**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).

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

junit junit 4.13.2 test

3. Create a new test class in your project.

CODE:

**SampleTest.java:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class SampleTests {  
  
 @Test  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  
 *assertEquals*(5, result);  
 }  
}

**Calculator.java:**

package com.Spring2.mySpringProject2;  
  
public class Calculator {

public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int subtract(int a, int b) {  
 return a - b;  
 }  
  
 public static class ExternalApi {  
 }  
  
 public static class MyService {  
 }  
}

**Pom.xml:**

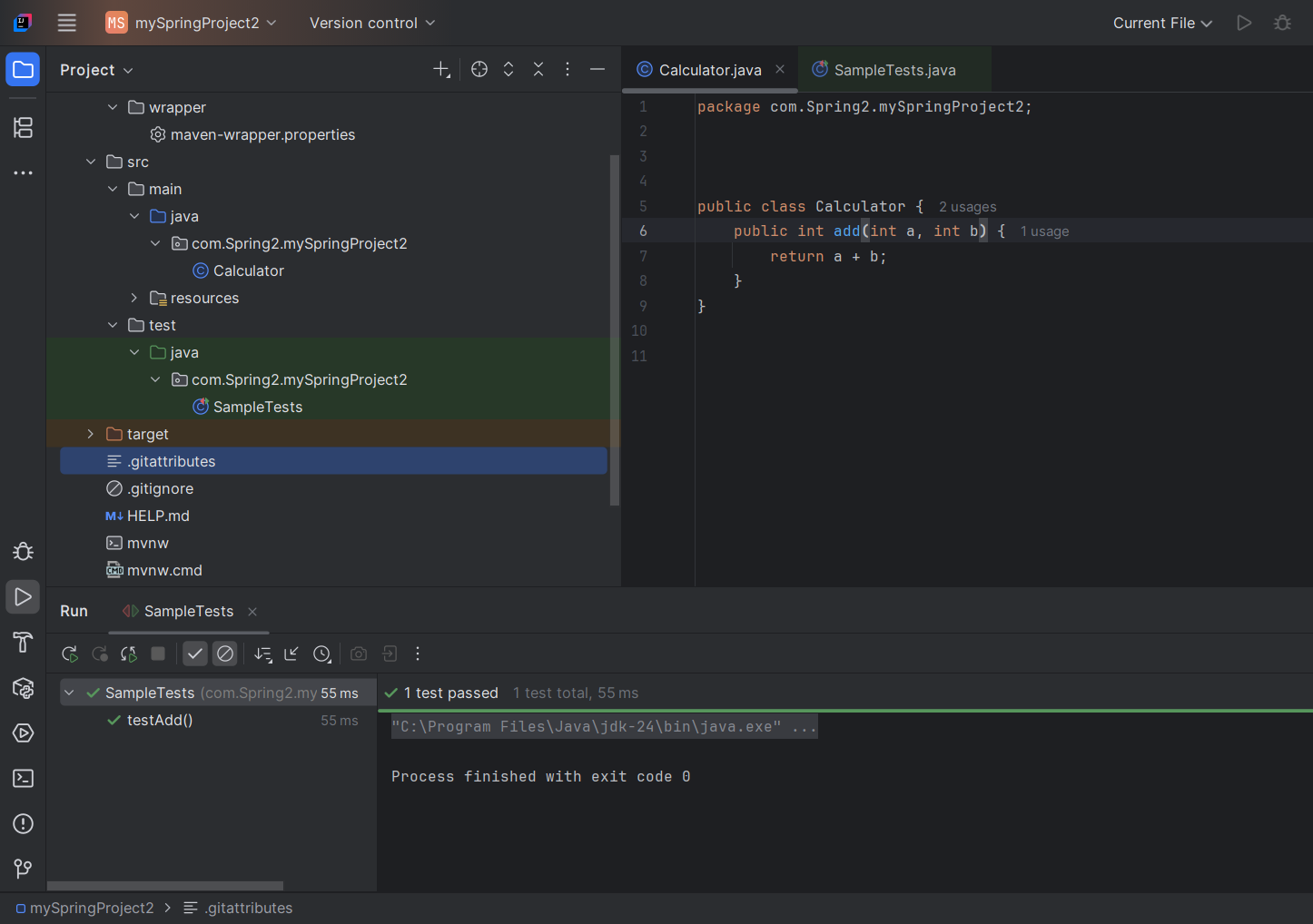
<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>



**Exercise 2:** **Writing Basic JUnit Tests Scenario:**

You need to write basic JUnit tests for a simple Java class.

