**Week 2 : Mockito Hands-On Exercises**

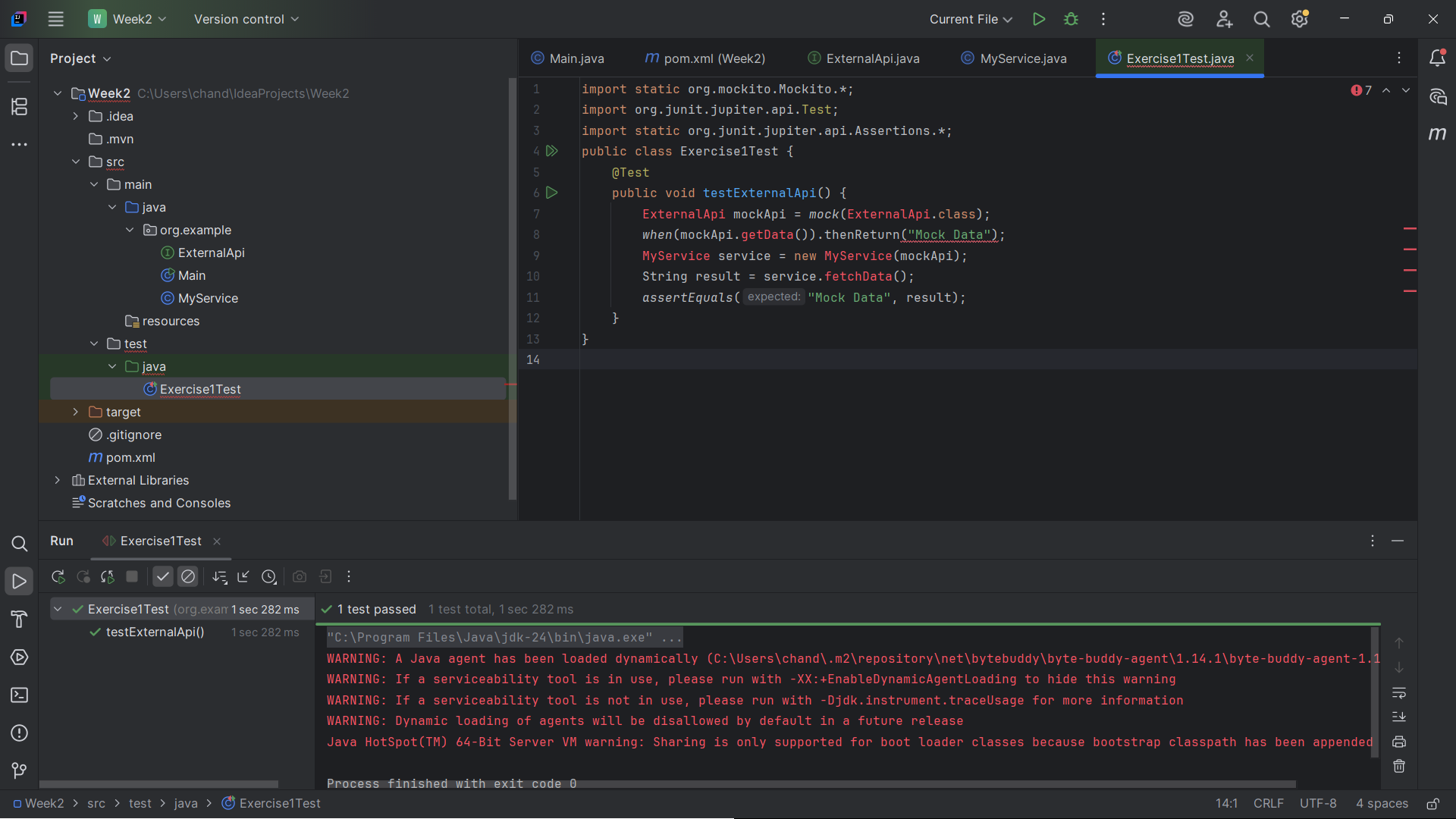
**Exercise 1: Mocking and Stubbing**

**CODE :**

**Excercise1Test.java**

import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
public class Exercise1Test {  
 @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);  
 }  
}

