**Week - 2**

**TDD using JUnit and Mockito**

**Exercise 1: Setting Up JUnit**

**Project Name** : Setting Up JUnit

**Exercise**  : JUnit Testing with Maven

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Introduction:**

This project demonstrates how to set up and run JUnit tests in a Maven-based Java project using Eclipse. The main objective is to test basic arithmetic operations using JUnit 4.

* **Tools and Technologies Used:**

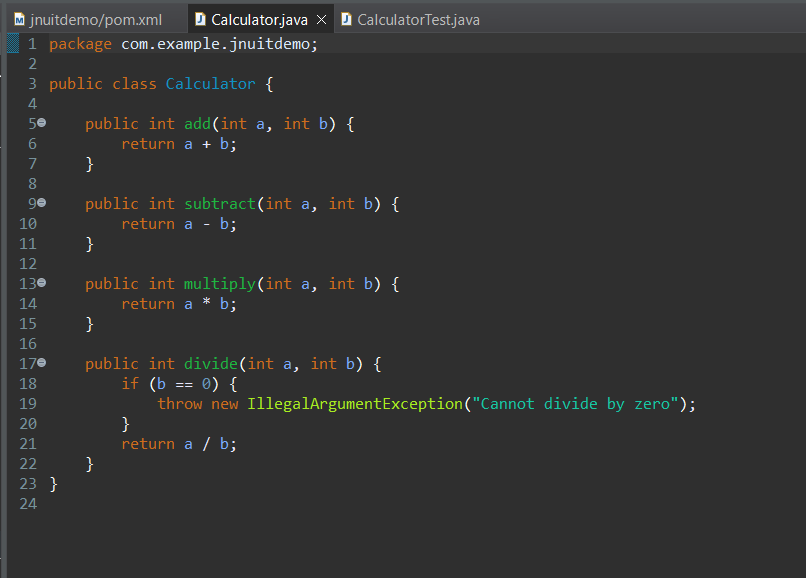
| **Tool/Tech** | **Version / Notes** |
| --- | --- |
| Java | Java 8 or 11 (whichever you used) |
| Eclipse IDE | Version info (e.g., 2023-12) |
| Maven | Confirm it's installed (e.g., 3.8.x) |
| JUnit | 4.13.2 |
| OS | Windows 10 / 11 |

* **Project Setup and Steps:**

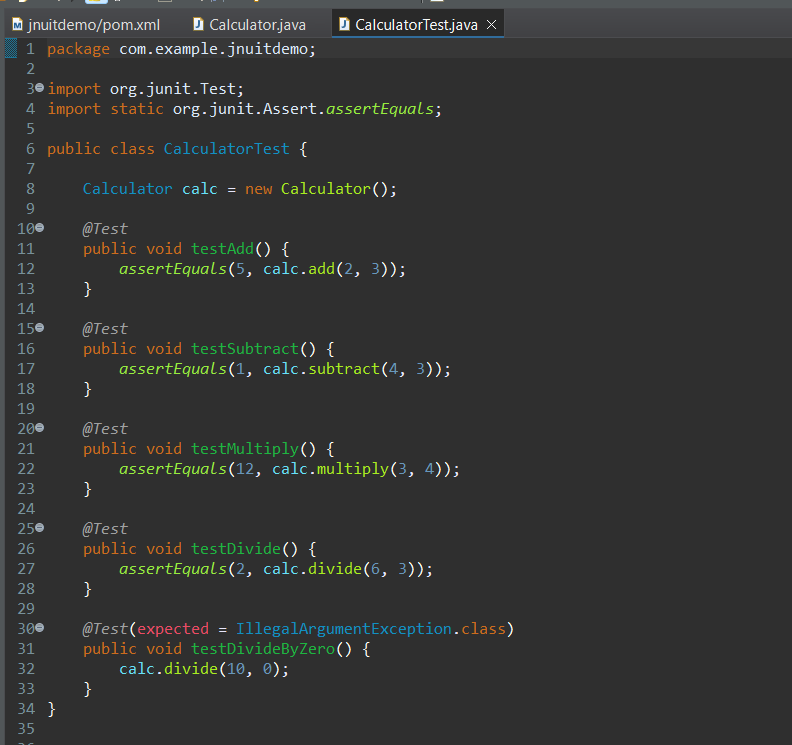
1. Created a Maven project in Eclipse
2. Selected archetype: maven-archetype-quickstart
3. Updated pom.xml to include JUnit 4.13.2 dependency
4. Created a Calculator.java class with basic math functions
5. Created a CalculatorTest.java file to test each method
6. Used @Test annotations to run assertions
7. Ran the test using Run As → JUnit Test

