**JUnit Testing Exercises**

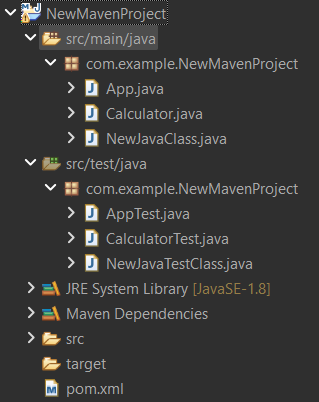
Exercise 1: Setting Up JUnit

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).



Steps:

1. Go to File > New >Project…
2. Select Maven project
3. Click next
4. Uncheck “Create a simple project (skip archetype selection)”
5. Choose an **archetype** (template) like maven-archetype-quickstart.
6. Click Next
7. Give Group Id and Artifact Id
8. Add required dependencies in pom.xml
9. Right click on the project > Maven > Update Project…
10. Check “Force Update Of Snapshots/Releases.”
11. Click OK
12. Check Maven Dependecies Folder

You can see:

1. A **"Maven Dependencies"** folder in the Project Explorer.
2. JUnit and other jars listed there.

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

pom.xml:

<dependency>

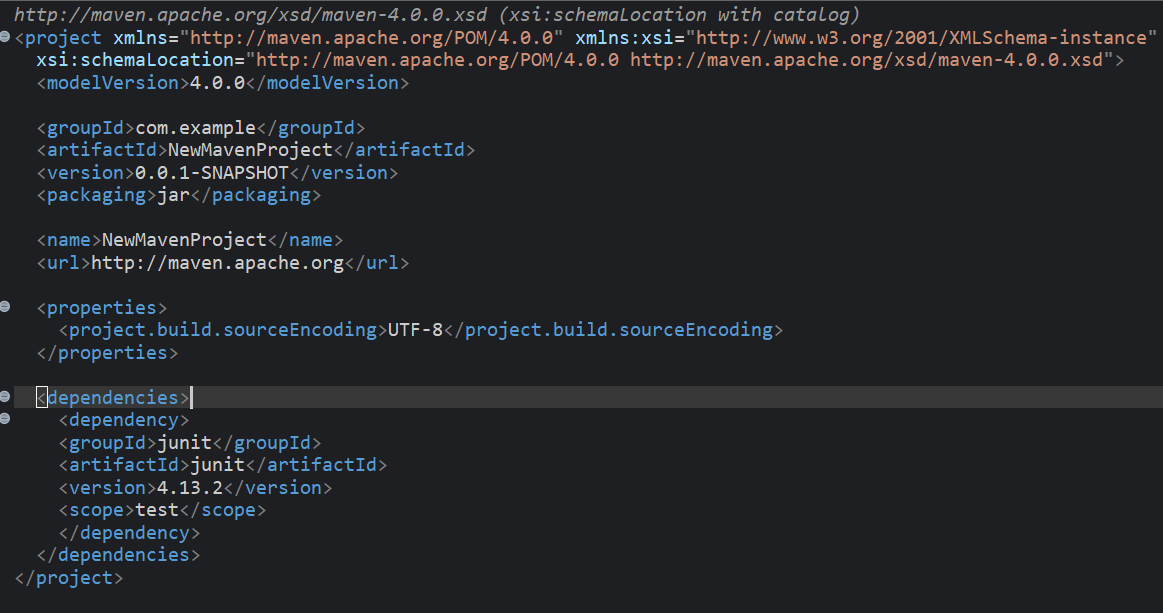
<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>



3. Create a new test class in your project.



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.

package com.example.NewMavenProject;

import static org.junit.Assert.\*;

import org.junit.Test;

public class NewJavaTestClass {

*@Test*

public void testAssert()

{

//assert equals

*assertEquals*(5,2+3);

//assert true

*assertTrue*(5>3);

//assert false

*assertFalse*(5<3);

//assert null

*assertNull*(null);

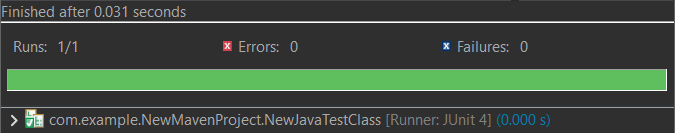
//assert not null

*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.

**Class Calculator**

package com.example.NewMavenProject;

public class Calculator {

public int add(int a,int b)

{

return a+b;

}

public int multiply(int a,int b)

{

return a\*b;

}

}

**Class CalculatorTest**

package com.example.NewMavenProject;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class CalculatorTest {

Calculator c;

*@Before*

public void setUp()

{

//arrange

c = new Calculator();

System.*out*.println("Setting up...");

}

*@After*

public void tearDownMethod()

{

System.*out*.println("Cleaning up...");

}

*@Test*

public void testAdd() {

//act

int result=c.add(10, 20);

System.*out*.println("add:"+result);

//assert

*assertEquals*(30,result);

}

*@Test*

public void testMultiply() {

int result=c.multiply(4, 20);

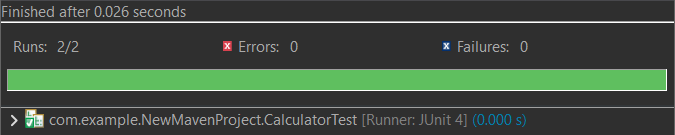
System.*out*.println("multiply:"+result);

*assertEquals*(80,result);

}

}

**Output**

****

**MOCKITO EXERCISE**

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.

**Class ExternalApi**

package com.example.Mockito;

public interface ExternalApi {

public String getData() ;

}

**Class MyService**

package com.example.Mockito;

public class MyService {

ExternalApi mockApi;

public MyService(ExternalApi mockApi) {

this.mockApi=mockApi;

}

public String fetchData() {

// **TODO** Auto-generated method stub

return mockApi.getData();

}

}

**Class MyServiceTest**

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");

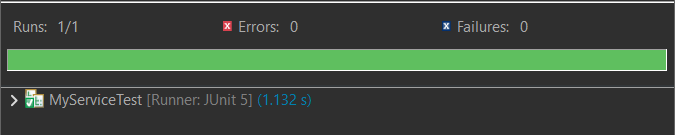
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



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.

**Class ExternalApi**

package com.example.Mockito;

public interface ExternalApi {

public String getData() ;

}

**Class MyService**

package com.example.Mockito;

public class MyService {

ExternalApi mockApi;

public MyService(ExternalApi mockApi) {

this.mockApi=mockApi;

}

public String fetchData() {

// **TODO** Auto-generated method stub

return mockApi.getData();

}

}

**Class MyServiceTest**

package com.example.Mockito;

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

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);

*when*(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

service.fetchData();

*verify*(mockApi).getData();

}

}

