**Candidate Name:ABHISHEK GOWDA M**

**Superset ID:- 6431229**

**WEEK – 2 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**(Test Driven Development)**

**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.

**Dependency added in pom.xml :**

A screenshot of a computer program

AI-generated content may be incorrect.

**Solution Code part :**

package com.example;  
  
public class Calculator

{  
 public int add(int a, int b)

{  
 return a + b;  
 }  
  
 public int subtract(int a, int b)

{  
 return a - b;  
 }  
 }

**Testing for the code part:**

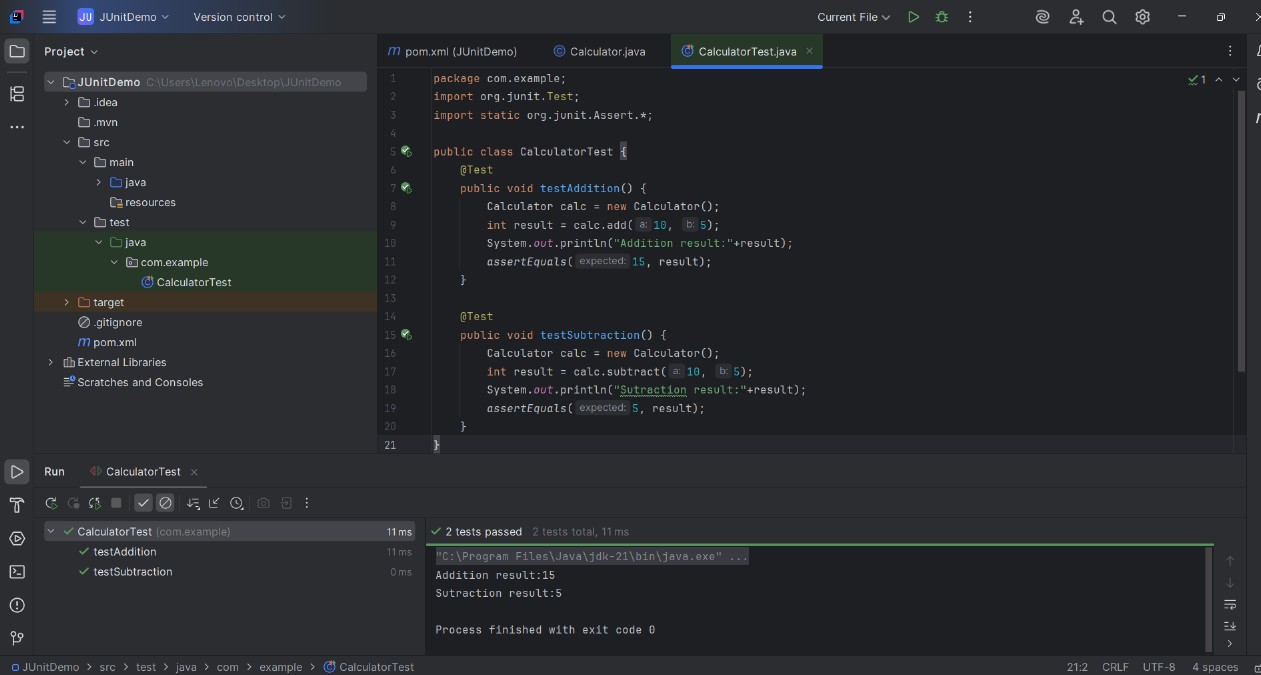
package com.example;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest

{  
  
 @Test  
 public void testAddition()

{  
 Calculator calc = new Calculator();  
 int result = calc.add(10, 5);  
 System.*out*.println("Addition result:"+result);  
 *assertEquals*(15, result);  
 }  
  
 @Test  
 public void testSubtraction()

{  
 Calculator calc = new Calculator();  
 int result = calc.subtract(10, 5);  
 System.*out*.println("Sutraction result:"+result);  
 *assertEquals*(5, result);  
 }  
 }

**Terminal Output For Test Pass Confirmation:**



**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:

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());

}

}

**Dependency added in pom.xml :**

A screenshot of a computer

AI-generated content may be incorrect.