Steps: 1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

**CODE:**

**MathUtils.java:**

package com.Spring2.mySpringProject2;  
  
public class MathUtils {  
 public int add(int a, int b) {  
 return a + b;  
 }  
  
 public int multiply(int a, int b) {  
 return a \* b;  
 }  
  
 public boolean isEven(int number) {  
 return number % 2 == 0;  
 }

}

**MathUtilsTest.java:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MathUtilsTest {  
  
 @Test  
 void testAdd() {  
 MathUtils math = new MathUtils();  
 *assertEquals*(7, math.add(3, 4));  
 }  
  
 @Test  
 void testMultiply() {  
 MathUtils math = new MathUtils();  
 *assertEquals*(12, math.multiply(3, 4));  
 }  
  
 @Test  
 void testIsEven() {  
 MathUtils math = new MathUtils();  
 *assertTrue*(math.isEven(4));  
 *assertFalse*(math.isEven(5));  
 }  
}

**Pom.xml:**

<dependency>

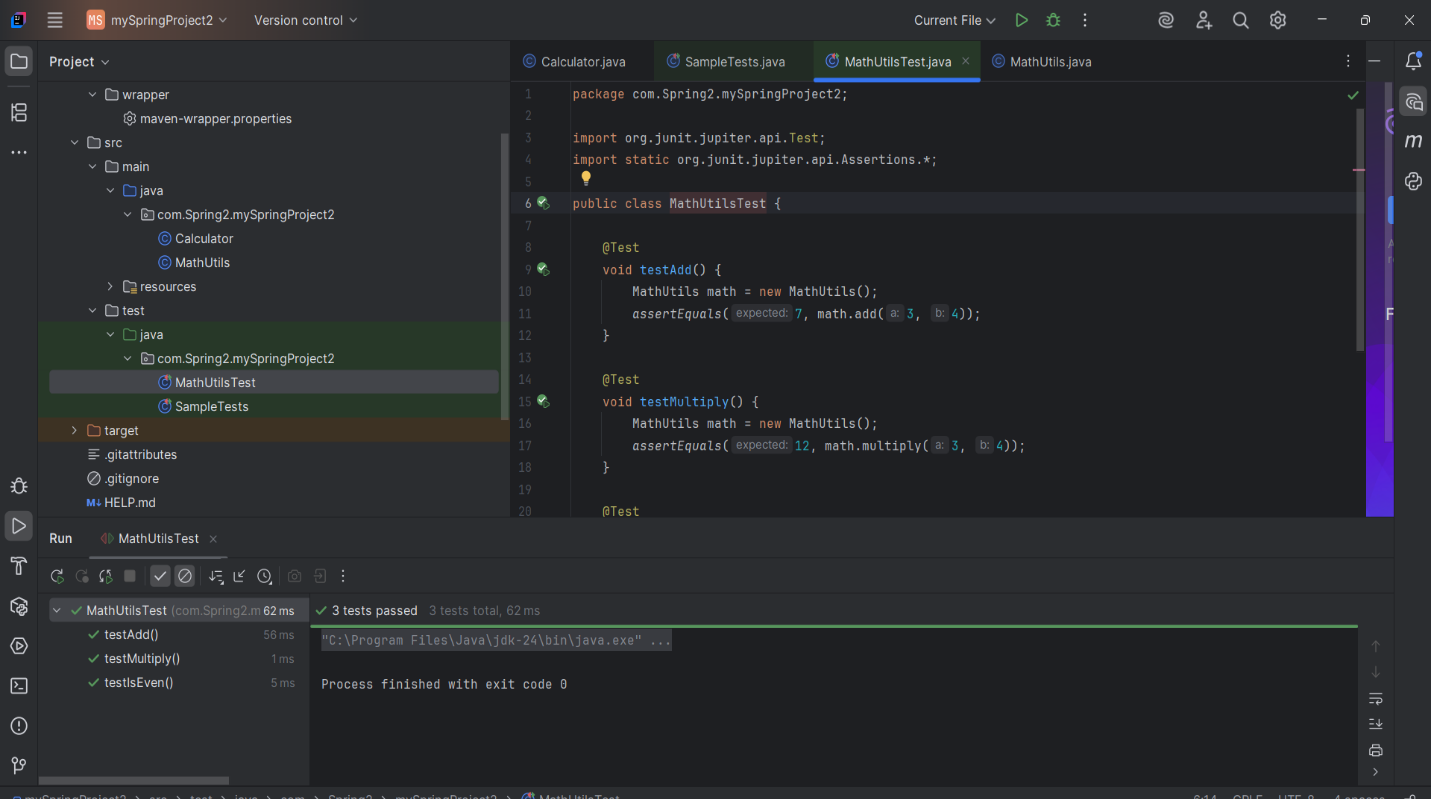
<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.12.2</version>

