MON-YYYY'));**

**END LOOP;**

**END;**

**OUTPUT:**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**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.**

**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.**

**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.**

**--------------------------------------------------------------------------------------------------------------------------------------**

**//SOLUTION:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**DATABASE : BANK1**

**TABLES : (I)** **savings\_accounts (II)** **employees (III)** **bank\_accounts**

**-- Savings Accounts Table**

**CREATE TABLE savings\_accounts (**

**account\_id NUMBER PRIMARY KEY,**

**customer\_id NUMBER,**

**balance NUMBER(12, 2)**

**);**

**-- Employees Table**

**CREATE TABLE employees (**

**emp\_id NUMBER PRIMARY KEY,**

**name VARCHAR2(50),**

**department VARCHAR2(30),**

**salary NUMBER(10, 2)**

**);**

**-- Bank Accounts Table (for fund transfer)**

**CREATE TABLE bank\_accounts (**

**account\_id NUMBER PRIMARY KEY,**

**customer\_id NUMBER,**

**balance NUMBER(12, 2)**

**);**

**--------INSERTION OF VALUES-------**

**savings\_accounts – For Monthly Interest**

**INSERT INTO savings\_accounts VALUES (201, 1, 10000.00);**

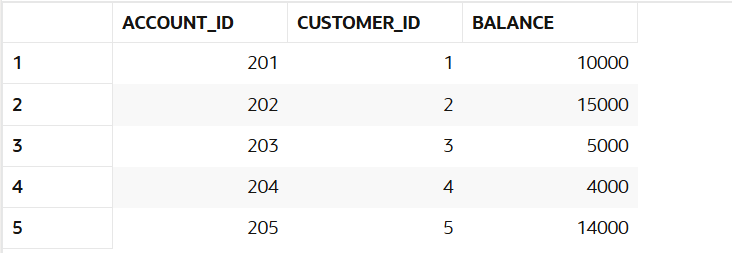
**INSERT INTO savings\_accounts VALUES (202, 2, 15000.00);**

**INSERT INTO savings\_accounts VALUES (203, 3, 5000.00);**

**INSERT INTO savings\_accounts VALUES (204,4,4000.00);**

**INSERT INTO savings\_accounts VALUES (205,5,14000.00);**

**SELECT \* FROM savings\_accounts;**

**OUTPUT:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**employees – For Bonus Update**

**INSERT INTO employees VALUES (301, 'Amit Sharma', 'Sales', 50000.00);**

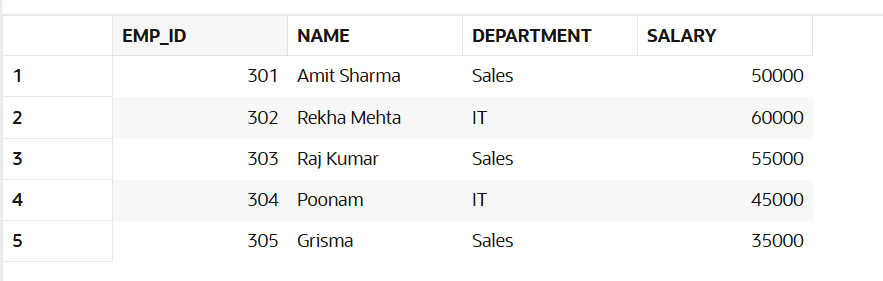
**INSERT INTO employees VALUES (302, 'Rekha Mehta', 'IT', 60000.00);**

**INSERT INTO employees VALUES (303, 'Raj Kumar', 'Sales', 55000.00);**

**INSERT INTO employees VALUES (304,'Poonam','IT',45000.00);**

**INSERT INTO employees VALUES (305,'Grisma''Sales',35000.00);**

**SELECT \* FROM employees;**

**OUTPUT:**

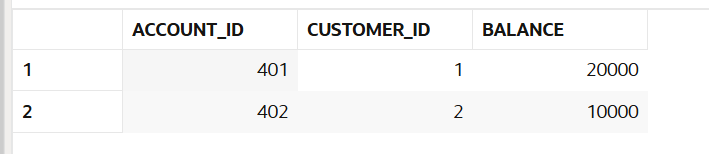
**--------------------------------------------------------------------------------------------------------------------------------------**