**Solution Code Part :**

**AssertionDemoTest.java**

package com.testdemo;  
  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class AssertionDemoTest

{  
  
 @Test  
 public void checkAllAssertions()

{  
 System.*out*.println("Starting assertions...");

int sum = 2 + 3;  
 System.*out*.println("Computed sum of 2 and 3: " + sum);  
 *assertEquals*("Sum of 2 and 3 should be 5", 5, sum);  
  
 boolean greater = 5 > 3;  
 System.*out*.println("Is 5 greater than 3? " + greater);  
 *assertTrue*("5 is greater than 3", greater);  
  
  
 boolean less = 5 < 3;  
 System.*out*.println("Is 5 less than 3? " + less);  
 *assertFalse*("5 is not less than 3", less);  
  
  
 String text = null;  
 System.*out*.println("Value of text: " + text);  
 *assertNull*("Text should be null", text);  
  
  
 Object obj = new Object();  
 System.*out*.println("Value of obj: " + obj);  
 *assertNotNull*("Object should not be null", obj);  
  
  
 System.*out*.println("All assertions completed.");  
 }  
 }

**Assertion Purpose**

assertEquals Validates arithmetic or logic matches expected

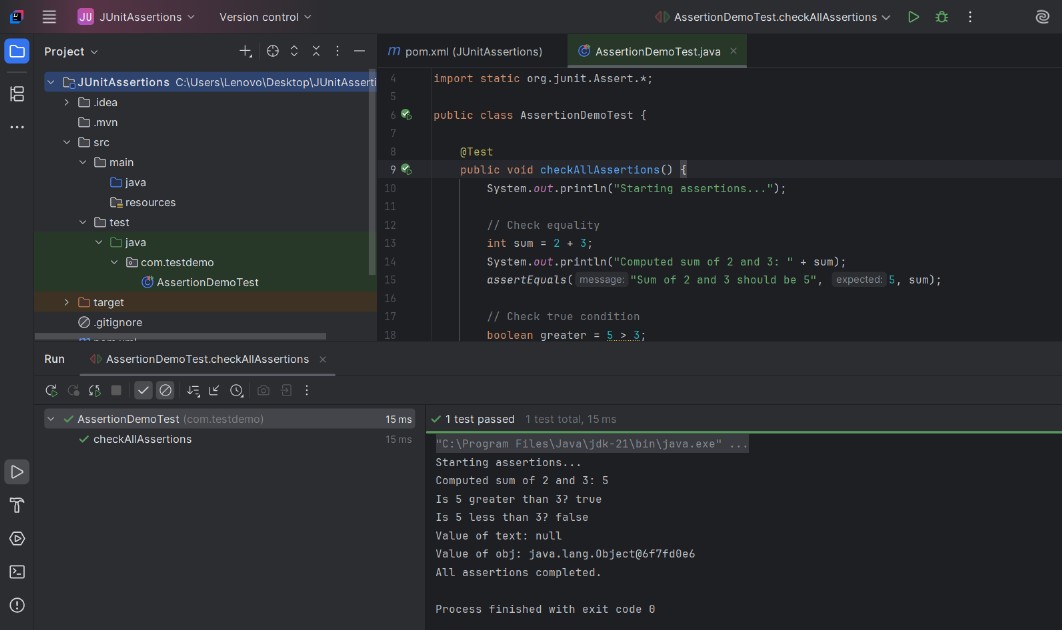
assertTrue Confirms condition is true

assertFalse Confirms condition is false

assertNull Ensures an object in null

assertNotNull Ensures an object is not null

**Terminal Output For Test Pass Confirmation:**



**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.

**Dependency added in pom.xml :**

A screenshot of a computer

AI-generated content may be incorrect.