<scope>test</scope>

</dependency>



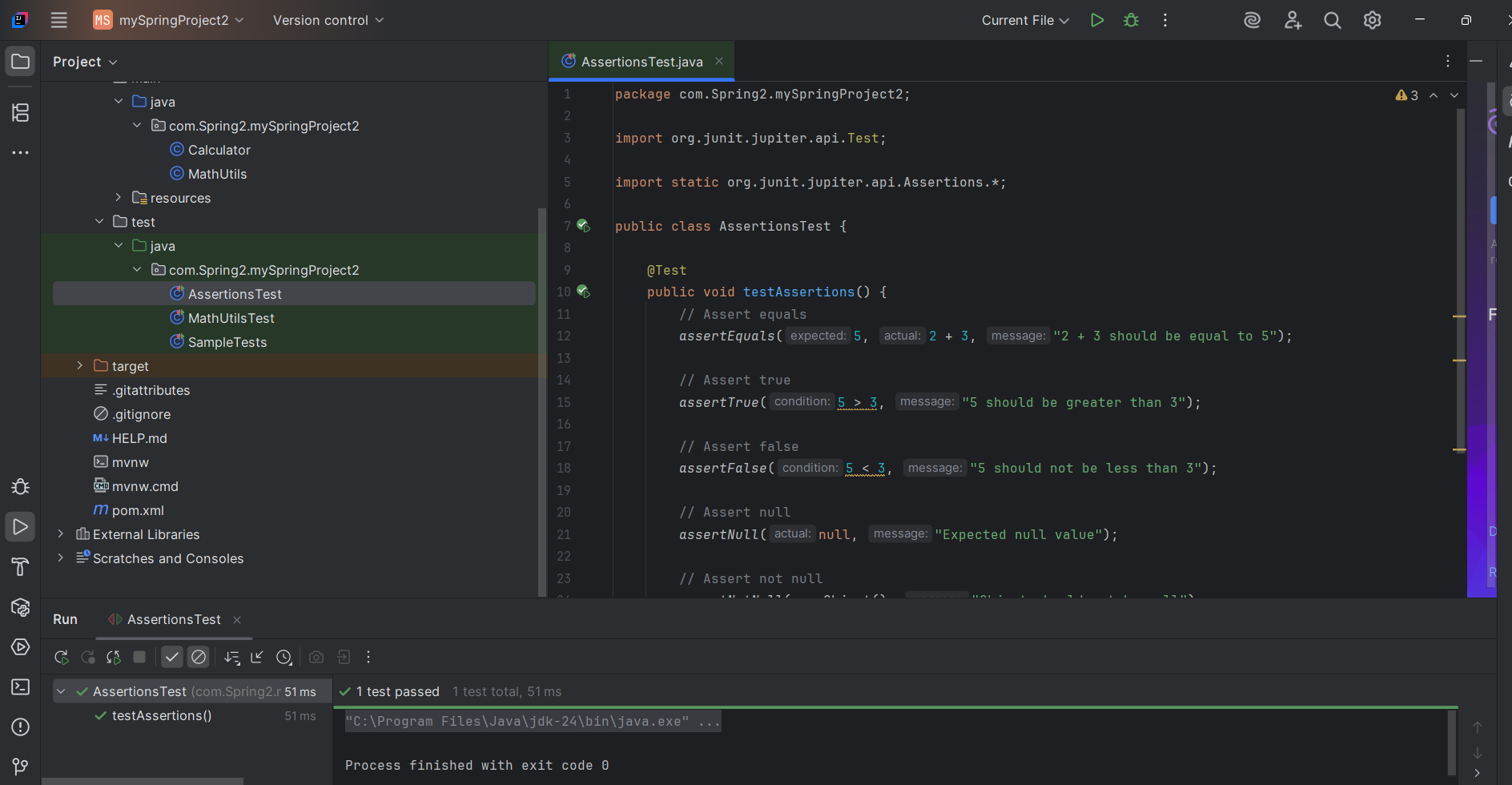
**Exercise 3: Assertions in JUnit Scenario:**

