PL/SQL

EXERCISE 1:

1. IF...THEN

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

DECLARE
    salary NUMBER := 15000;
BEGIN
    IF salary > 10000 THEN
        DBMS_OUTPUT.PUT_LINE('High Salary');
    END IF;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
         salary NUMBER := 15000;
     BEGIN
         IF salary > 10000 THEN...
Show more...


High Salary


PL/SQL procedure successfully completed.
```

2. IF...THEN...ELSE

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

DECLARE
    salary NUMBER := 12000;
BEGIN
    IF salary > 10000 THEN
        DBMS_OUTPUT.PUT_LINE('High Salary');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Normal Salary');
    END IF;
END;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
         salary NUMBER := 12000;
     BEGIN
         IF salary > 10000 THEN...
Show more...


High Salary


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.006
```

2. 3. IF...ELSIF...ELSE

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;
DECLARE
    salary NUMBER := 12000;
BEGIN
    IF salary > 20000 THEN
        DBMS_OUTPUT.PUT_LINE('Very High Salary');
    ELSIF salary > 10000 THEN
        DBMS_OUTPUT.PUT_LINE('High Salary');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Average Salary');
    END IF;
END;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
         salary NUMBER := 12000;
     BEGIN
         IF salary > 20000 THEN...
Show more...


High Salary


PL/SQL procedure successfully completed.
```

4. Simple CASE Statement

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;
DECLARE
    grade CHAR(1) := 'B';
BEGIN
    CASE grade
        WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
        WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Good');
        WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Fair');
        ELSE DBMS_OUTPUT.PUT_LINE('Needs Improvement');
    END CASE;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
         grade CHAR(1) := 'B';
     BEGIN
         CASE grade...
Show more...


Good


PL/SQL procedure successfully completed.
```

5. Searched CASE Statement

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;
DECLARE
    marks NUMBER := 78;
BEGIN
    CASE
        WHEN marks >= 90 THEN DBMS_OUTPUT.PUT_LINE('A Grade');
        WHEN marks >= 75 THEN DBMS_OUTPUT.PUT_LINE('B Grade');
        WHEN marks >= 60 THEN DBMS_OUTPUT.PUT_LINE('C Grade');
        ELSE DBMS_OUTPUT.PUT_LINE('Fail');
    END CASE;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
       marks NUMBER := 78;
     BEGIN
       CASE ...
Show more...

B Grade

PL/SQL procedure successfully completed.
```

6. LOOP (Basic loop)

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

DECLARE
    i NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('i = ' || i);
        i := i + 1;
        EXIT WHEN i > 5;
    END LOOP;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

7. WHILE loop

```
SET SERVEROUTPUT ON;

DECLARE
    i NUMBER := 1;
BEGIN
    WHILE i <= 7 LOOP
        DBMS_OUTPUT.PUT_LINE('i = ' || i);
        i := i + 1;
    END LOOP;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7

PL/SQL procedure successfully completed.
```

8. FOR loop

```
SET SERVEROUTPUT ON;

BEGIN
    FOR i IN 1..5 LOOP
        DBMS_OUTPUT.PUT_LINE('i = ' || i);
    END LOOP;
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
i = 1
i = 2
i = 3
i = 4
i = 5

PL/SQL procedure successfully completed.
```

EXERCISE 2: *STORED PROCEDURE* **What is a**

Stored Procedure?

A stored procedure is a named block of code that performs a task and is stored in the Oracle database. You can call it whenever needed.

1. Simple Stored Procedure: Basic procedure with no parameters.

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

BEGIN
  -- Inner procedure declaration (not stored permanently)
  PROCEDURE say_hello IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('Hello from PL/SQL Procedure!');
  END;
END;
/
  -- Call the procedure
  say_hello;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> BEGIN
       DBMS_OUTPUT.PUT_LINE('Hello from PL/SQL Procedure!');
     END;


Hello from PL/SQL Procedure!


PL/SQL procedure successfully completed.
```

2. Procedure with IN Parameter: Learn how to pass input to a procedure

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

DECLARE
    PROCEDURE greet_user(name IN VARCHAR2) IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Hello, ' || name || '!');
    END;
BEGIN
    greet_user('Shalu');
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
        PROCEDURE greet_user(name IN VARCHAR2) IS
        BEGIN
            DBMS_OUTPUT.PUT_LINE('Hello, ' || name || '!');...
Show more...


Hello, Shalu!