**Solution Code part :**

package com.unittest.sample;  
  
public class SimpleCalculator

{  
  
 public int add(int a, int b)

{  
 return a + b;  
 }  
  
 public int subtract(int a, int b)

{  
 return a - b;  
 }

}

**Testing for the code part:**

package com.unittest.sample;  
  
import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class SimpleCalculatorTest

{  
  
 private SimpleCalculator calculator;  
  
 @Before  
 public void setUp()

{  
 calculator = new SimpleCalculator();  
 System.*out*.println("Setup: Calculator is ready");  
 }  
  
 @After  
 public void tearDown()

{  
 calculator = null;  
 System.*out*.println("Teardown: Calculator is reset");  
 }  
  
 @Test  
 public void testAddition()

{  
 // Act  
 int result = calculator.add(7, 3);  
 System.*out*.println("Addition result:"+result);  
 // Assert  
 *assertEquals*("7 + 3 should equal 10", 10, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 int result = calculator.subtract(9, 4);  
 System.*out*.println("Subtraction result:"+result);  
 *assertEquals*("9 - 4 should equal 5", 5, result);  
 }  
}

**Terminal Output For Test Pass Confirmation:**

A screenshot of a computer

AI-generated content may be incorrect.

**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);

}

}

**Dependency added in pom.xml :**

A screenshot of a computer program

AI-generated content may be incorrect.

**Solution Code Part :**

**1. ExternalApi.java**

public interface ExternalApi

{  
 String getData();  
 }

**2. MyService.java**

public class MyService

{  
 private ExternalApi api;  
  
 public MyService(ExternalApi api)

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

{  
 return api.getData();  
 }  
 }

**3. MyServiceTest.java**

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

}

**4. AppTest.java**

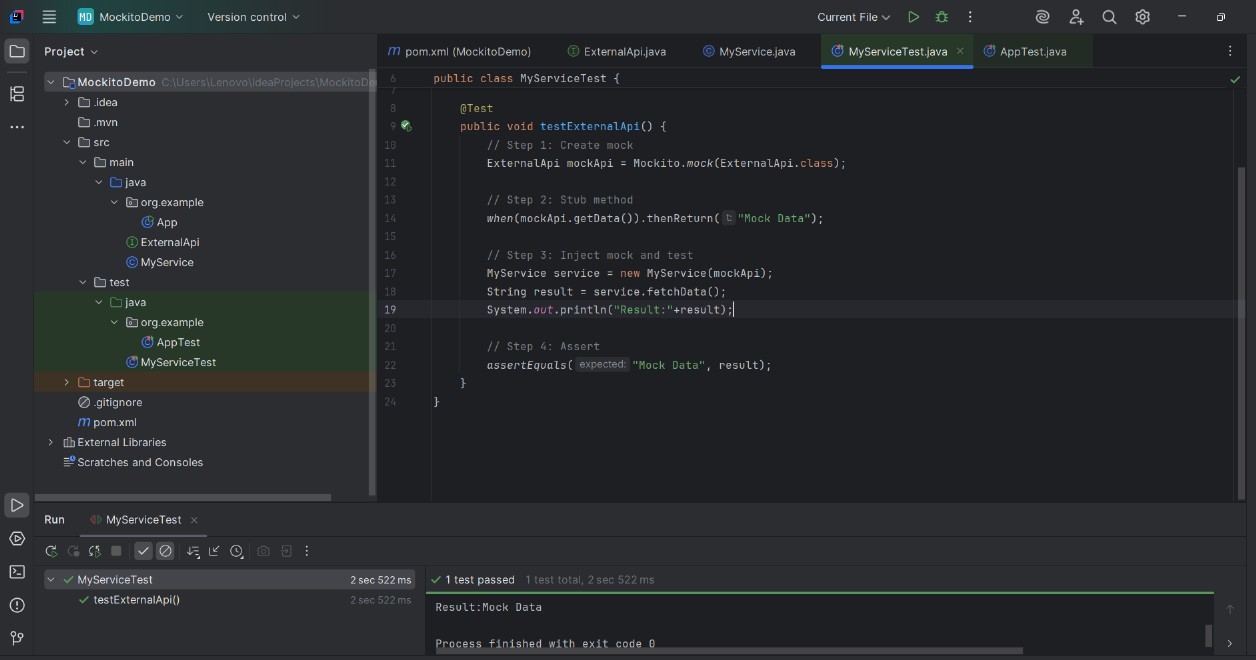
package org.example;

import static org.junit.jupiter.api.Assertions.*assertTrue*;  
  
import org.junit.jupiter.api.Test;  
  