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

Steps: 1. Write tests using various JUnit assertions.

**CODE:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class AssertionsTest {  
  
 @Test  
 public void testAssertions() {  
 // Assert equals  
 *assertEquals*(5, 2 + 3, "2 + 3 should be equal to 5");  
  
 // Assert true  
 *assertTrue*(5 > 3, "5 should be greater than 3");  
  
 // Assert false  
 *assertFalse*(5 < 3, "5 should not be less than 3");  
  
 // Assert null  
 *assertNull*(null, "Expected null value");  
  
 // Assert not null  
 *assertNotNull*(new Object(), "Object should not be null");  
 }  
}

}

**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. Solution Code: import static org.mockito.Mockito.\*; import org.junit.jupiter.api.Test; import org.mockito.Mockito;

**CODE:**

**ExternalApi.java:**

package com.Spring2.mySpringProject2;  
  
public interface ExternalApi {  
 String getData();

}

**MyService.java:**

package com.Spring2.mySpringProject2;  
  
public class MyService {  
  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }

}

**MyServiceTest.java:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
  
 *when*(mockApi.getData()).thenReturn("Mock Data");  
  
  
 MyService service = new MyService(mockApi);  
  
  
 String result = service.fetchData();  
 *assertEquals*("Mock Data", result);  
 }

}

**Pom.xml:**

<dependency>

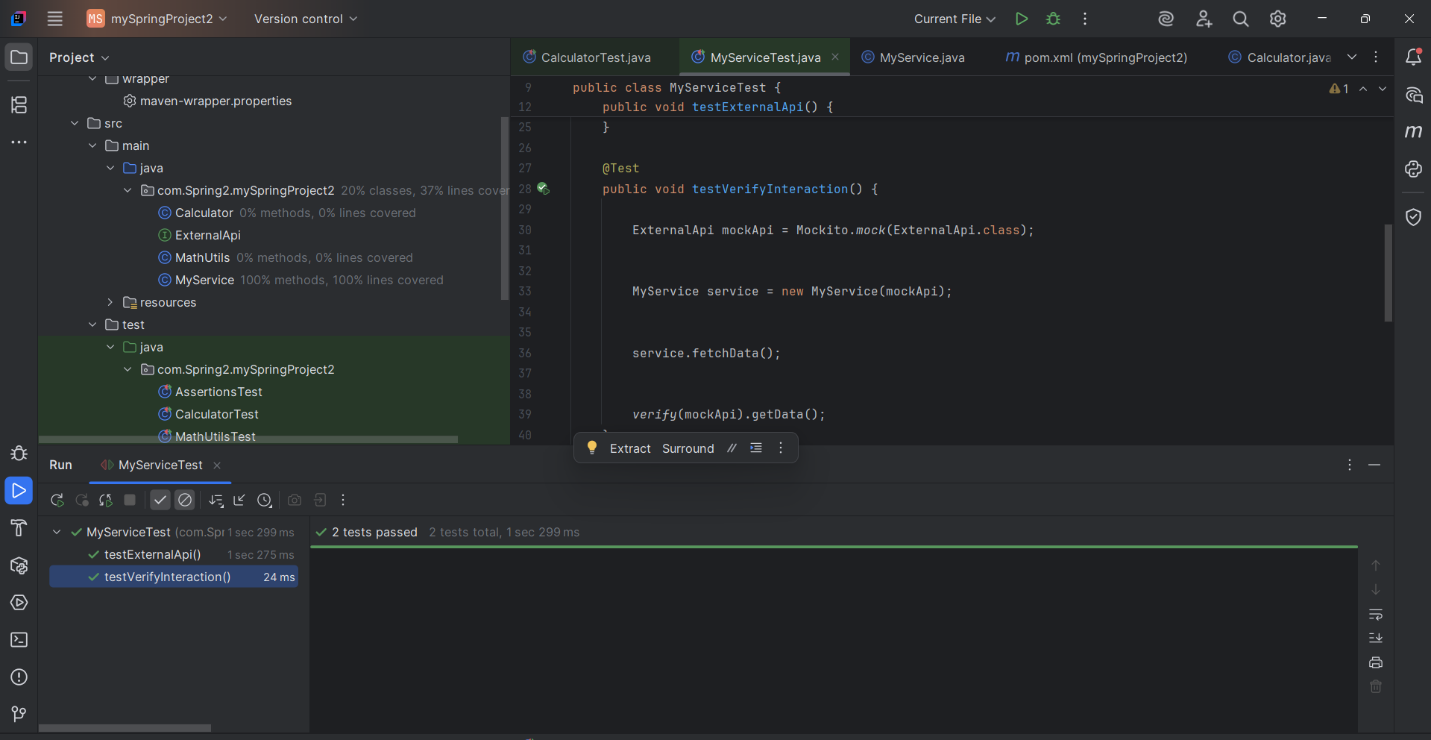
<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.11.0</version>