* **Source Code:**

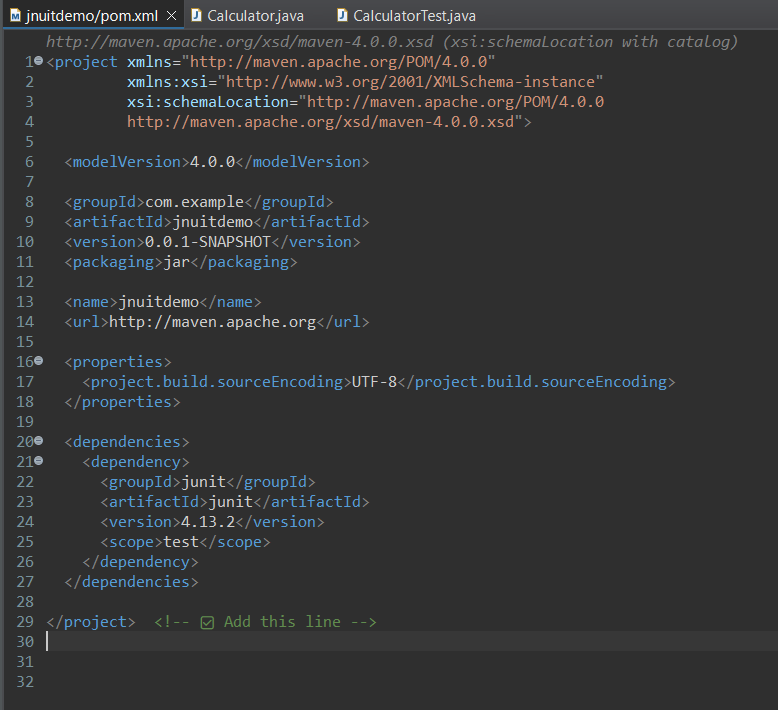
1. **Calculator.java**



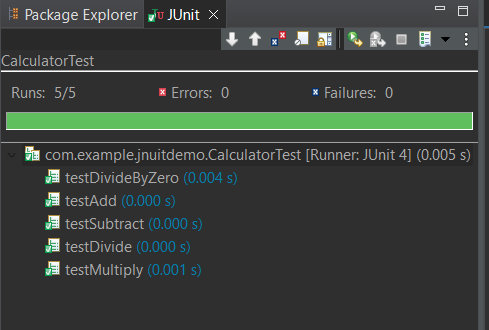
1. **CalculatorTest.java**



1. **Pom.xml**



* **Output:**



**Exercise 2 : Assertions in JUnit**

**Project Name** : Assertions in JUnit

**Exercise**  : JUnit Assertions Testing with Maven

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Objective:**

The goal of this exercise is to demonstrate the use of various assertion methods in JUnit, such as assertEquals, assertTrue, assertFalse, assertNull, and assertNotNull. These assertions help validate the expected outcomes in unit tests.

