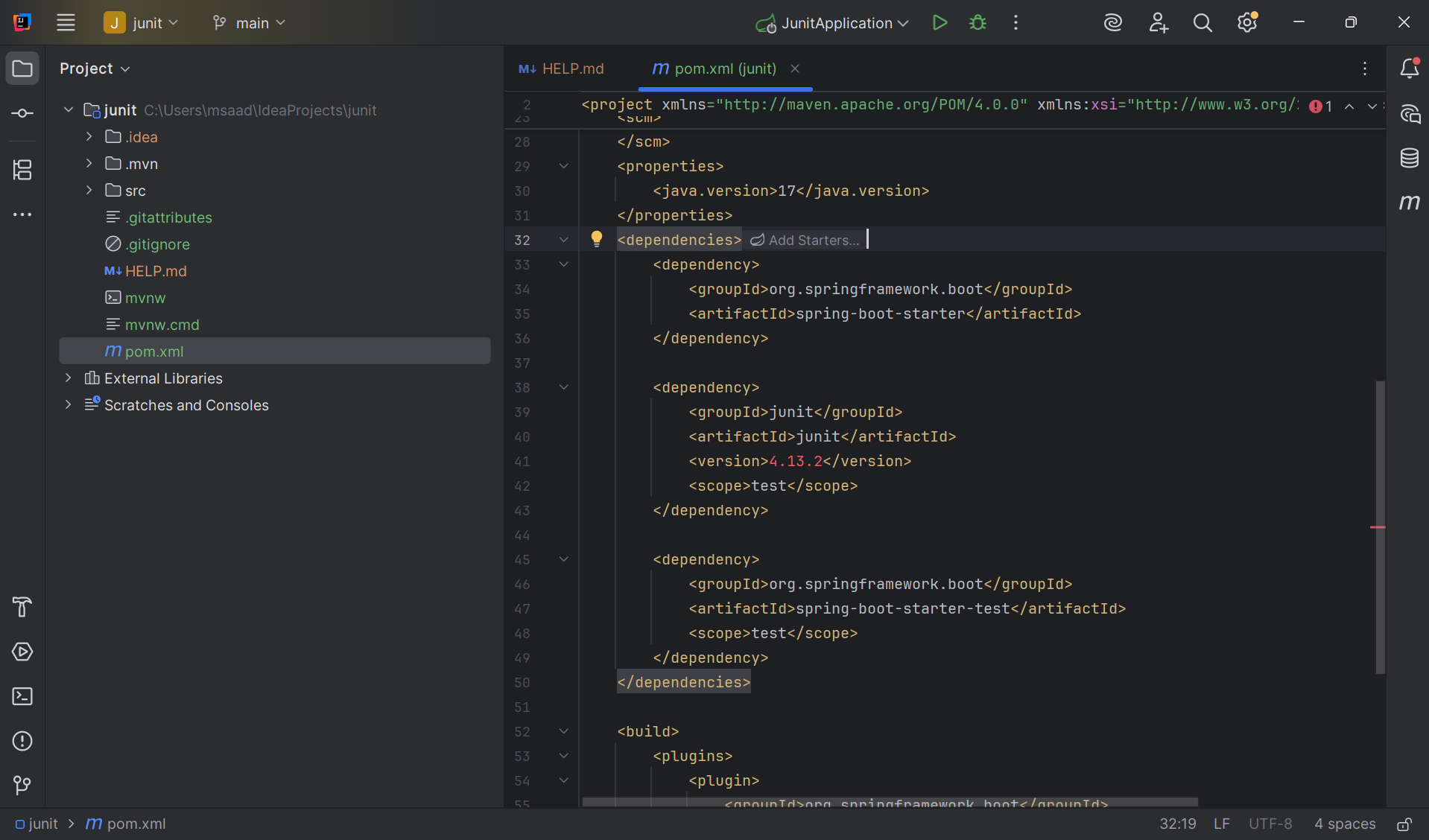
**JUnit Testing Exercises**

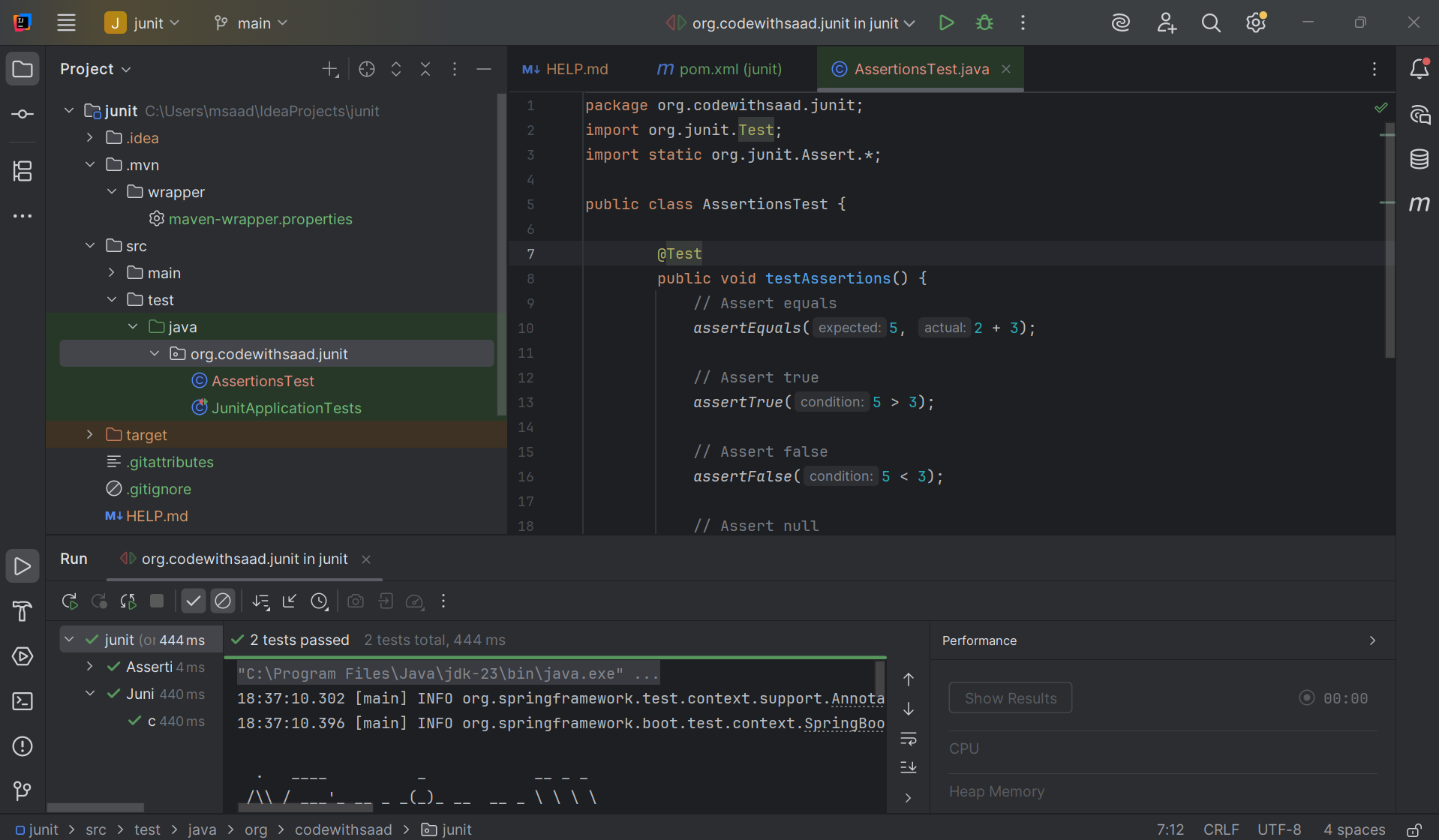