**OUTPUT:**



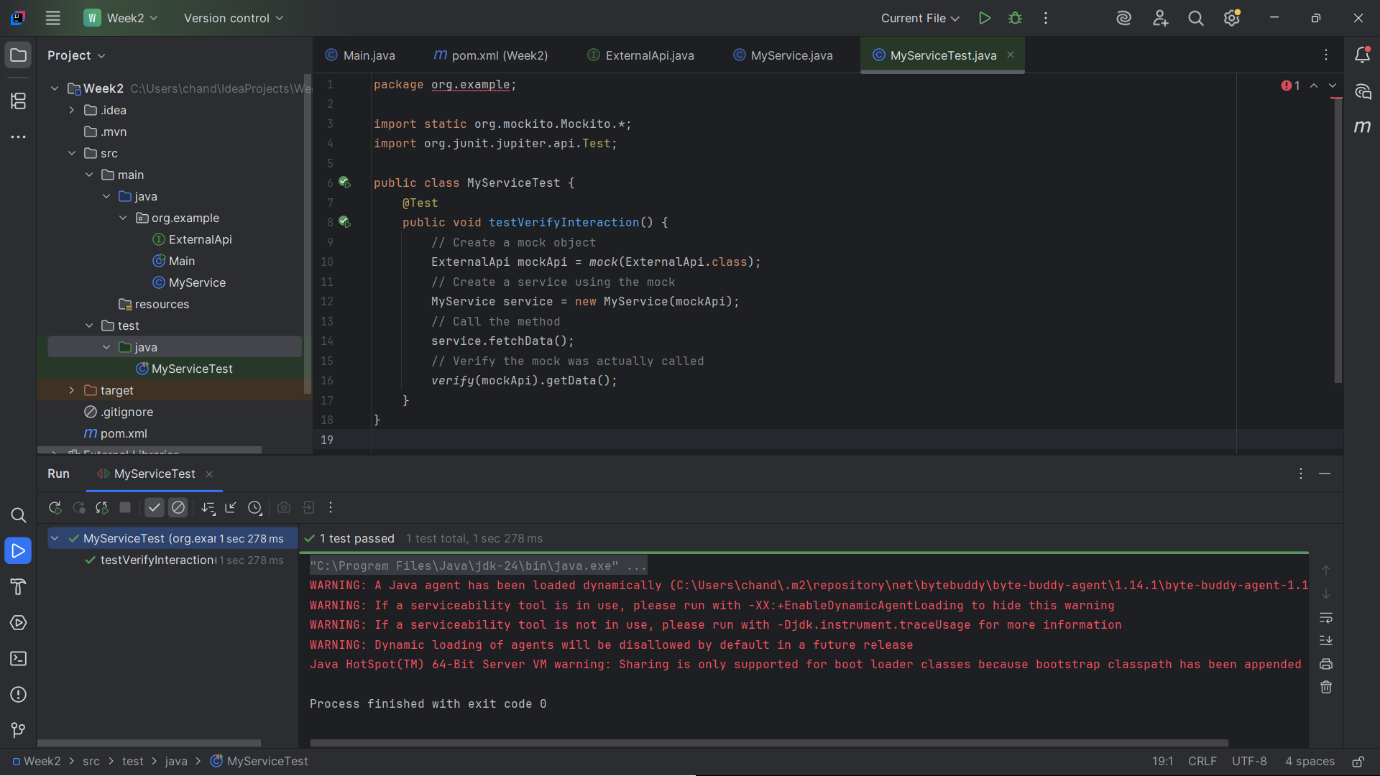
**Exercise 2: Verifying Interactions**

**CODE :**

**MyServiceTest.java**

package org.example;  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
public class MyServiceTest {  
 @Test  
 public void testVerifyInteraction() {  
 // Create a mock object  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 // Create a service using the mock  
 MyService service = new MyService(mockApi);  
 // Call the method  
 service.fetchData();  
 // Verify the mock was actually called  
 *verify*(mockApi).getData();  
 }  
}