<scope>test</scope>

</dependency>



**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.

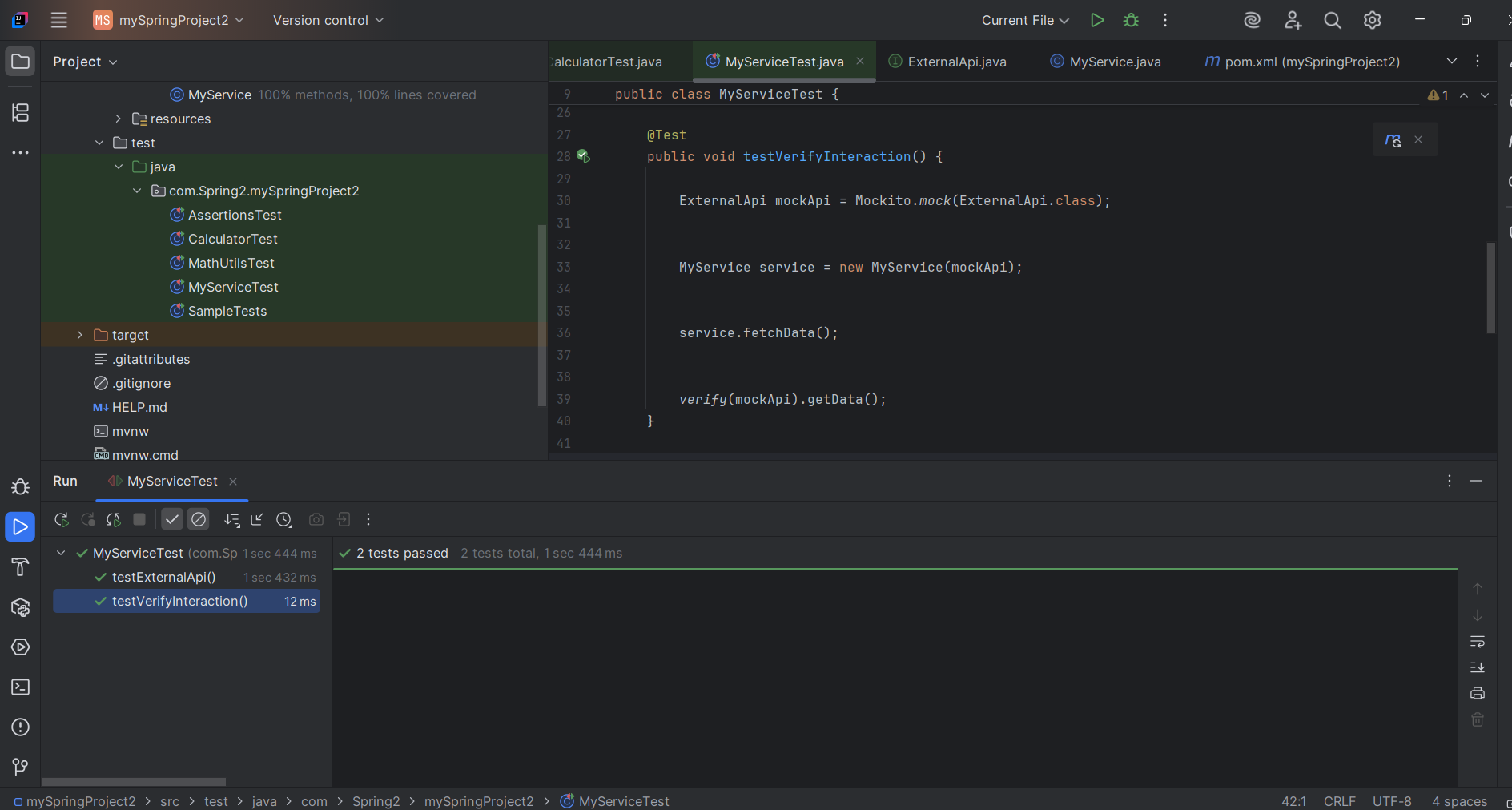
3. Verify the interaction.