PL/SQL procedure successfully completed.
```

3. Procedure with IN and OUT Parameters: Learn how to return values from a procedure using OUT.

[SQL Worksheet]*

```
SET SERVEROUTPUT ON;

DECLARE
    result NUMBER;

    PROCEDURE get_square(num IN NUMBER, square OUT NUMBER) IS
    BEGIN
        square := num * num;
    END;
BEGIN
    get_square(6, result);
    DBMS_OUTPUT.PUT_LINE('Square is: ' || result);
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
        result NUMBER;  -- Declare variable first

        PROCEDURE get_square(num IN NUMBER, square OUT NUMBER) IS...
Show more...


Square is: 36


PL/SQL procedure successfully completed.
```

4. Procedure with DML (Insert into a table)

```
SET SERVEROUTPUT ON;
-- Create the table if not exists (handled safely)
BEGIN
    EXECUTE IMMEDIATE '
        CREATE TABLE students (
            id NUMBER PRIMARY KEY,
            name VARCHAR2(50)
        )';
EXCEPTION
    WHEN OTHERS THEN
        IF SQLCODE = -955 THEN
            NULL; --Table already exists
        ELSE
            RAISE;
        END IF;
END;
/
DECLARE
    PROCEDURE insert_student(sid IN NUMBER, sname IN VARCHAR2) IS
    BEGIN
        INSERT INTO students(id, name) VALUES (sid, sname);
        DBMS_OUTPUT.PUT_LINE('Student inserted: ' || sname);
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Student ID already exists for ' || sname);
    END;
BEGIN
    insert_student(101, 'Amit');
    insert_student(101, 'Ravi');
END;
/
```

Query result | Script output | DBMS output | Explain Plan | SQL history

```
SQL> DECLARE
        PROCEDURE insert_student(sid IN NUMBER, sname IN VARCHAR2) IS
        BEGIN
            INSERT INTO students(id, name) VALUES (sid, sname);...
Show more...


Student ID already exists for Anu
Student ID already exists for Ravi


PL/SQL procedure successfully completed.
```

5. Procedure with exception handling

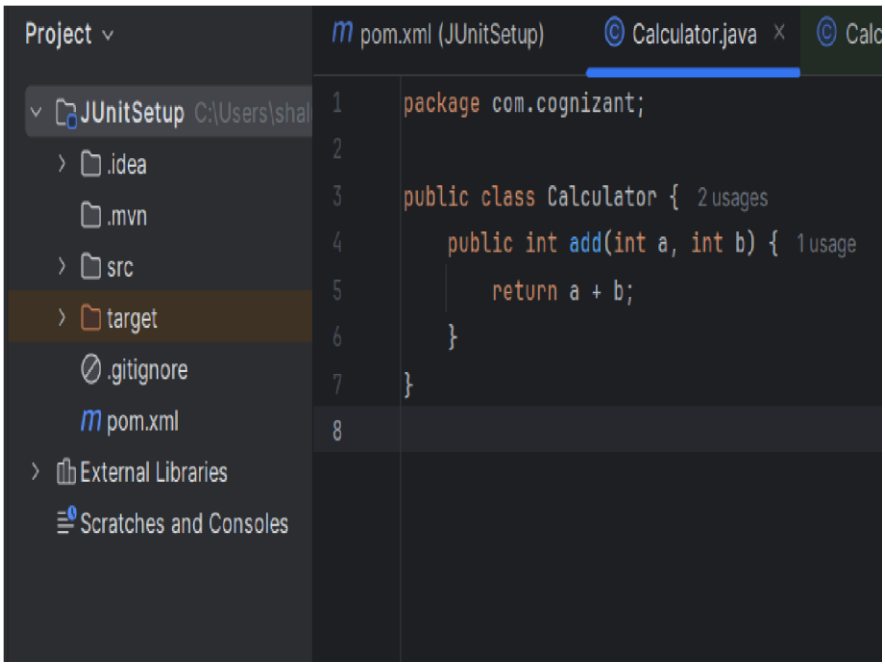
```
DECLARE
    PROCEDURE divide_numbers(a IN NUMBER, b IN NUMBER) IS
        result NUMBER;
    BEGIN
        result := a / b;
        DBMS_OUTPUT.PUT_LINE('Result: ' || result);
    EXCEPTION
        WHEN ZERO_DIVIDE THEN
            DBMS_OUTPUT.PUT_LINE('Division by zero not allowed.');
    END;
BEGIN
    divide_numbers(10, 2);  -- Valid
    divide_numbers(10, 0);  -- Triggers exception
END;
/
```

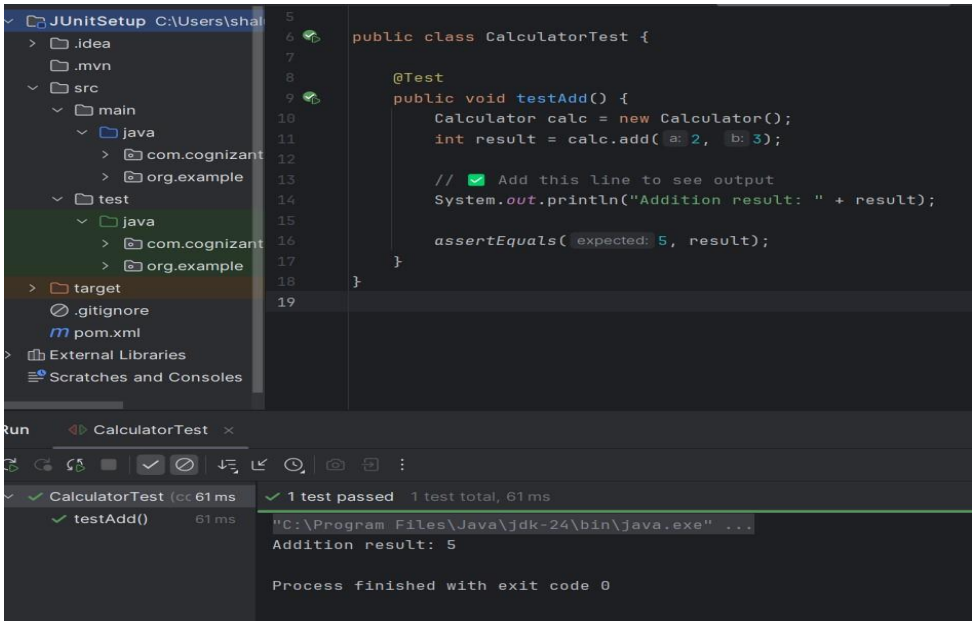
Query result | Script output | DBMS output | Explain Plan | SQL history