**Exercise 1: Setting Up Junit**

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

****

**Exercise 3: Assertions in JUnit**

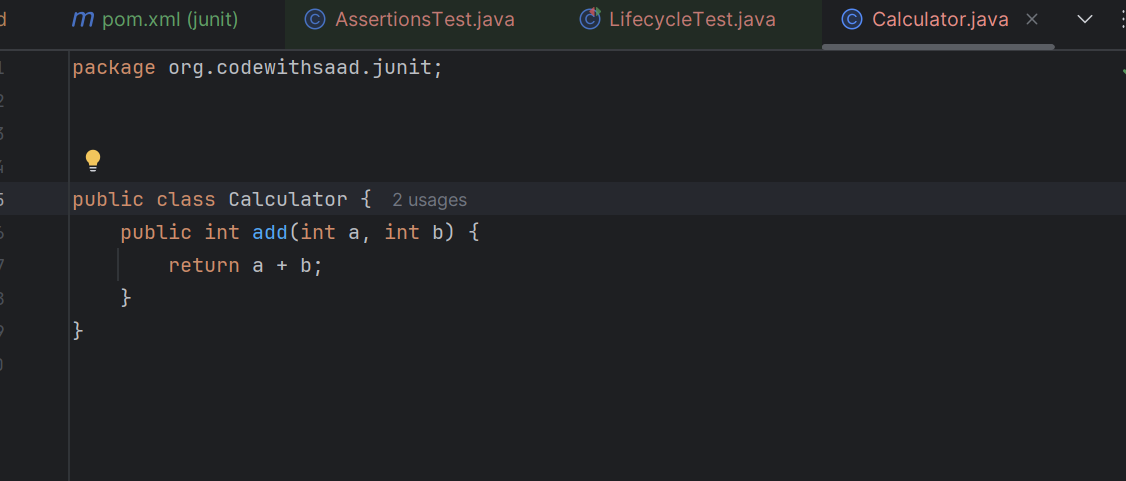
**Scenario:** You need to use different assertions in JUnit to validate your test results.