public class AppTest   
{

@Test  
 public void shouldAnswerWithTrue()  
 {  
 *assertTrue*( true );  
 }

}

**Terminal Output For Test Pass Confirmation:**



**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);

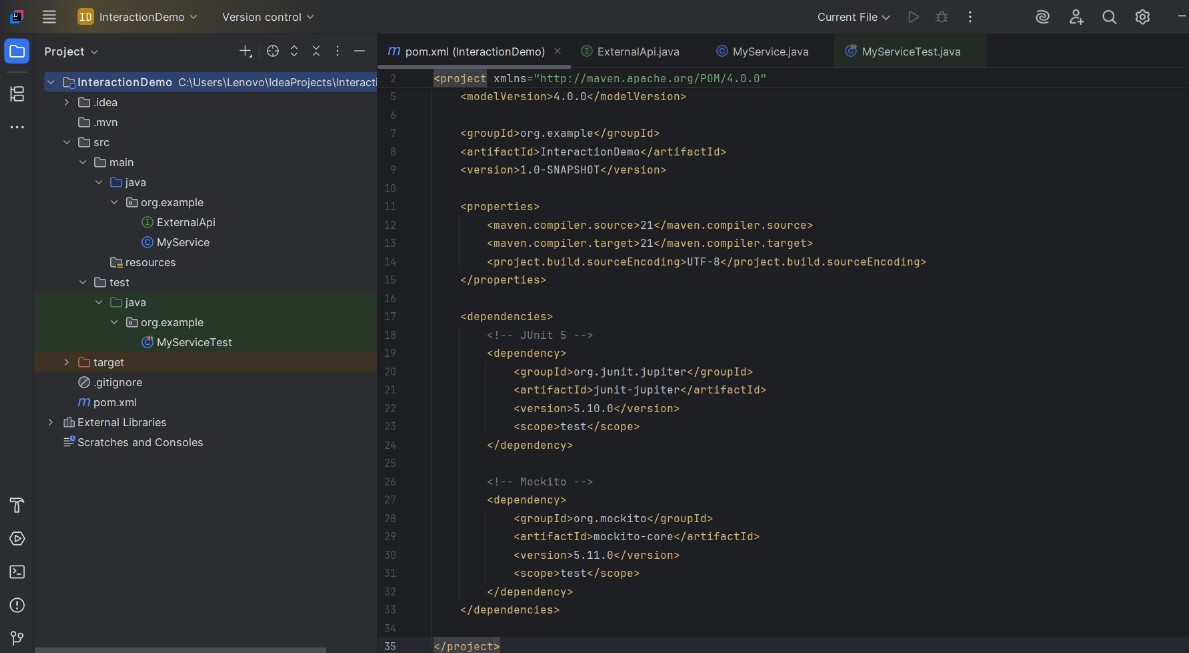
service.fetchData();

verify(mockApi).getData();

    }

}

**Dependency added in pom.xml :**



**Solution Code Part :**

**1. ExternalApi.java**

package org.example;  
  
 public interface ExternalApi

{  
  
 String getData(String token);  
 }

**2. MyService.java**

package org.example;  
  
 public class MyService

{  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api)