```
        result NUMBER;
    BEGIN...
Show more...


Result: 5
Division by zero not allowed.


PL/SQL procedure successfully completed.
```

TDD using JUnit5 and Mockito

Exercise 1: Setting Up JUnit

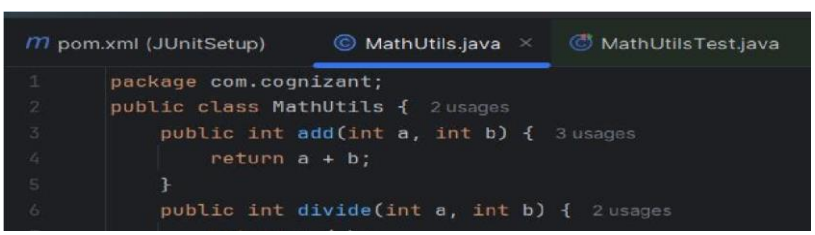




Exercise 2: Assertions in JUnit

Assertion, Use case

- AssertEquals: Compare expected and actual values
- AssertNotNull: ensure object is not null
- AssertArrayEquals: Compare arrays
- AssertThrows: Test if exception is thrown
- AssertTrue/False: Validate boolean conditions



pom.xml (JUnitSetup) | MathUtils.java | MathUtilsTest.java | App

```java
package com.cognizant;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class MathUtilsTest {
    MathUtils utils = new MathUtils();  7 usages
    @Test
    void testAddition() {
        int result = utils.add( a: 2,  b: 3);
        System.out.println("Addition Result: " + result);
        assertEquals( expected: 5, result);
    }
    @Test
    void testDivide() {
        int result = utils.divide( a: 6,  b: 3);
        System.out.println("Division Result: " + result);
        assertEquals( expected: 2, result);
    }
    @Test
    void testGreeting() {
        String greeting = utils.getGreeting();
        System.out.println("Greeting: " + greeting);
        assertNotNull(greeting);
        assertEquals( expected: "Hello, JUnit!", greeting);
    }
```

pom.xml (JUnitSetup) | MathUtils.java | MathUtilsTest.java | AppTest.java

```java
public class MathUtilsTest {
    void testGreeting() {
        System.out.println("Greeting: " + greeting);
        assertNotNull(greeting);
        assertEquals( expected: "Hello, JUnit!", greeting);
    }
    @Test
    void testArrayEquality() {
        int[] expected = {1, 2, 3};
        int[] actual = utils.getArray();
        System.out.println("Array Content: " + java.util.Arrays.toString(actual));
        assertArrayEquals(expected, actual);
    }
    @Test
    void testException() {
            System.out.println("Expecting ArithmeticException for divide by zero");
        assertThrows(ArithmeticException.class, () -> utils.divide( a: 10,  b: 0));
    }
    @Test
    void testTrueCondition() {
        boolean condition = utils.add( a: 2,  b: 2) == 4;
        System.out.println("True Condition Passed: " + condition);
        assertTrue(condition);
    }
```

pom.xml (JUnitSetup) | MathUtils.java | MathUtilsTest.java | AppTest.jav