* **Tools and Technologies Used:**

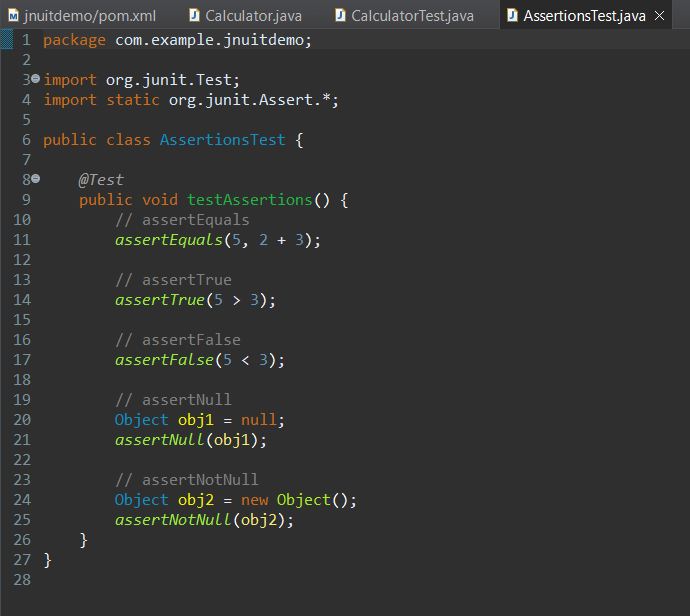
| **Tool/Technology** | **Details** |
| --- | --- |
| Java | Java 8 / Java 11 (whichever you used) |
| Eclipse IDE | Version (e.g., 2023-12) |
| JUnit | 4.13.2 |
| Maven | Present in the project |
| OS | Windows 10 / 11 |

* **Implementation:**

This is the continuation of the above exercise(Setting up JUnit)

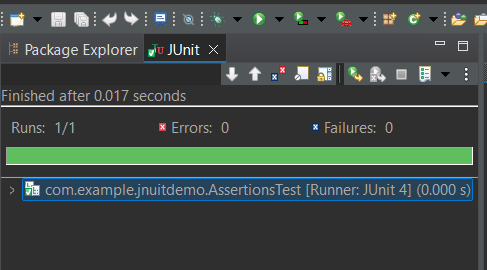
**Step-1:** Create a New Test Class **“AssertionsTest.java”**

Location: src/test/java/com/example/jnuitdemo/AssertionsTest.java



**Step 2:** Run the AssertionsTest.java file using JUnit

* **Output:**



* **Explanations of Assertions:**

| **Assertion** | **Purpose** |
| --- | --- |
| assertEquals(expected, actual) | Checks if two values are equal |
| assertTrue(condition) | Checks if a condition is true |
| assertFalse(condition) | Checks if a condition is false |
| assertNull(object) | Passes if the object is null |
| assertNotNull(object) | Passes if the object is **not null** |

**Exercise 3: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup andTeardown Methods in JUnit**

**Project Name** : AAA in JUnit

**Exercise**  : Arrange-Act-Assert (AAA) Pattern, Setup and Teardown in JUnit

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Objective:**

To organize JUnit test cases using the Arrange-Act-Assert (AAA) pattern and apply @Before and @After annotations for setup and teardown methods.

* **Tools and Technologies Used:**

| **Tool/Technology** | **Details** |
| --- | --- |
| Java | Java 8 / Java 11 (whichever you used) |
| Eclipse IDE | Version (e.g., 2023-12) |
| JUnit | 4.13.2 |
| Maven | Present in the project |
| OS | Windows 10 / 11 |

* **Steps followed:**

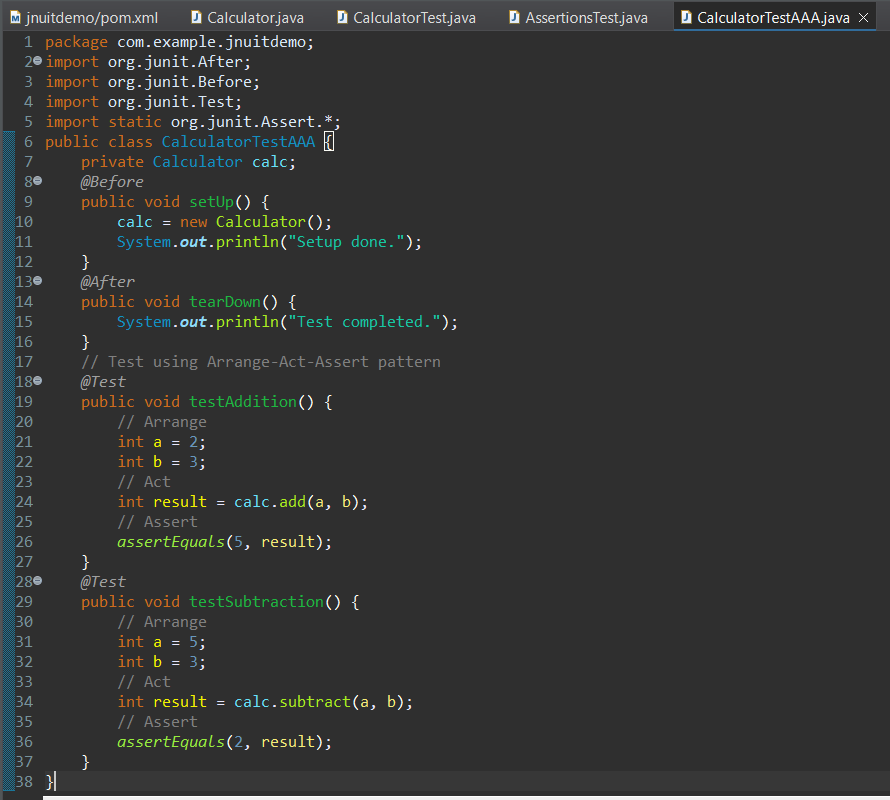
1. Created a Maven project in Eclipse.
2. Added JUnit dependency in pom.xml.
3. Created a Calculator class with methods.
4. Created CalculatorTestAAA class in src/test/java.
5. Used @Before to set up objects.
6. Used @After for cleanup.
7. Wrote test cases using AAA pattern.
8. Ran the test using JUnit Test option.