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

{  
 return api.getData("token123");   
 }  
 }

**3. MyServiceTest.java**

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test

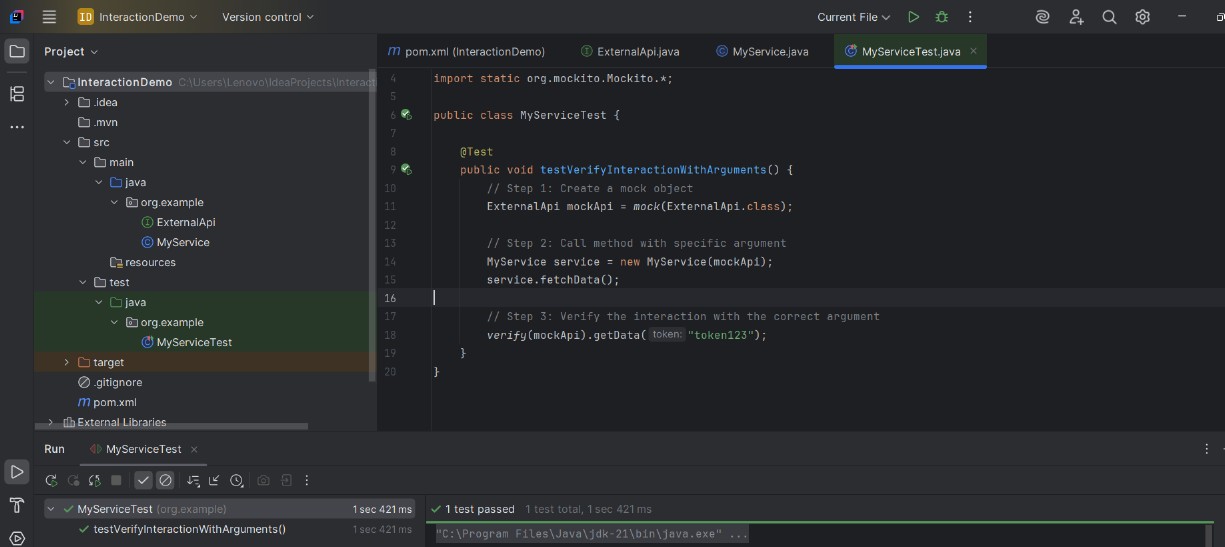
public void testVerifyInteractionWithArguments()

{  
 ExternalApi mockApi = *mock*(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();  
 *verify*(mockApi).getData("token123");  
 }  
 }

**Terminal Output For Test Pass Confirmation:**



**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.

**Step-by-Step Solution:**

**1. Add SLF4J and Logback dependencies to your pom.xml file:**

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

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

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

**2. Create a Java class that uses SLF4J for logging:**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

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

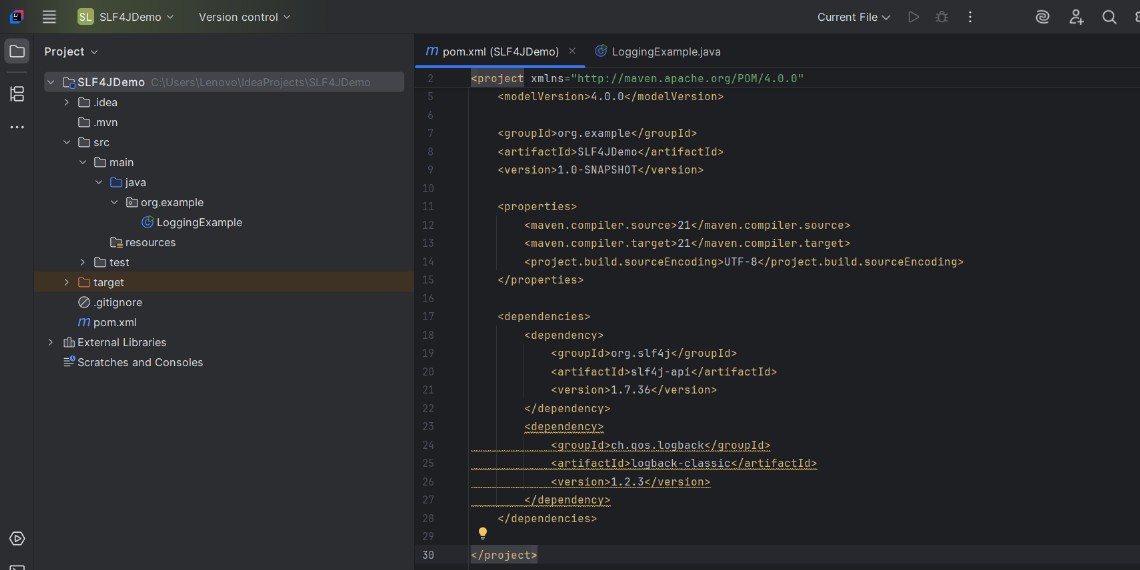
public static void main(String[] args) {

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

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

    } }

**Dependency added in pom.xml :**



**Solution Code Part :**

**LoggingExample.java**

package org.example;  
  
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");  
 }  
}

**Terminal Output For Test Pass Confirmation:**

A screenshot of a computer program

AI-generated content may be incorrect.