```java
public class MathUtilsTest {
    void testTrueCondition() {
        boolean condition = utils.add( a: 2,  b: 2) == 4;
        System.out.println("True Condition Passed: " + condition);
        assertTrue(condition);
    }
    @Test
    void testFalseCondition() {
        boolean condition = utils.add( a: 2,  b: 2) == 5;
        System.out.println("False Condition Passed: " + !condition);
        assertFalse(condition);
    }
}
```

✓ 7 tests passed 7 tests total, 155 ms

```
"C:\Program Files\Java\jdk-24\bin\java.exe" ...
Expecting ArithmeticException for divide by zero
True Condition Passed: true
False Condition Passed: true
Addition Result: 5
Greeting: Hello, JUnit!
Division Result: 2
Array Content: [1, 2, 3]

Process finished with exit code 0
```

Exercise 3: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods

The **Arrange-Act-Assert (AAA)** Pattern helps organize each test case into three clear steps: first, you arrange or set up the necessary data and environment; second, you act by calling the method under test; and finally, you assert the expected outcome. A test fixture refers to reusable setup code, such as creating a Bank Account object before each test, which ensures consistency and avoids duplication. The **@BeforeEach** annotation is used to execute setup code before each test method runs, making sure every test starts with a clean state. Similarly, **@AfterEach** is used to define any cleanup or logging that should happen after each test completes. These practices together make your tests more readable, maintainable, and reliable.

pom.xml (JUnitSetup) | BankAccount.java | BankAccountTest.java | AppTest.java

```java
package com.cognizant;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
public class BankAccountTest {
    BankAccount account;  7 usages
    @BeforeEach
    void setUp() {
        account = new BankAccount(); // Arrange (Fixture)
        System.out.println("Setup complete."); }
    @AfterEach
    void tearDown() {
        System.out.println("Test complete.\n");}
    @Test
    void testDeposit() {
        account.deposit( amount: 100);                // Act
        assertEquals( expected: 100, account.getBalance()); } // Assert
    @Test
    void testWithdraw() {
        account.deposit( amount: 200);
        account.withdraw( amount: 50);               // Act
        assertEquals( expected: 150, account.getBalance());} //Assert
    @Test
    void testWithdrawThrows() {
        assertThrows(IllegalArgumentException.class, () -> account.withdraw( amount: 500)); //Assert
    }
}
```

pom.xml (JUnitSetup) | BankAccount.java | BankAccountTest.java | AppTest.java

```java
package com.cognizant;

public class BankAccount {  no usages
    private double balance = 0;  4 usages

    public void deposit(double amount) { balance += amount; }  no usages

    public void withdraw(double amount) {  no usages
        if (amount > balance) throw new IllegalArgumentException("Insufficient funds");
        balance -= amount;
    }

    public double getBalance() { return balance; }  no usages
}
```

```
✓ 3 tests passed  3 tests total, 70 ms
"C:\Program Files\Java\jdk-24\bin\j
Setup complete.
Test complete.

Setup complete.
Test complete.

Setup complete.
Test complete.


Process finished with exit code 0
```

Mockito exercises

Exercise 1: Mocking and Stubbing

pom.xml (MockitoExample) | UserRepository.java | UserService.java | UserServiceTest.java | AppTest.java

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>MockitoExample</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>MockitoExample</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
```

pom.xml (MockitoExample) | UserRepository.java | UserSe

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xml
  <dependencies>
    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.10.0</version>
        <scope>test</scope>
      </dependency>
    <!-- Mockito -->
    <dependency>
      <groupId>org.mockito</groupId>
      <artifactId>mockito-core</artifactId>
      <version>5.11.0</version>
      <scope>test</scope>
    </dependency>

  </dependencies>
</project>
```

pom.xml (MockitoExample) | UserRepository.java

```
package com.cognizant;

public interface UserRepository {  no usages
    String getUsernameById(int id);  no usages
}
```

pom.xml (MockitoExample) | UserRepository.java | UserService.java