* **Source Code:**

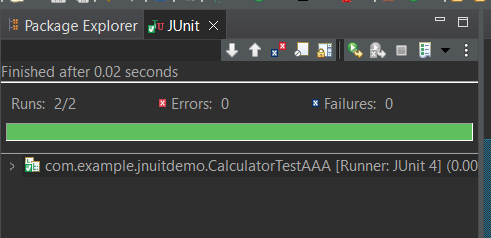
1. **Calculator.java**



1. **CalculatorTestAAA.java**



* **Outputs:**



**Exercise 4: Mocking and Stubbing**

**Project Name** : Mocking and Stubbing

**Exercise**  : Mocking and Stubbing using JUnit & Mockito

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Introduction:**

This exercise demonstrates how to test a class that depends on an external API using Mockito for mocking and JUnit for assertions. The goal is to isolate the class under test from its dependencies.

* **Steps Followed:**

1. Created a Maven project in Eclipse named `mockingdemo`.
2. Added dependencies for JUnit 5 and Mockito in the `pom.xml`.
3. Created an interface `ExternalApi` with a method `getData()`.
4. Created a class `MyService` which uses `ExternalApi`.
5. Created a test class `MyServiceTest`:

- Used Mockito to create a mock of `ExternalApi`

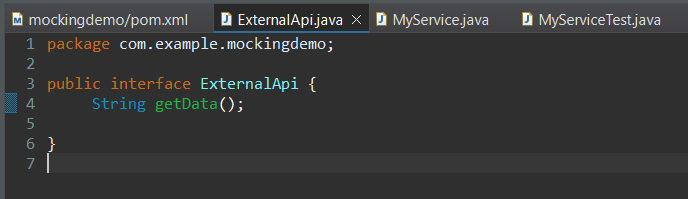
- Stubbed the `getData()` method to return "Mock Data"

-Called the method via `MyService` and validated the result using `assertEquals()`.