**OUTPUT:**



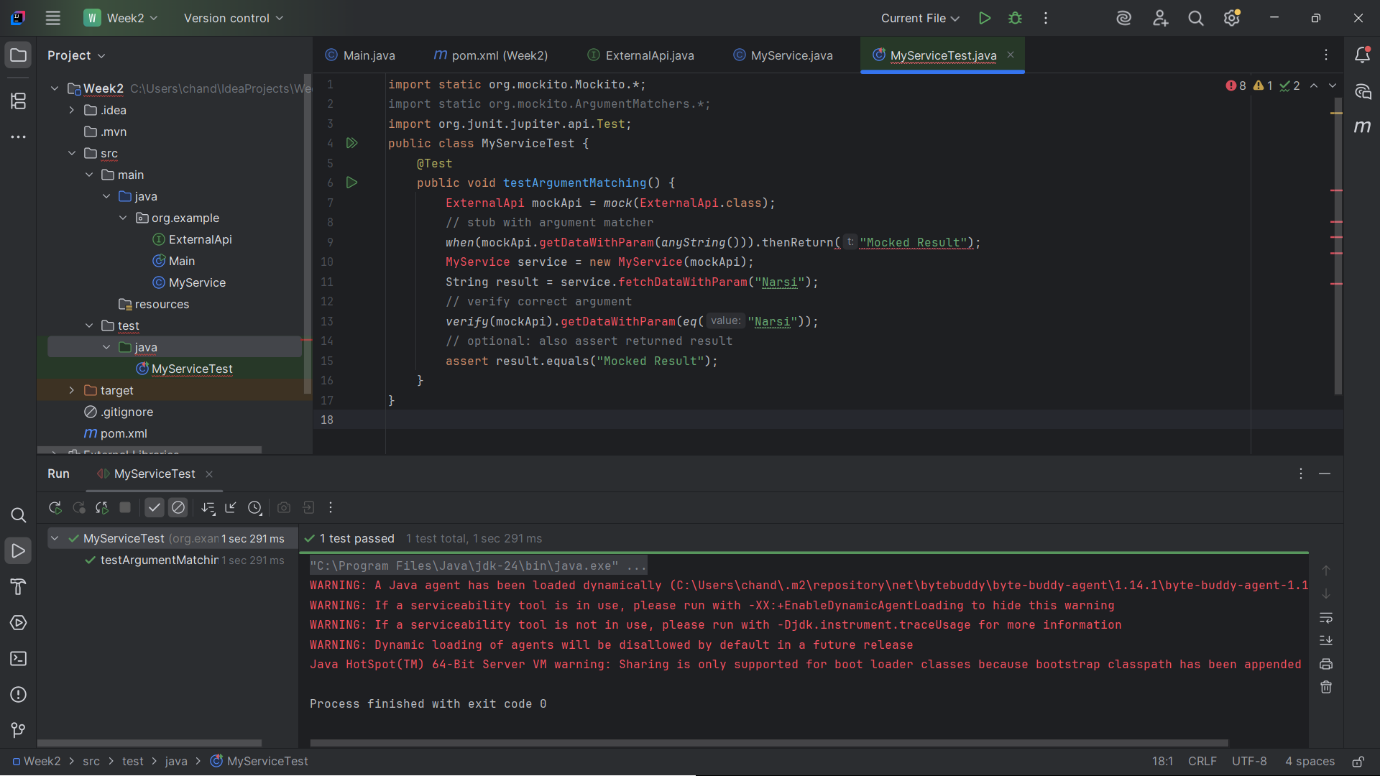
**Exercise 3: Argument Matching**

**CODE :**

**MyServiceTest.java**

import static org.mockito.Mockito.\*;  
import static org.mockito.ArgumentMatchers.\*;  
import org.junit.jupiter.api.Test;  
public class MyServiceTest {  
 @Test  
 public void testArgumentMatching() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 // stub with argument matcher  
 *when*(mockApi.getDataWithParam(*anyString*())).thenReturn("Mocked Result");  
 MyService service = new MyService(mockApi);  
 String result = service.fetchDataWithParam("Narsi");  
 // verify correct argument  
 *verify*(mockApi).getDataWithParam(*eq*("Narsi"));  
 // optional: also assert returned result  
 assert result.equals("Mocked Result");  
 }  
}