```java
package com.cognizant;
public class UserService {  2 usages
    private UserRepository repository;  2 usages
    public UserService(UserRepository repository) {  1 usage
        this.repository = repository; }
    public String getFormattedUsername(int id) {  1 usage
        String name = repository.getUsernameById(id);
        return "User: " + name; }
}
```

pom.xml (MockitoExample) | UserRepository.java | UserService.java | UserServ

```java
package com.cognizant;

import org.junit.jupiter.api.Test;
import org.mockito.Mockito;
import static org.junit.jupiter.api.Assertions.assertEquals;
public class UserServiceTest {
    @Test
    void testGetFormattedUsername() {
        UserRepository mockRepo = Mockito.mock(UserRepository.class);
        Mockito.when(mockRepo.getUsernameById(1)).thenReturn( t: "Shalu");
        // Inject mock into service
        UserService service = new UserService(mockRepo);
        // Act
        String result = service.getFormattedUsername( id: 1);
        // Assert
        assertEquals( expected: "User: Shalu", result);
        System.out.println("Test passed with result: " + result); }
}
```

```
Test passed with result: User: Shalu

Process finished with exit code 0
```

Exercise 2: **Verifying Interactions**

In Mockito, verifying interactions helps ensure that specific methods were called (or not called) on mocked dependencies. This is crucial when testing behavior — especially in scenarios like sending notifications, updating databases, or logging. For instance, after a login() method is called, we may want to confirm that a notification service's send() method was triggered exactly once. Mockito provides verify() for this purpose, and it also allows checking that no more or unwanted interactions occurred using methods like verifyNoMoreInteractions() or verifyNever().

Example: Verifying Login Notification with Mockito

pom.xml (MockitoExample) × NotificationService.java ×

```java
package com.cognizant;

public class NotificationService {  no usages
    public void send(String message) {  no usages
        System.out.println("Sending: " + message);
    }
}
```

pom.xml (MockitoExample) NotificationService.java User.java

```java
package com.cognizant;
public class User {  4 usages
    private NotificationService notificationService;  2 usage

    public User(NotificationService notificationService) {
        this.notificationService = notificationService;
    }
    public void login() {  2 usages
        notificationService.send("Login Successful");
    }
}
```

pom.xml (MockitoExample) NotificationService.java UserTest.java × AppTes

```java
package com.cognizant;

import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.*;
public class UserTest {
    @Test
    public void testLoginSendsNotification() {
        NotificationService mockService = mock(NotificationService.class);
        User user = new User(mockService);
        user.login();
        // Test file
        verify(mockService).send("Login Successful"); }
    @Test
    public void testLoginDoesNotSendOtherMessage() {
        NotificationService mockService = mock(NotificationService.class);
        User user = new User(mockService);
        user.login();
        verify(mockService, never()).send("Wrong Message");
    }
}
```

SL4J Logging exercises

Exercise 3: Logging Error Messages and Warning Levels

pom.xml (SLF4JExampleProject) | LoggerExample.java | AppTest.java

```java
package com.cognizant;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class LoggerExample {
    private static final Logger logger = LoggerFactory.getLogger(LoggerExample.class);
    public void performTask(int input) {  3 usages
        logger.info("Task started");
        if (input < 0) {
            logger.warn("Negative input provided: {}", input);
        }
        try {
            int result = 10 / input;
            logger.info("Computation result: {}", result);
        } catch (ArithmeticException e) {
            logger.error("Error occurred while computing: Division by zero", e);
        }
        logger.info("Task completed");
    }
    public static void main(String[] args) {
        LoggerExample example = new LoggerExample();
        example.performTask( input: 5);    // Normal
        example.performTask( input: 0);    // Will trigger error
        example.performTask( input: -2);   // Will trigger warning
    }
}
```

```
[main] INFO com.cognizant.LoggerExample - Task started
[main] INFO com.cognizant.LoggerExample - Computation result: 2
[main] INFO com.cognizant.LoggerExample - Task completed
[main] INFO com.cognizant.LoggerExample - Task started
[main] ERROR com.cognizant.LoggerExample - Error occurred while computing:
 Division by zero
java.lang.ArithmeticException Create breakpoint : / by zero
    at com.cognizant.LoggerExample.performTask(LoggerExample.java:18)
    at com.cognizant.LoggerExample.main(LoggerExample.java:30)
[main] INFO com.cognizant.LoggerExample - Task completed
[main] INFO com.cognizant.LoggerExample - Task started
[main] WARN com.cognizant.LoggerExample - Negative input provided: -2
[main] INFO com.cognizant.LoggerExample - Computation result: -5
[main] INFO com.cognizant.LoggerExample - Task completed

Process finished with exit code 0
```