**bank\_accounts – For Funds Transfer**

**INSERT INTO bank\_accounts VALUES (401, 1, 20000.00); -- Source**

**INSERT INTO bank\_accounts VALUES (402, 2, 10000.00); -- Destination**

**OUTPUT:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 1: Process Monthly Interest**

**PL/SQL**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS**

**BEGIN**

**FOR acc IN (SELECT account\_id, balance FROM savings\_accounts) LOOP**

**UPDATE savings\_accounts**

**SET balance = acc.balance + (acc.balance \* 0.01)**

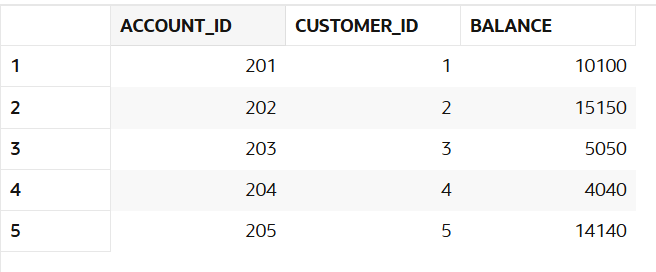
**WHERE account\_id = acc.account\_id;**

**END LOOP;**

**COMMIT;**

**END;**

**OUTPUT:**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 2: Update Employee Bonus:** **Add bonus (as %) to employees in a given department.**

**PL/SQL**

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (**

**p\_department VARCHAR2,**

**p\_bonus\_pct NUMBER -- Example: pass 0.10 for 10%**

**) IS**

**BEGIN**

**UPDATE employees**

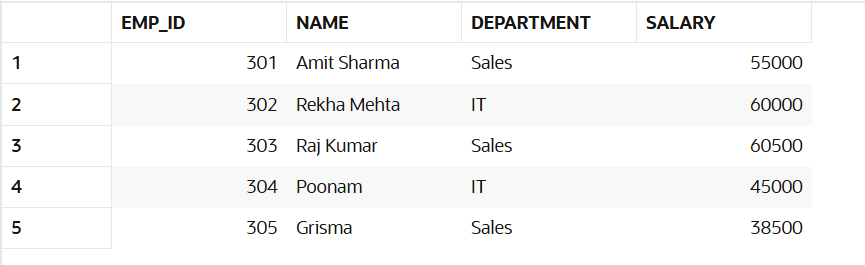
**SET salary = salary + (salary \* p\_bonus\_pct)**

**WHERE department = p\_department;**

**COMMIT;**

**END;**

**OUTPUT:**

****

**--------------------------------------------------------------------------------------------------------------------------------------**

**Scenario 3: TransferFunds(401, 402, 5000):** **Transfer ₹5000 from account 401 ➝ 402. Only if sufficient balance.**

**PL/SQL**

**CREATE OR REPLACE PROCEDURE TransferFunds (**

**p\_from\_account NUMBER,**

**p\_to\_account NUMBER,**

**p\_amount NUMBER**

**) IS**

**v\_balance NUMBER;**

**BEGIN**

**-- Check balance of source account**

**SELECT balance INTO v\_balance**

**FROM bank\_accounts**

**WHERE account\_id = p\_from\_account;**

**IF v\_balance < p\_amount THEN**

**RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in source account.');**

**END IF;**

**-- Deduct from source**

**UPDATE bank\_accounts**

**SET balance = balance - p\_amount**

**WHERE account\_id = p\_from\_account;**

**-- Add to destination**

**UPDATE bank\_accounts**

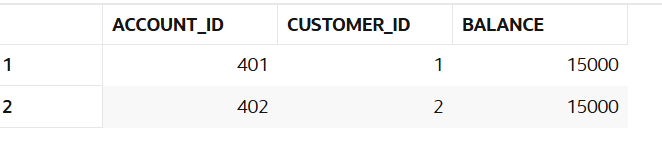
**SET balance = balance + p\_amount**

**WHERE account\_id = p\_to\_account;**

**COMMIT;**

**END;**

**OUTPUT:**

****

**--------------------------------------------------------------------------------------------------------------------------------------JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