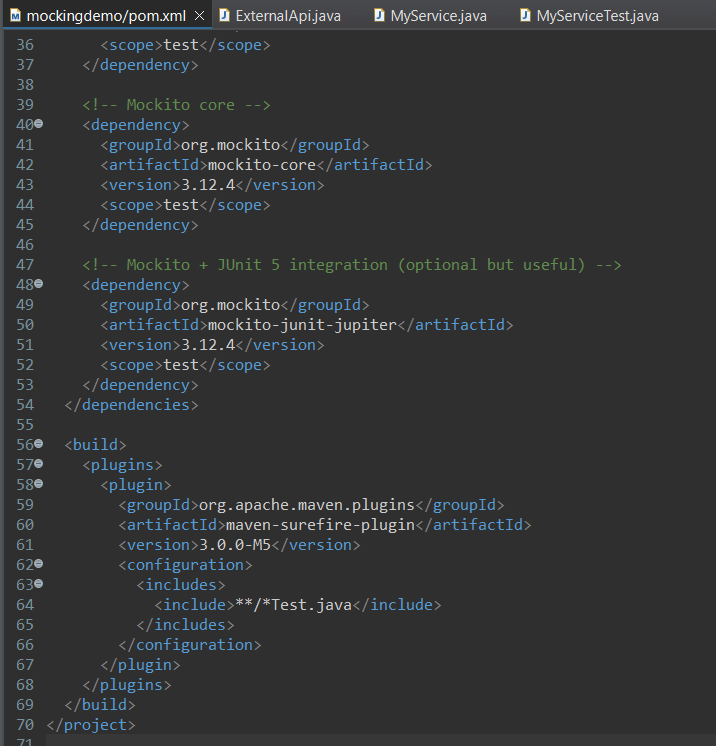
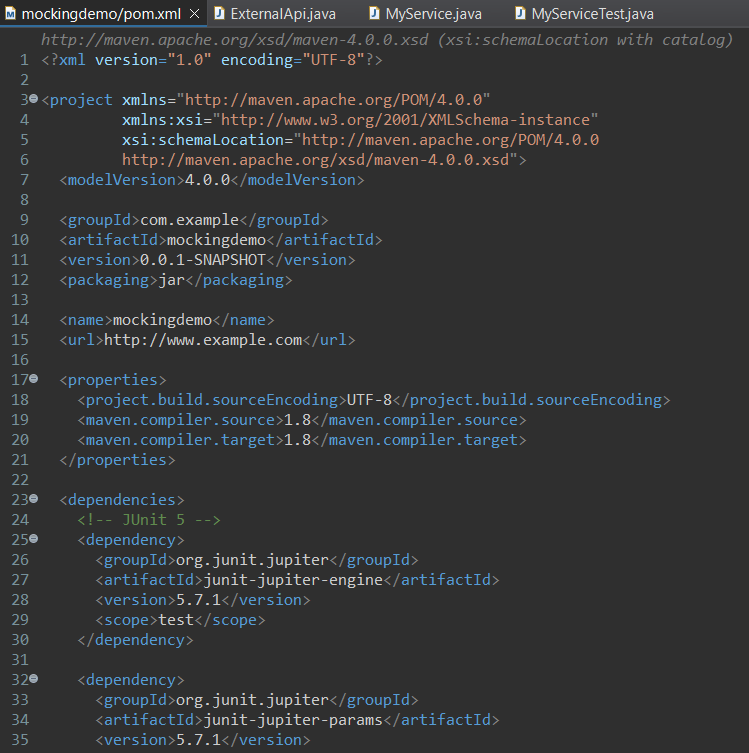
1. Ran the test successfully in Eclipse.