****

**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.

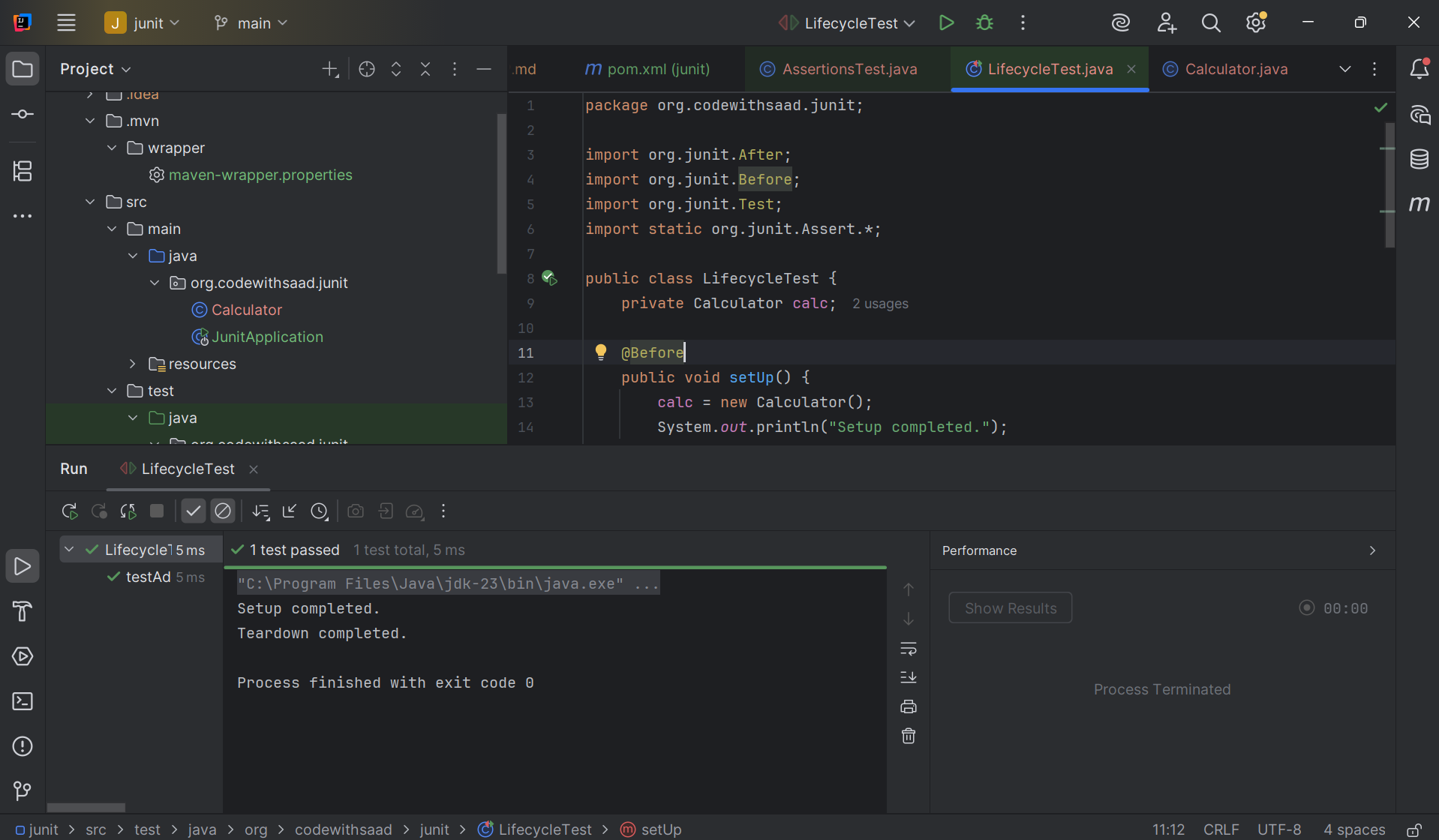
**Calculator.java**

****

**LifecycleTest.java**

package org.codewithsaad.junit;  
  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class LifecycleTest {  
 private Calculator calc;  
  
 @Before  
 public void setUp() {  
 calc = new Calculator();  
 System.*out*.println("Setup completed.");  
 }  
  
 @Test  
 public void testAdd() {  
 // Arrange done in setup  
 // Act  
 int result = calc.add(2, 3);  
 // Assert  
 *assertEquals*(5, result);  
 }  
  
 @After  
 public void tearDown() {  
 System.*out*.println("Teardown completed.");  
 }  
}