**Scenario:**

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

**Steps:**

**1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).**

**2. Add JUnit dependency to your project. If you are using Maven, add the following to your**

**pom.xml:**

**<dependency>**

**<groupId>junit</groupId>**

**<artifactId>junit</artifactId>**

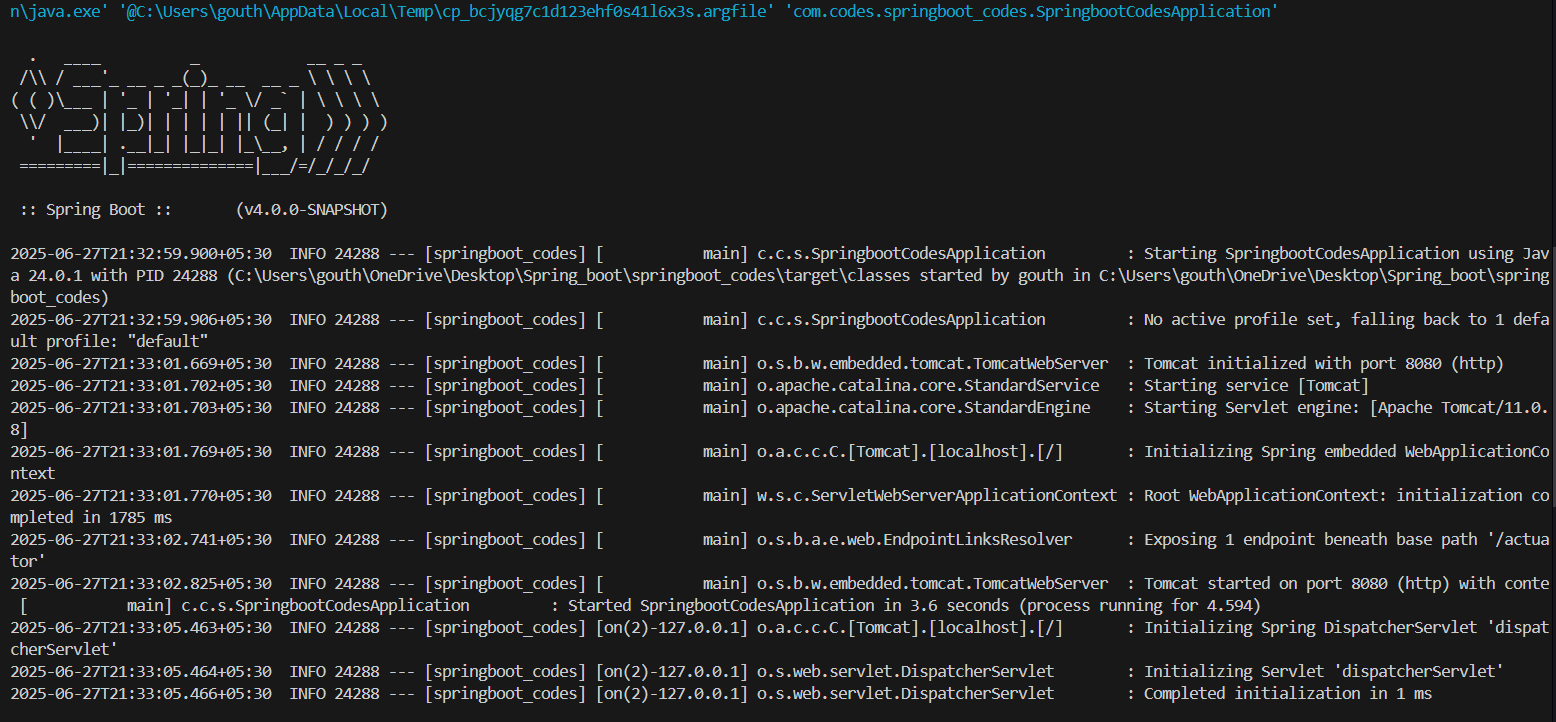
**<version>4.13.2</version>**

**<scope>test</scope>**

**</dependency>**

**3. Create a new test class in your project.**

**// OUTPUT:**

**-------------------------------------------------------------------------------------------------------------------------------------- Exercise 3: Assertions in Junit**

Scenario:

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

