**JUnit Testing Exercises**

Exercise 1: Setting Up JUnit

Scenario:

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

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<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>JUnitCalculator</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>22</maven.compiler.source>  
 <maven.compiler.target>22</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- JUnit 4 for testing -->  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Surefire to run JUnit tests -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.22.2</version>  
 <configuration>  
 <includes>  
 <include>\*\*/\*Test.java</include>  
 </includes>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

Example.test

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class ExampleTest {

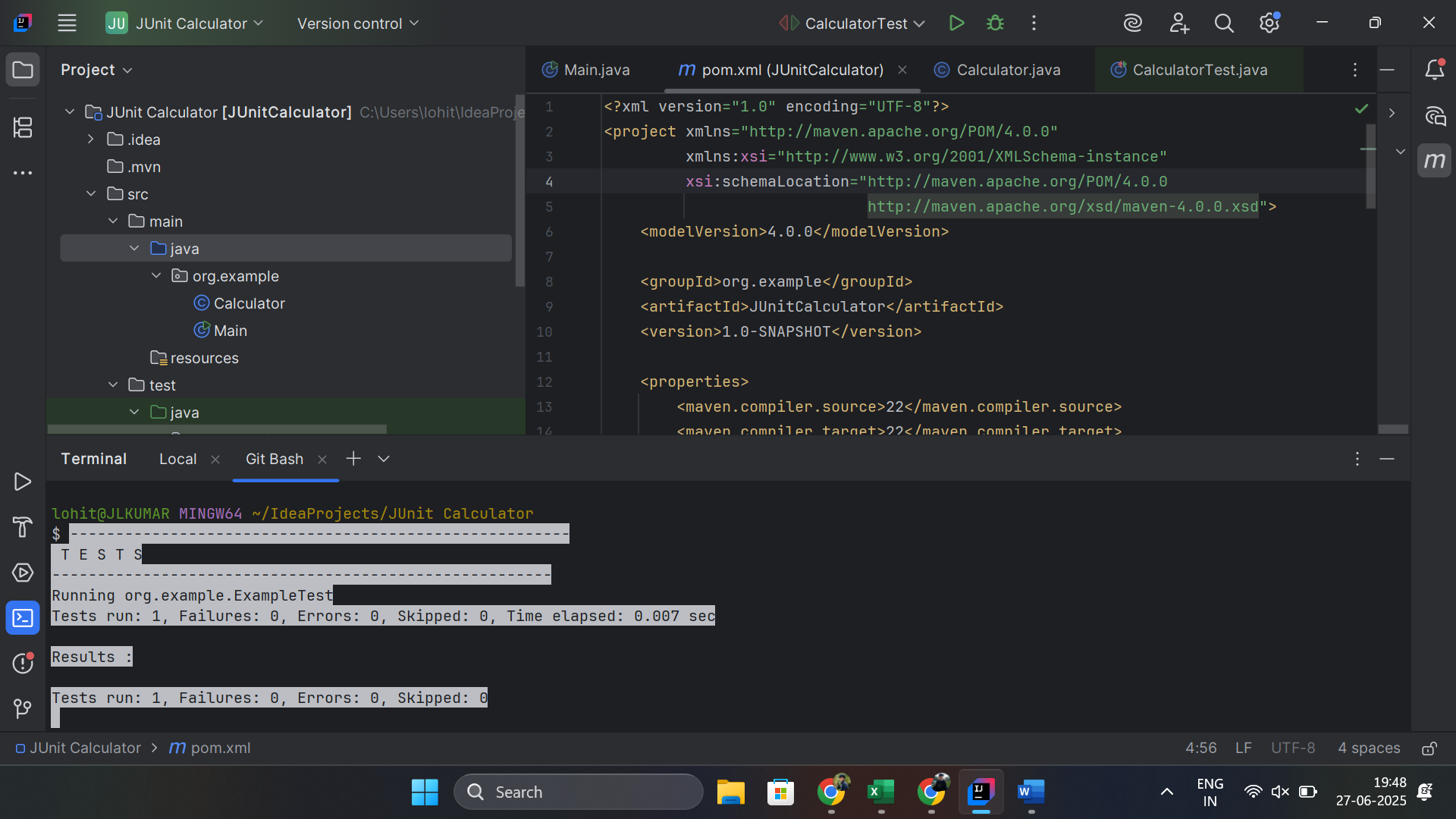
@Test

public void testJUnitSetup() {

assertTrue("JUnit is working", true);

}

}



Exercise 3: Assertions in JUnit

Scenario:

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

**AssertionsTest.java**

package org.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals("2 + 3 should equal 5", 5, 2 + 3);

assertTrue("5 > 3 should be true", 5 > 3);

assertFalse("5 < 3 should be false", 5 < 3);

Object nullObj = null;

assertNull("Object should be null", nullObj);

Object nonNullObj = new Object();

assertNotNull("Object should not be null", nonNullObj);

}

}

A screenshot of a computer

AI-generated content may be incorrect.

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.

**AssertionsCalculator.java**

**package org.example;**

**import org.junit.Before;**

**import org.junit.After;**

**import org.junit.Test;**

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

**public class AAACalculatorTest {**

**private Calculator calc;**

**@Before**

**public void setUp() {**

**calc = new Calculator();**

**}**

**@After**

**public void tearDown() {**

**calc = null;**

**}**

**@Test**

**public void testAdd\_AAA() {**

**int result = calc.add(7, 8);**

**assertEquals(15, result);**

**}**

**@Test**

**public void testDivideByZero\_AAA() {**

**try {**

**calc.divide(5, 0);**

**fail("Expected ArithmeticException when dividing by zero");**

**} catch (ArithmeticException ex) {**

**assertEquals("/ by zero", ex.getMessage());**

**}**

**}**

**}**

A screenshot of a computer

AI-generated content may be incorrect.

**3. Mockito exercises**

Mockito Hands-On 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.

**External Interface.java**

package org.example;

public interface ExternalApi {

String getData();

}

**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();

}

}

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:

**MyServiceTest.java**

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
class MyServiceTest {  
  
 @Test  
 void testMockingAndStubbing() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 *assertEquals*("Mock Data", result);  
 }  
  
 @Test  
 void testVerifyInteraction() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
  
 service.fetchData();  
  
 *verify*(mockApi).getData();  
 }  
}

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();

}

}