**Output:**

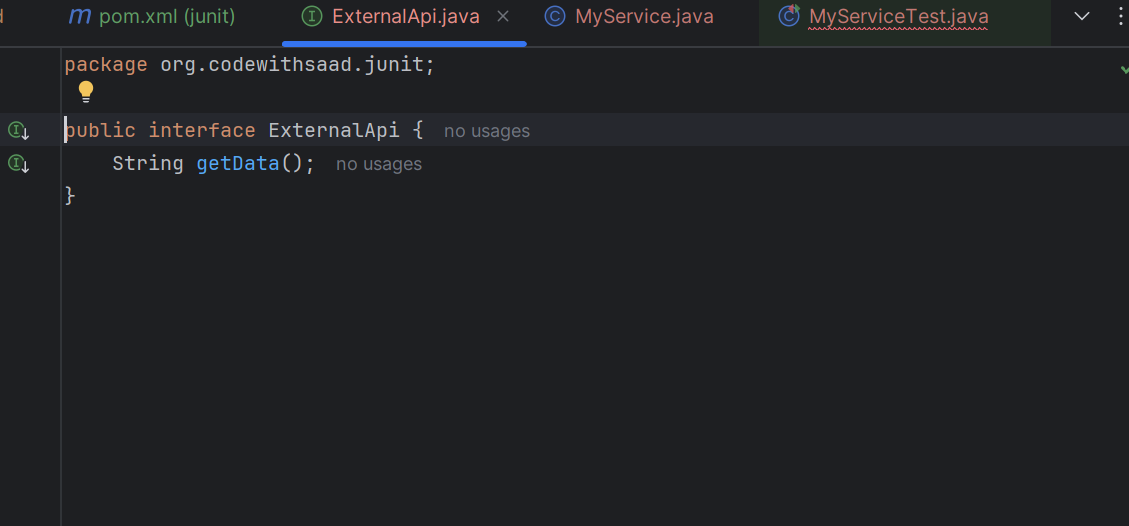


**Mockito Hands-On Exercises**

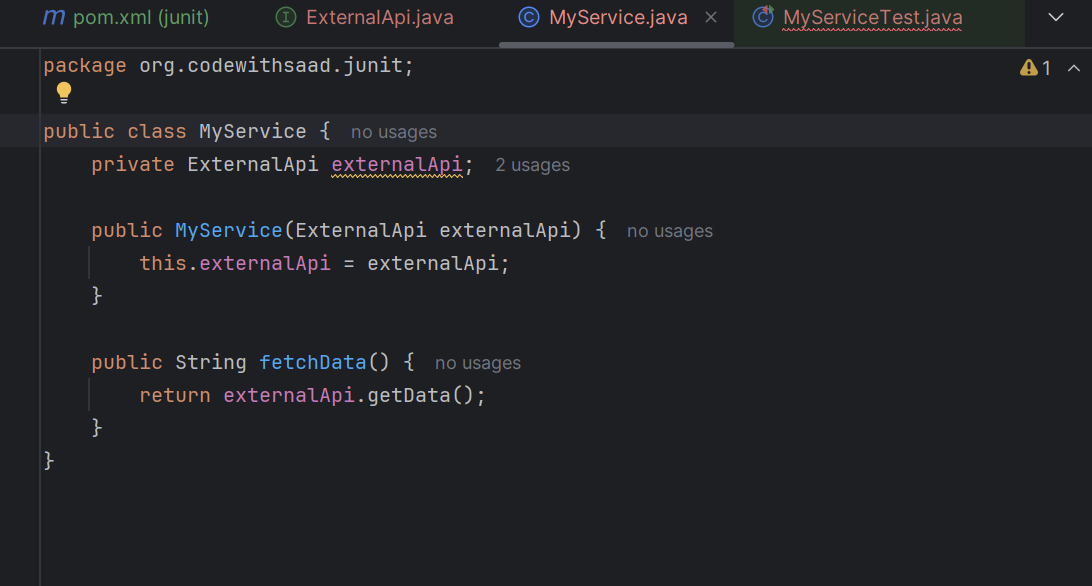
**Exercise 1: Mocking and Stubbing**

**Scenario:** You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

**ExternalApi.java**



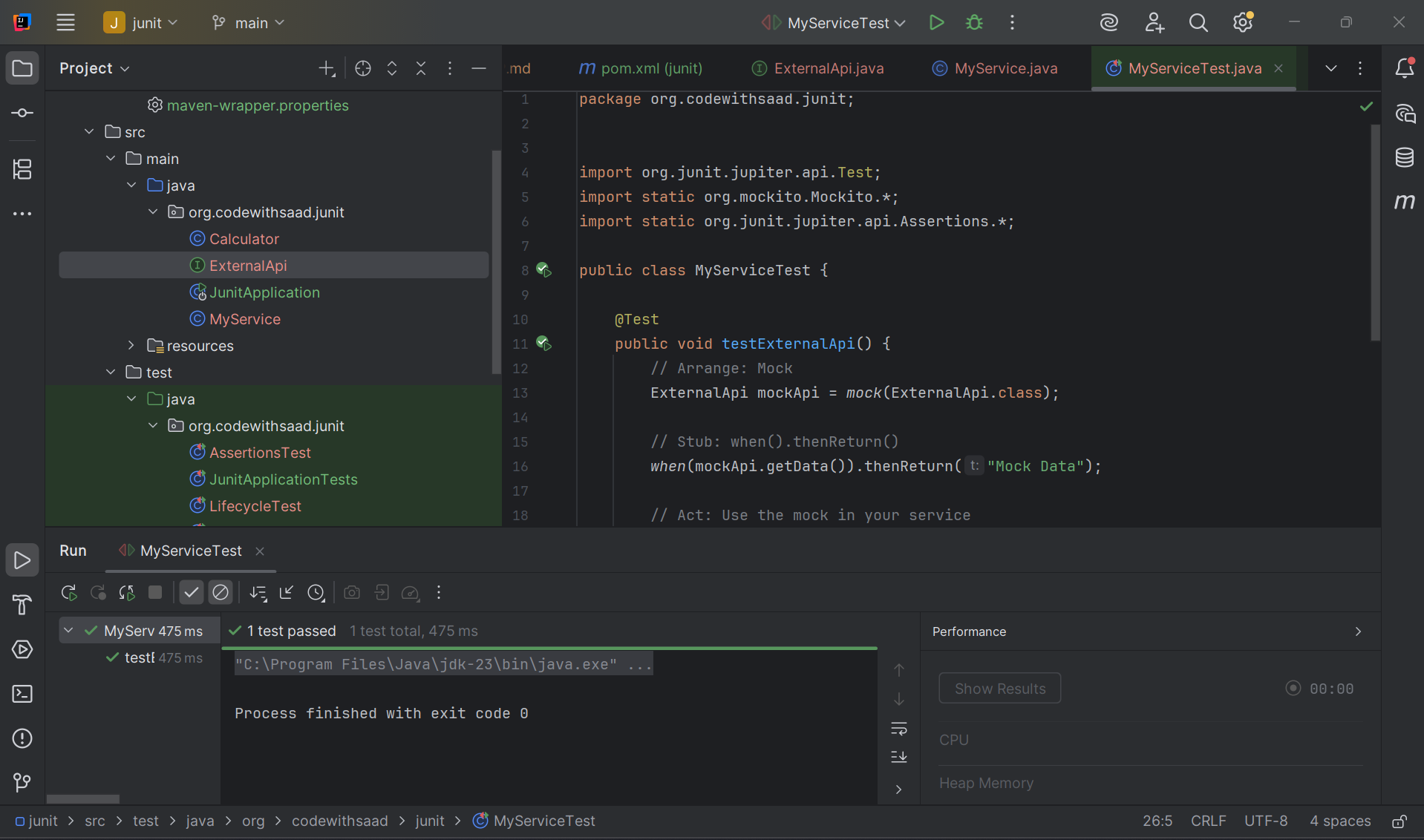
**MyServiceTest.java**



**MyServiceTest.java (testing code)**

package org.codewithsaad.junit;  
  
package org.codewithsaad.mockito;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
 // Arrange: Mock  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 // Stub: when().thenReturn()  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
 // Act: Use the mock in your service  
 MyService service = new MyService(mockApi);  
 String result = service.fetchData();  
  
 // Assert  
 *assertEquals*("Mock Data", result);  
 }  
}

**Output:**



**Exercise 2: Verifying Interactions**

**Scenario:** You need to ensure that a method is called with specific arguments.

**In the above testing code we add another piece of code,**

@Test

public void testVerifyInteraction() {

// Arrange: Mock

ExternalApi mockApi = mock(ExternalApi.class);

// Act: Use the mock

MyService service = new MyService(mockApi);