* **Source Code:**

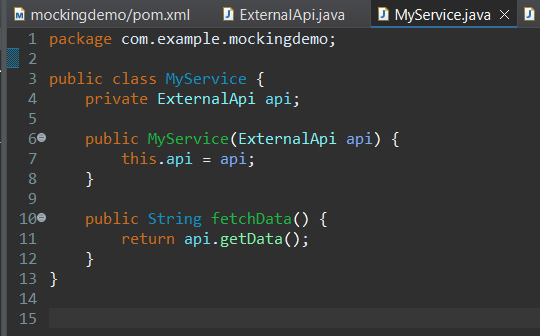
1. **ExternalApi.java(Interface)**



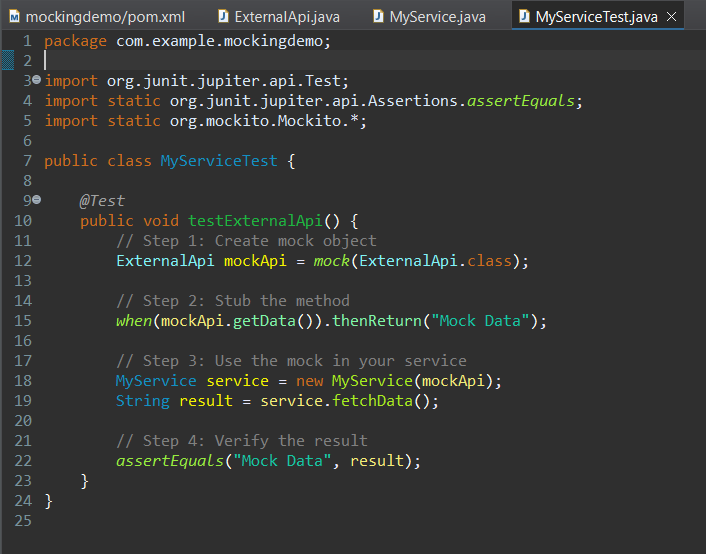
1. **Pom.xml**



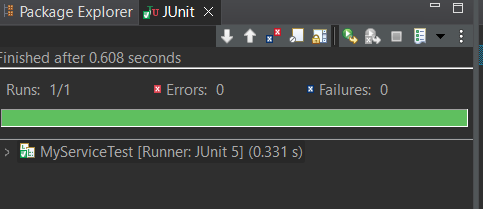
1. **MyService.java**



1. **MySeviceTest.java**



* **Output:**



**Exercise 5: Verifying Interactions**

**Project Name** : Verifying Interactions

**Exercise**  : Verifying Interactions using JUnit & Mockito

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Scenario:**

You need to ensure that a method is called with specific arguments. This helps confirm that the tested method correctly interacts with its dependencies.

* **Objective:**

To verify whether a mock object method was called with expected parameters using Mockito and JUnit.

* **Steps Followed:**

**Step 1:** Create a mock object

A mock object of an interface ExternalApi is created using Mockito.

### Step 2: Call the method with specific arguments

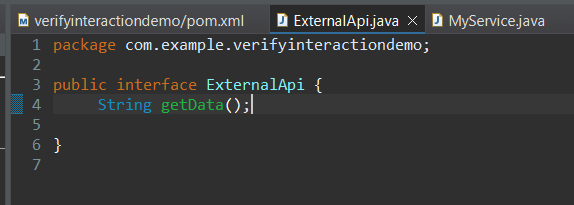
The service class (MyService) uses this mock and calls the getData() method.

### Step 3: Verify the interaction

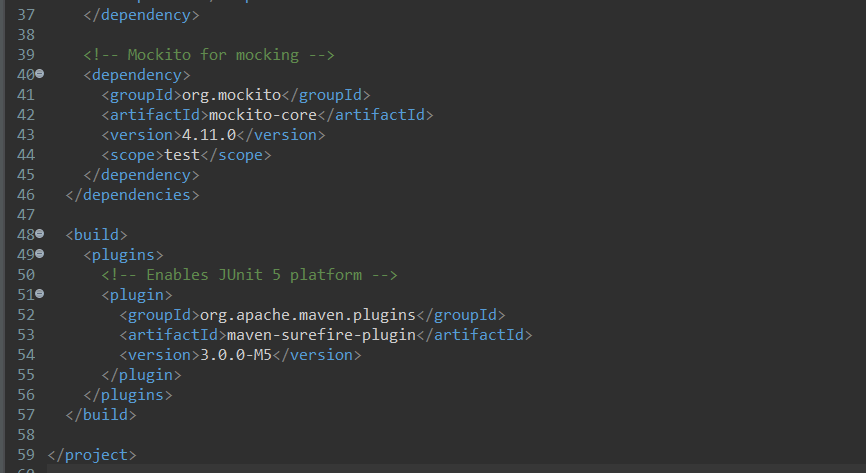
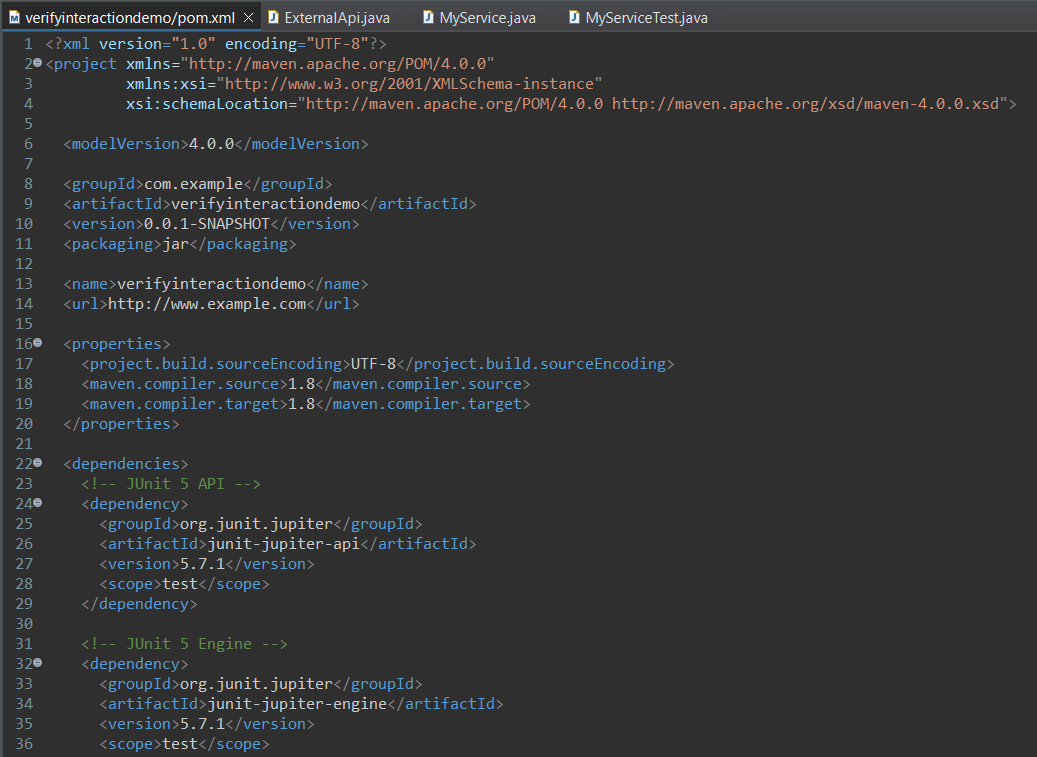
We then verify that getData() was indeed called on the mock.