**OUTPUT:**

****

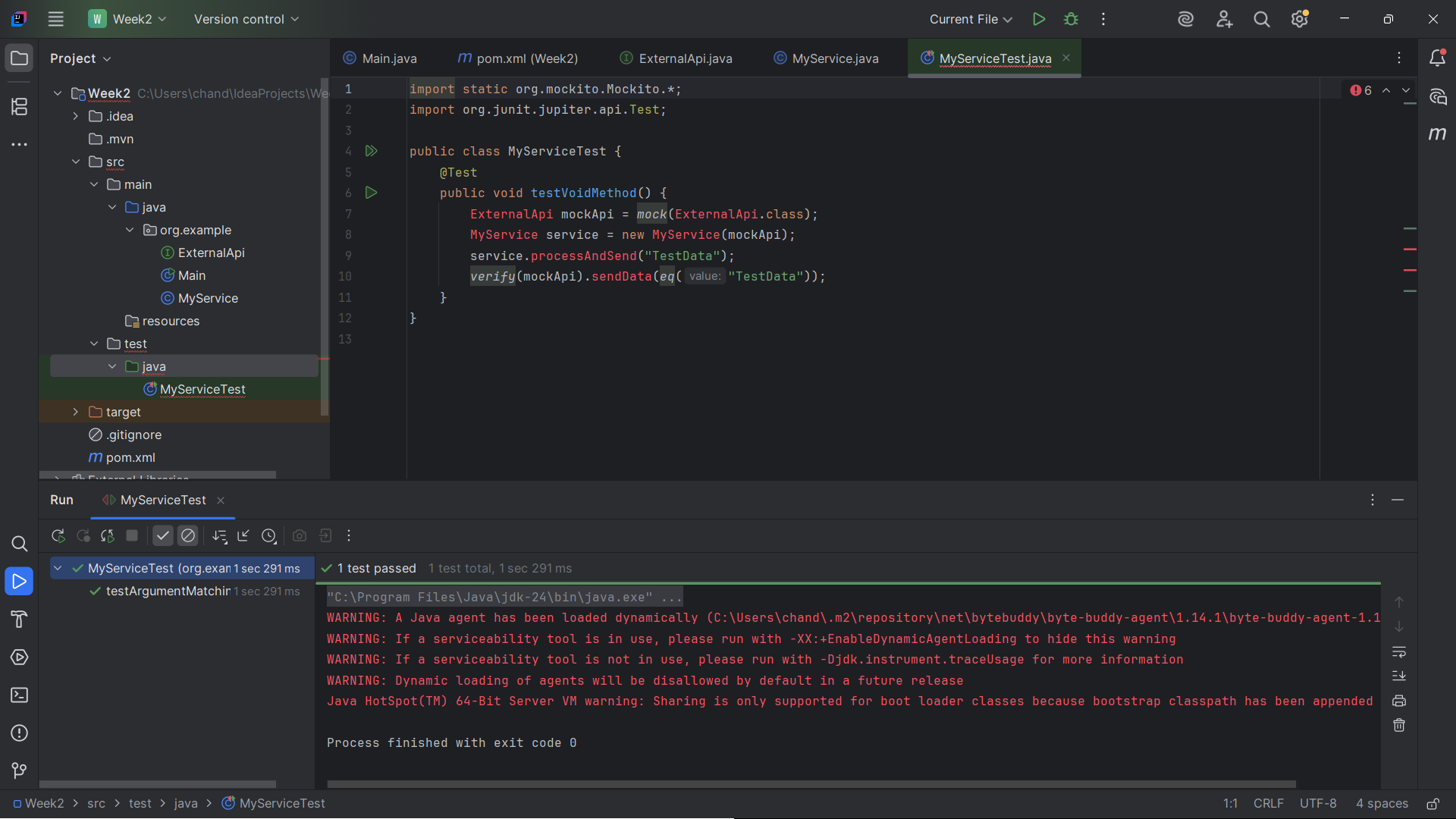
**Exercise 4: Handling Void Methods**

**CODE :**

**MyServiceTest.java**

import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
  
public class MyServiceTest {  
 @Test  
 public void testVoidMethod() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
 service.processAndSend("TestData");  
 *verify*(mockApi).sendData(*eq*("TestData"));  
 }  
}