Steps:

1. Write tests using various JUnit assertions. Solution Code:

Solution Code:

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);

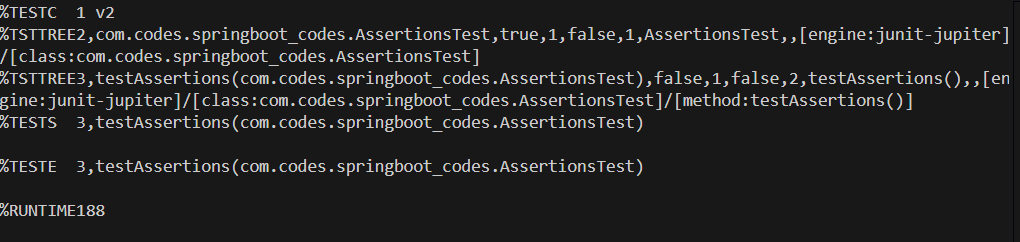
// Assert not null

assertNotNull(new Object());

}

}

**//OUTPUT:**

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

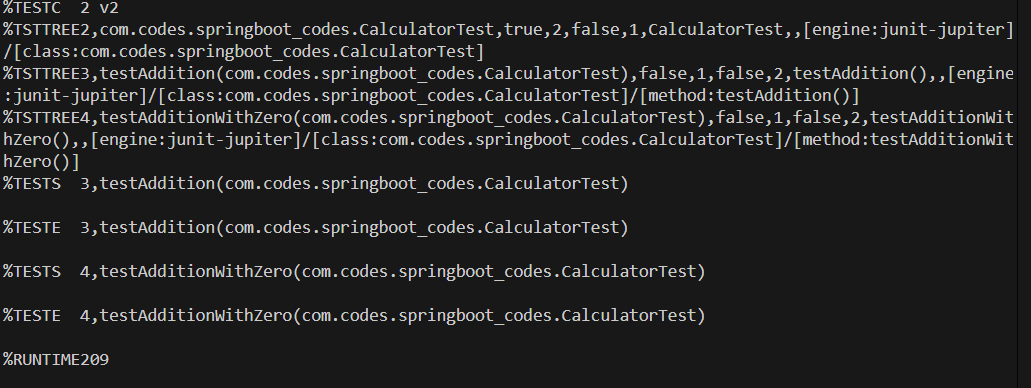
Scenario:

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.

**//OUTPUT:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**Mockito exercises;**

**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.

Solution Code:

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");

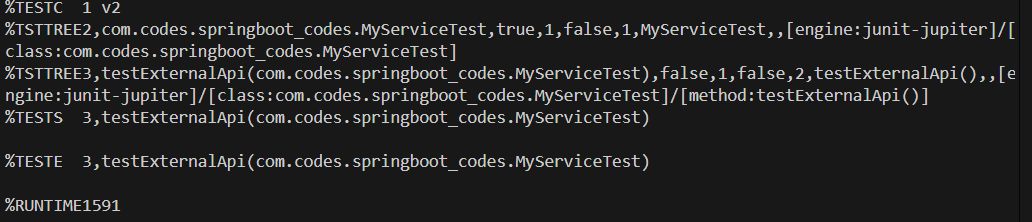
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**//OUTPUT:**