* **Source code:**

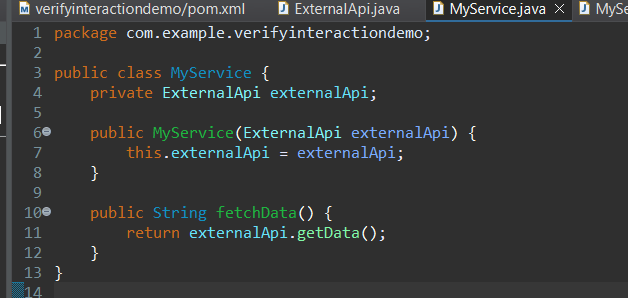
1. **ExternalApi.java**



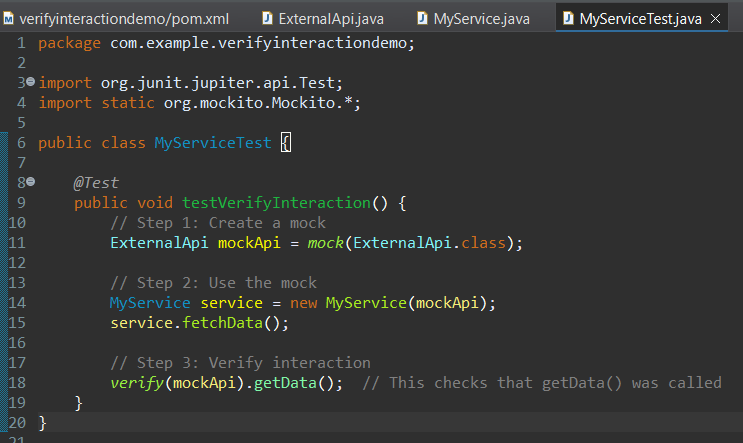
1. **Pom.xml**



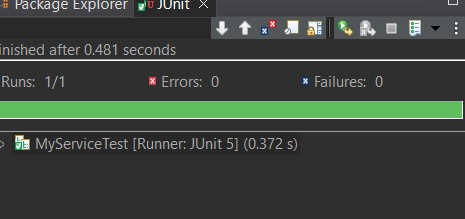
1. **MyService.java**



1. **MyServiceTest.java**



* **Outputs:**



**SLF4J logging framework**

**Exercise 6: Logging Error Messages and Warning Levels**

**Project Name** : Logging Error Messages and Warning Levels

**Exercise**  : Logging Error Messages and Warning Levels using SLF4J logging framework

**Name**  : Kotha Akanksha

**Superset Id** : 6373762

* **Objective:**

To write a Java application that demonstrates logging error messages and warning levels using **SLF4J** and **Logback**.