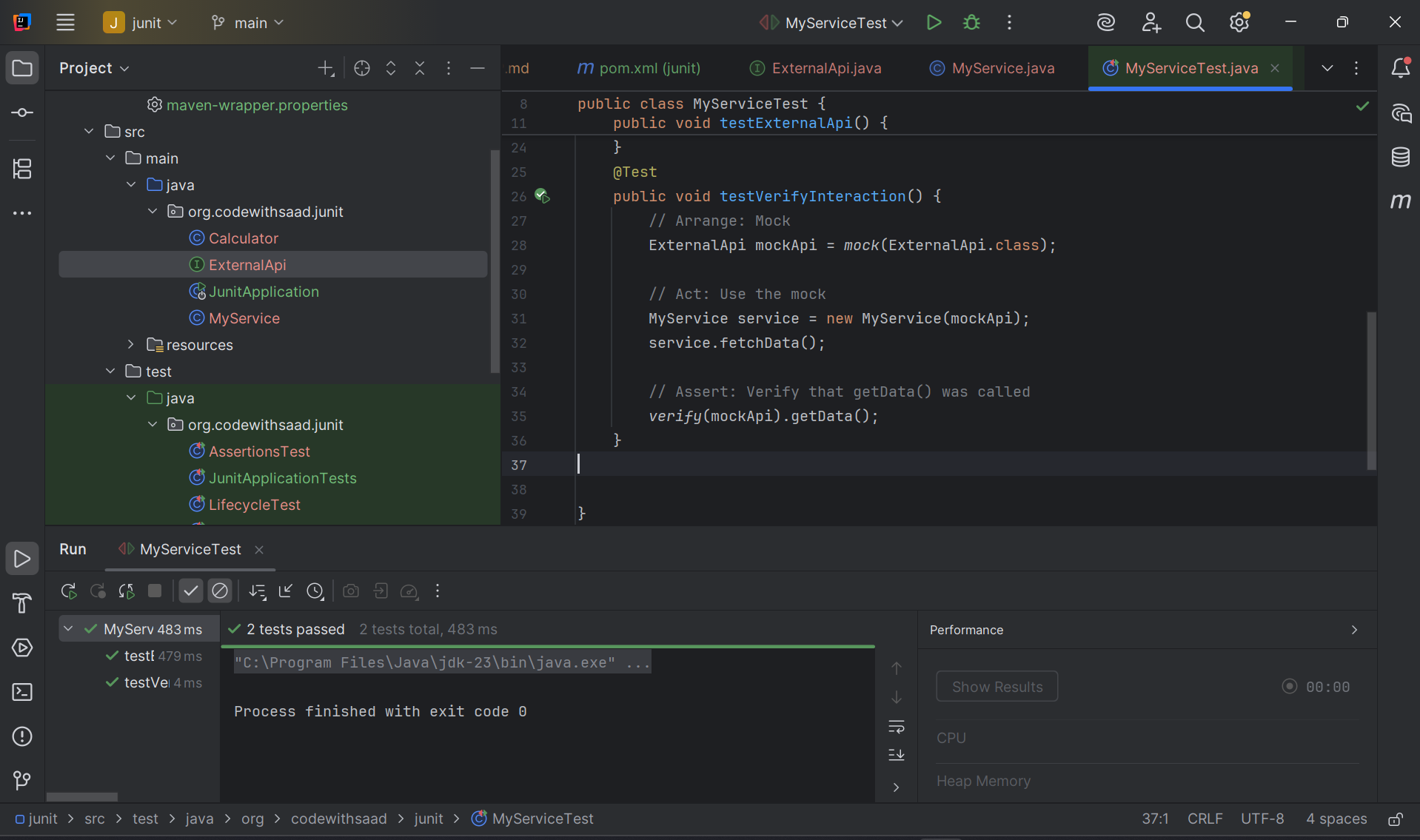
service.fetchData();

// Assert: Verify that getData() was called

verify(mockApi).getData();