**--------------------------------------------------------------------------------------------------------------------------------------**

**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.

Solution Code:

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);

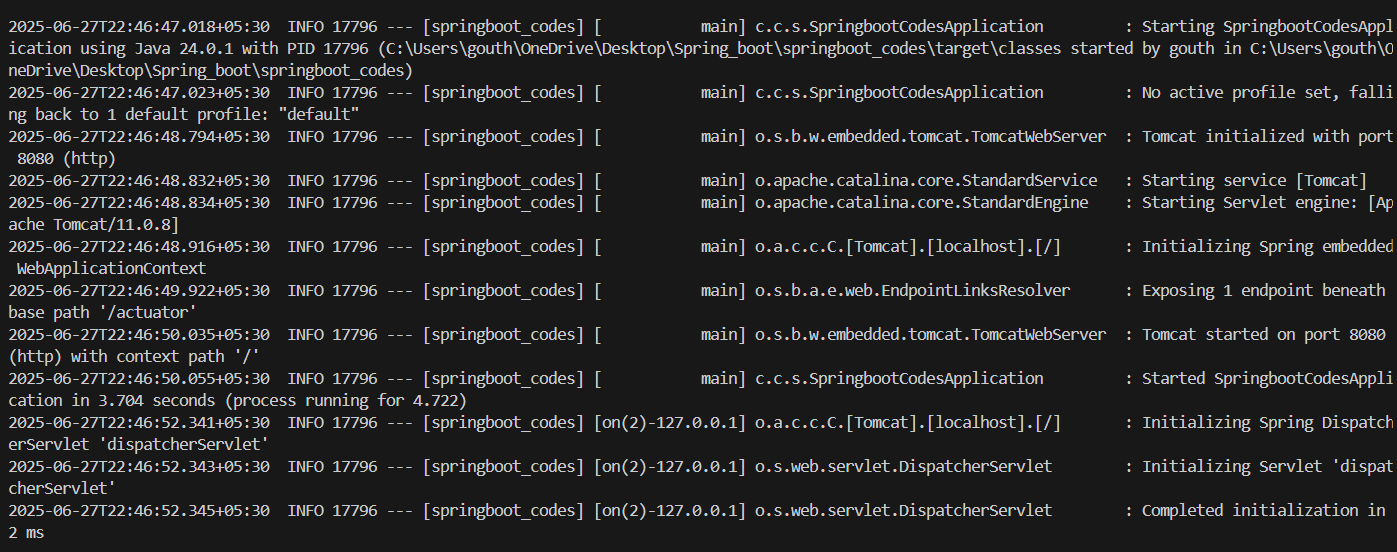
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**//OUTPUT:**

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

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

<dependency>

<groupld>org.slf4j</groupld>

<artifactld>slf4j-api</artifactid>

<version>1.7.30</version>

</dependency>

<dependency>

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

<artifactld>logback-classic</artifactld>

<version>1.2.3</version>

</dependency> //Create java class

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

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

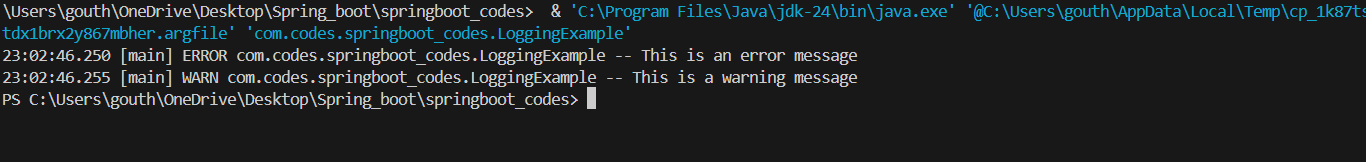
public static void main(String[] args) {

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

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

}

}

**//OUTPUT:**