* **Scenario:**

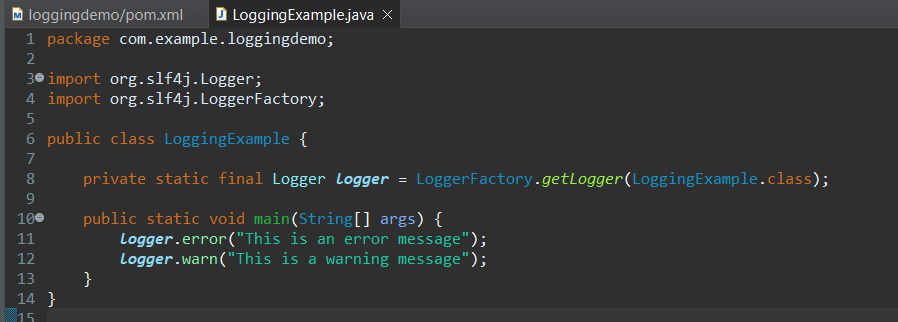
We need to use SLF4J for logging and Logback as the underlying logging implementation. The program should log an error message and a warning message using appropriate log levels.

* **Steps Followed:**

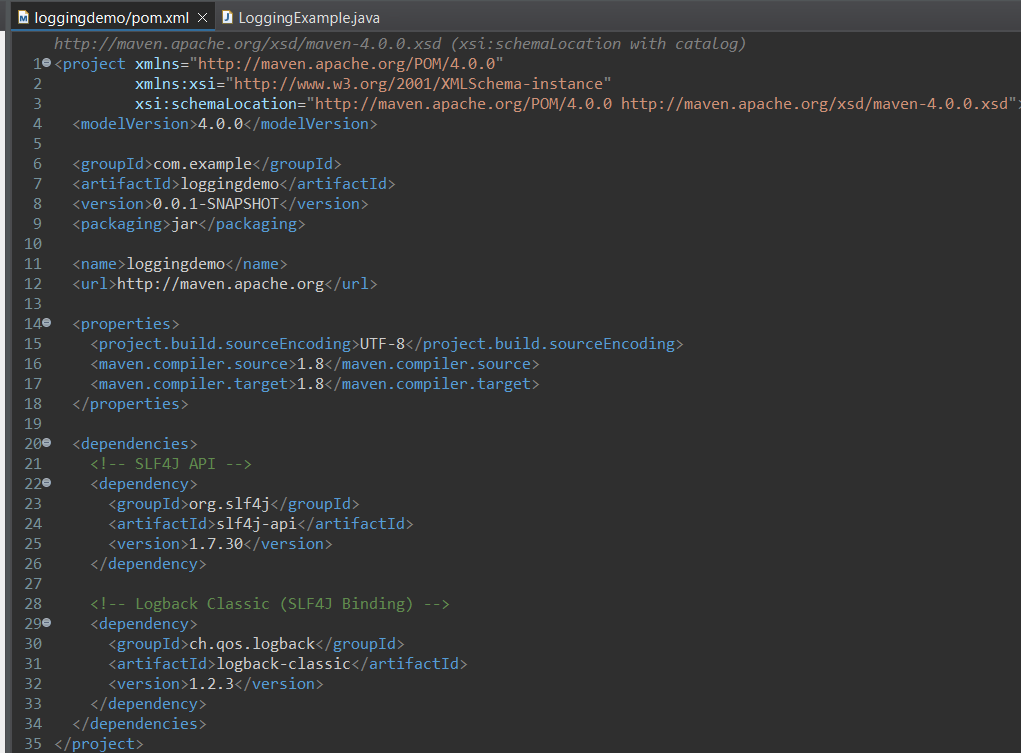
1. Created a Maven Project in Eclipse named ‘loggingdemo’
2. Added Dependencies in pom.xml
3. Created a class ‘LoggingExample.java’
4. Ran the application successfully.

* **Source code:**

1. **LoggingExample.java**



1. **Pom.xml**



* **Output:**