}

**Output:**



**Logging using SLF4J**

**Exercise 1: Logging Error Messages and Warning Levels**

**Task:** Write a Java application that demonstrates logging error messages and warning levels

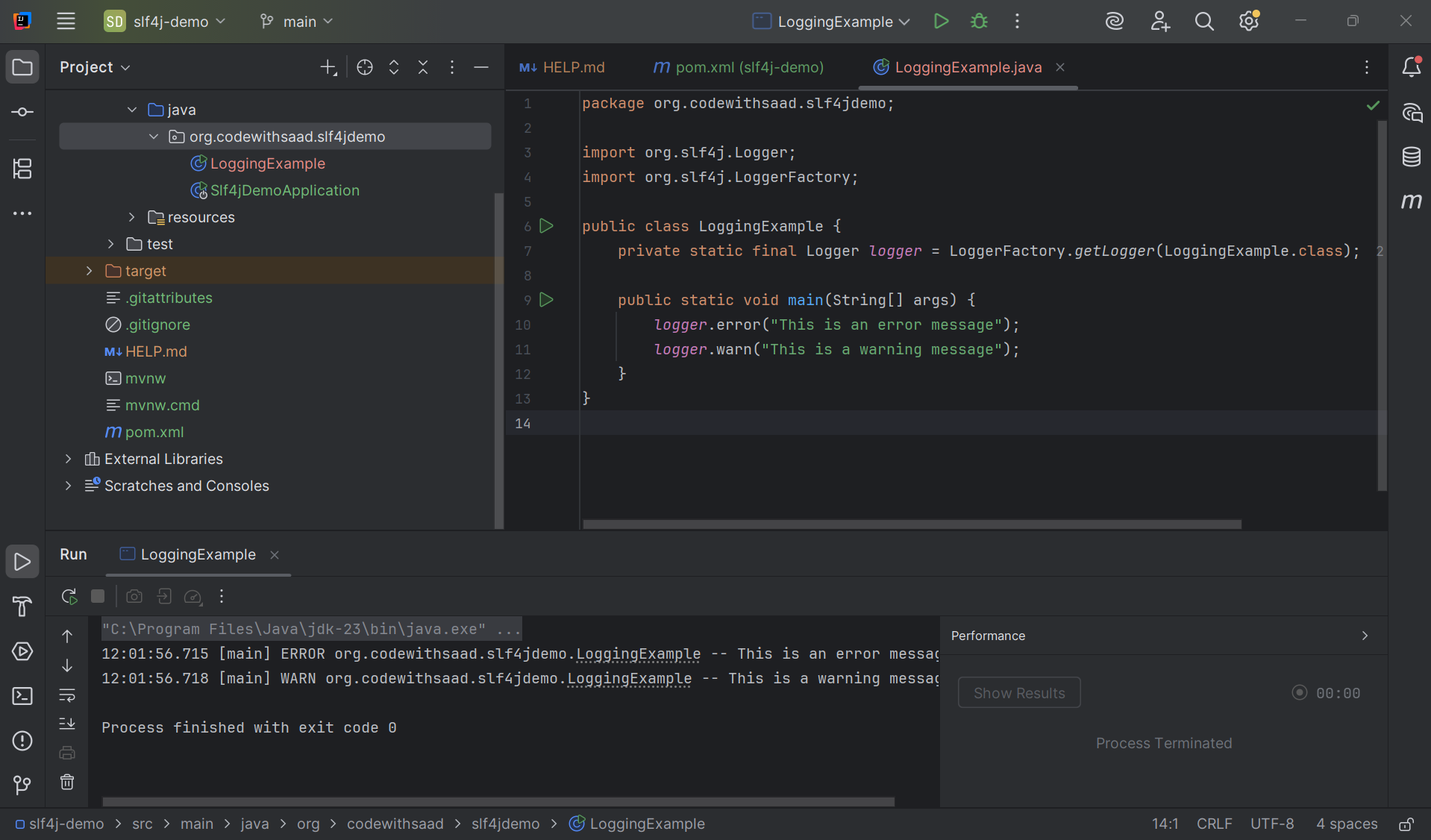
using SLF4J.

**I am not adding the dependencies given in the handbook as the dependencies are already integrated in my IntelliJ IDE**

**LoggingExample.java**

****

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ThankYou\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**