Solution Code:

**MyServiceTest.java:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
  
 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  
 public void testVerifyInteraction() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 MyService service = new MyService(mockApi);  
  
  
 service.fetchData();  
  
  
 *verify*(mockApi).getData();  
 }

}



**Exercise 3: Argument Matching Scenario:**

You need to verify that a method is called with specific arguments.

Steps: 1. Create a mock object.

2. Call the method with specific arguments.

3. Use argument matchers to verify the interaction.

**CODE:**

**ExternalApi.java:**

package com.Spring2.mySpringProject2;  
  
public interface ExternalApi {  
 String getData();  
 void sendData(String data);

}

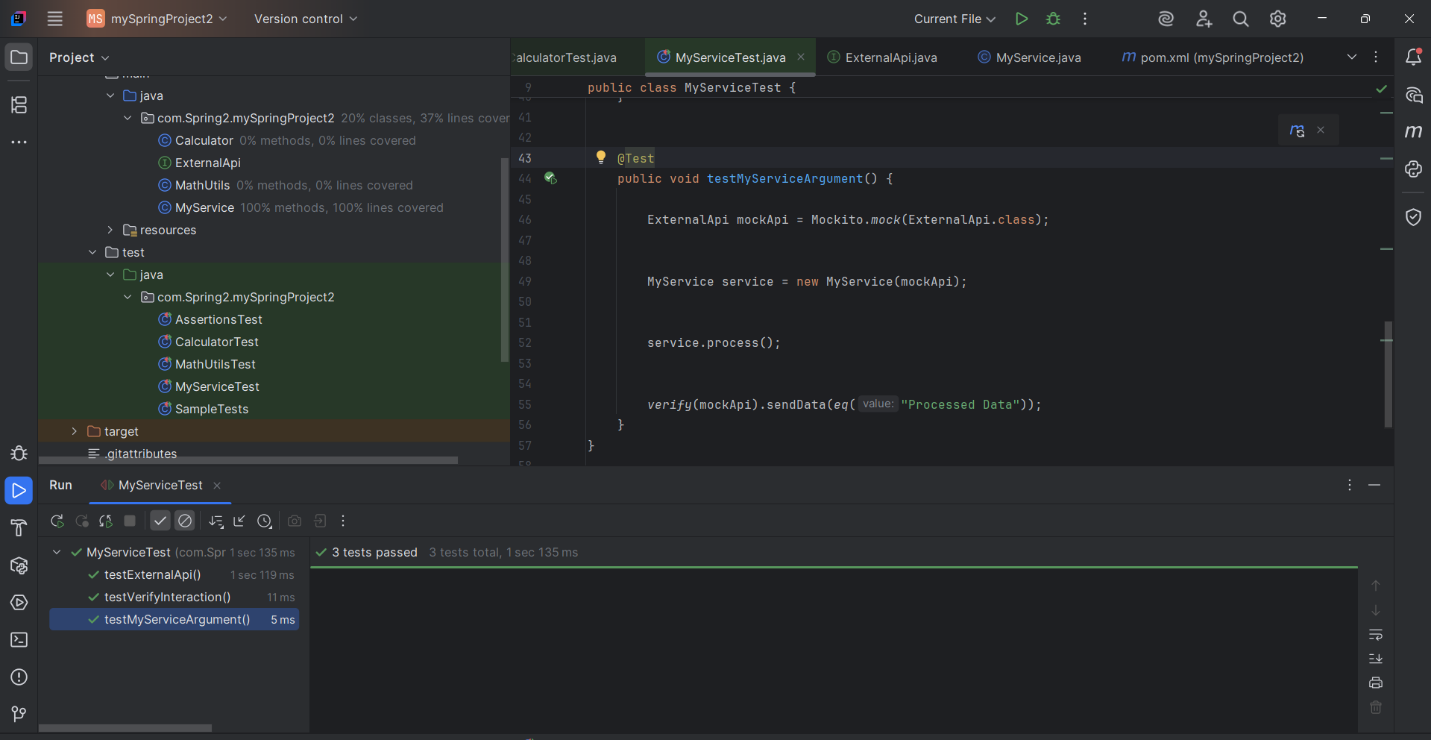
**MyService.java:**

package com.Spring2.mySpringProject2;  
  
public class MyService {  
  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
  
 public void process() {  
 api.sendData("Processed Data");  
 }

}

**MyServiceTest.java:**

@Test  
public void testMyServiceArgument() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 MyService service = new MyService(mockApi);  
  
  
 service.process();  
  
  
 *verify*(mockApi).sendData(*eq*("Processed Data"));  
}



**Exercise 4: Handling Void Methods Scenario:**

You need to test a void method that performs some action.

Steps: 1. Create a mock object.

2. Stub the void method.

3. Verify the interaction.

**ExternalApi.java:**

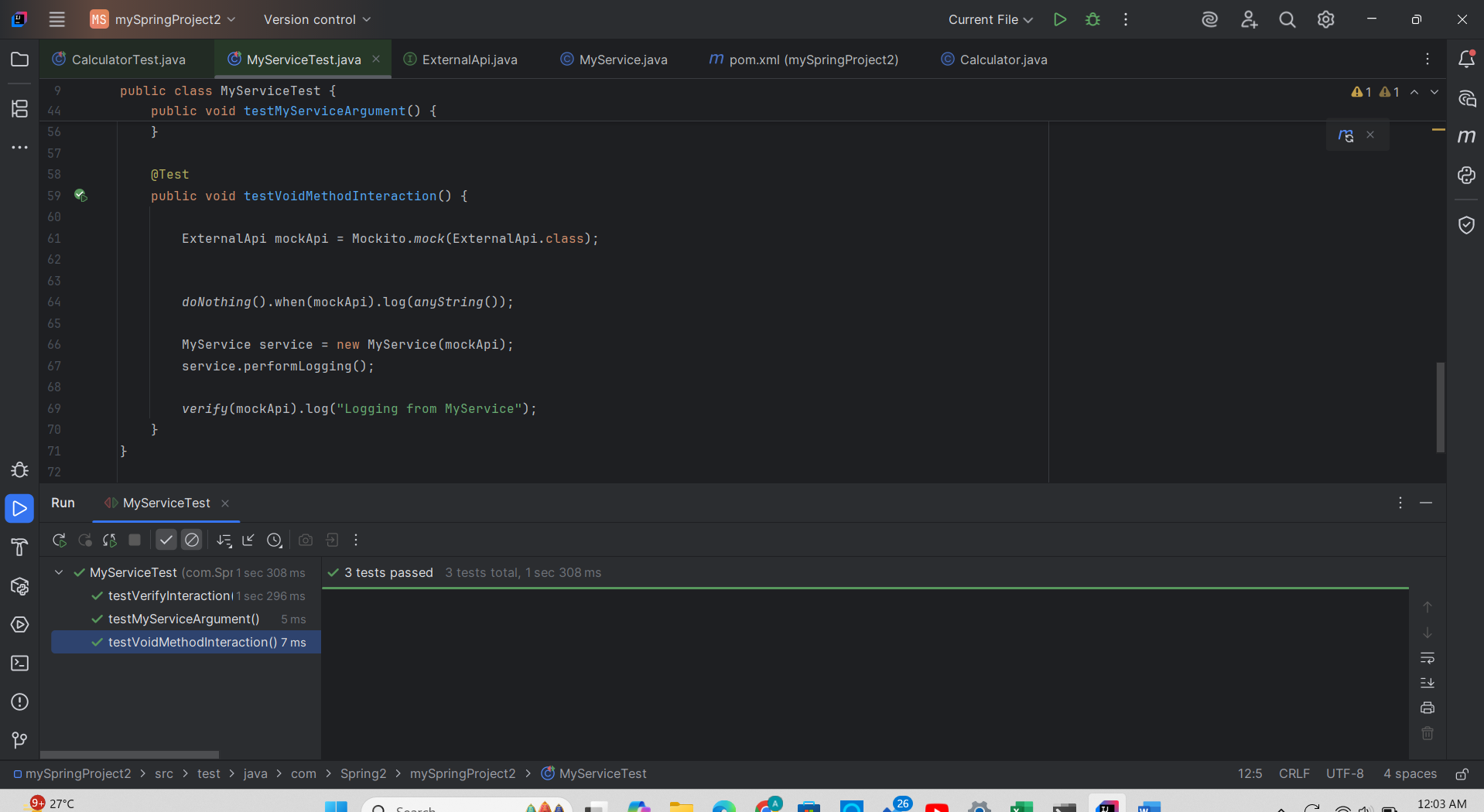
void log(String message);

**MyService.java:**

public void performLogging() {  
 api.log("Logging from MyService");  
}

**MyServiceTest.java:**

@Test  
public void testVoidMethodInteraction() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 *doNothing*().when(mockApi).log(*anyString*());  
  
 MyService service = new MyService(mockApi);  
 service.performLogging();  
  
 *verify*(mockApi).log("Logging from MyService");  
}



**Exercise 5: Mocking and Stubbing with Multiple Returns Scenario:**

You need to test a service that depends on an external API with multiple return values.

Steps: 1. Create a mock object for the external API.

2. Stub the methods to return different values on consecutive calls.

3. Write a test case that uses the mock object.

**ExternalApi.java:**

package com.Spring2.mySpringProject2;  
  
public interface ExternalApi {  
  
 String getData();  
 void sendData(String data);  
 void log(String message);  
  
  
}

**MyServiceTest.java:**

package com.Spring2.mySpringProject2;  
  
public class MyService {  
  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
  
 public void process() {  
 api.sendData("Processed Data");  
 }  
  
 public void performLogging() {  
 api.log("Logging from MyService");  
 }  
  
 public String fetchDataMultipleTimes() {  
 String first = api.getData();  
 String second = api.getData();  
 return first + " | " + second;  
 }  
}

**MyServiceTest.java:**

package com.Spring2.mySpringProject2;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.Mockito;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
  
 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  
 public void testVerifyInteraction() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 MyService service = new MyService(mockApi);  
  
  
 service.fetchData();  
  
  
 *verify*(mockApi).getData();  
 }  
  
  
 @Test  
 public void testMyServiceArgument() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 MyService service = new MyService(mockApi);  
  
  
 service.process();  
  
  
 *verify*(mockApi).sendData(*eq*("Processed Data"));  
 }  
  
 @Test  
 public void testVoidMethodInteraction() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
  
 *doNothing*().when(mockApi).log(*anyString*());  
  
 MyService service = new MyService(mockApi);  
 service.performLogging();  
  
 *verify*(mockApi).log("Logging from MyService");  
 }  
  
 @Test  
 public void testMultipleReturns() {  
  
 ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);  
  
 *when*(mockApi.getData())  
 .thenReturn("First Call")  
 .thenReturn("Second Call");  
  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchDataMultipleTimes();  
  
 System.*out*.println(result); // Output: First Call | Second Call  
 *verify*(mockApi, *times*(2)).getData();  
 }  
  
}