**OUTPUT:**



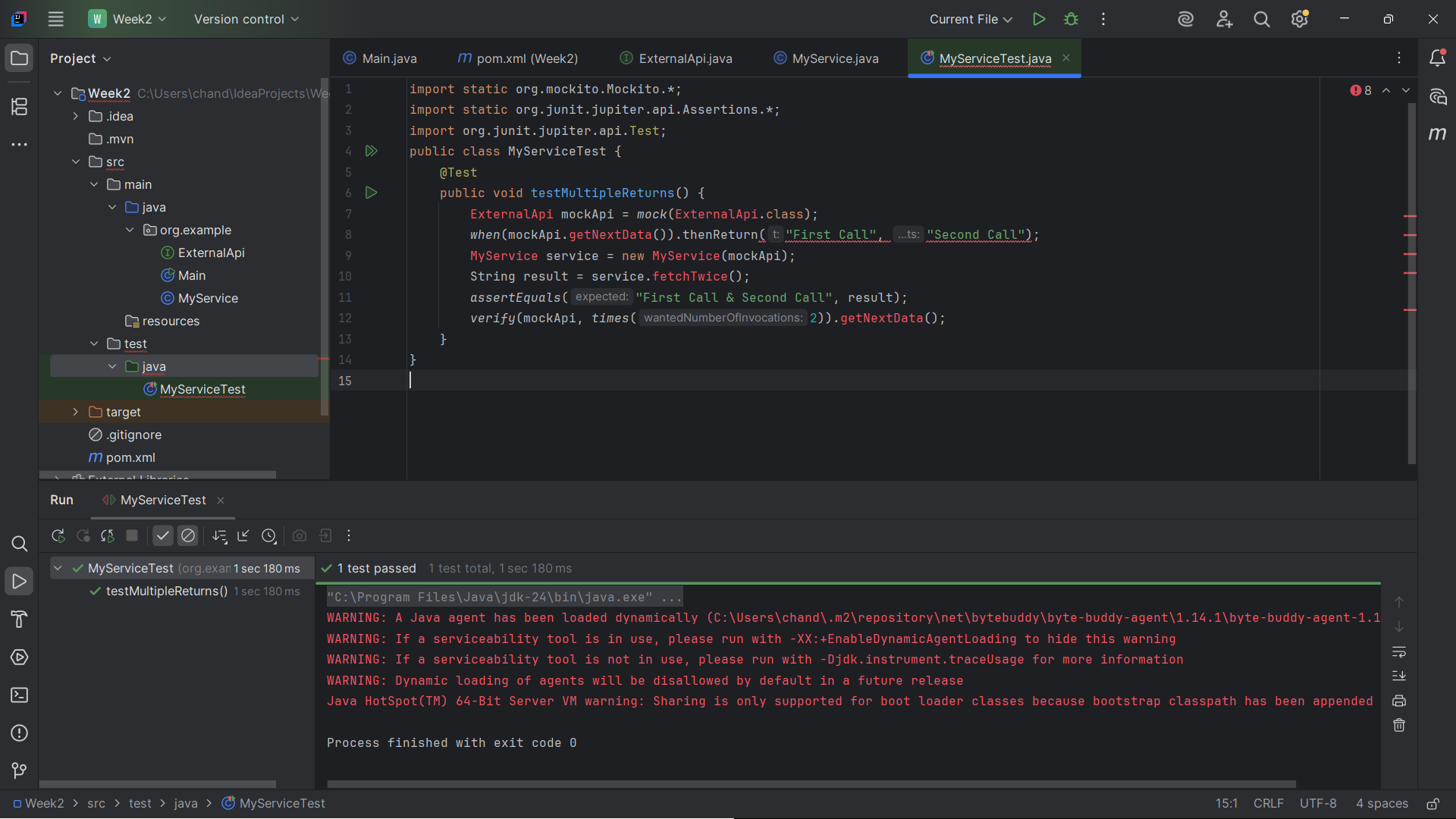
**Exercise 5: Mocking and Stubbing with Multiple Returns**

**CODE :**

**MyServiceTest.java**

import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
public class MyServiceTest {  
 @Test  
 public void testMultipleReturns() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getNextData()).thenReturn("First Call", "Second Call");  
 MyService service = new MyService(mockApi);  
 String result = service.fetchTwice();  
 *assertEquals*("First Call & Second Call", result);  
 *verify*(mockApi, *times*(2)).getNextData();  
 }  
}

**OUTPUT:**

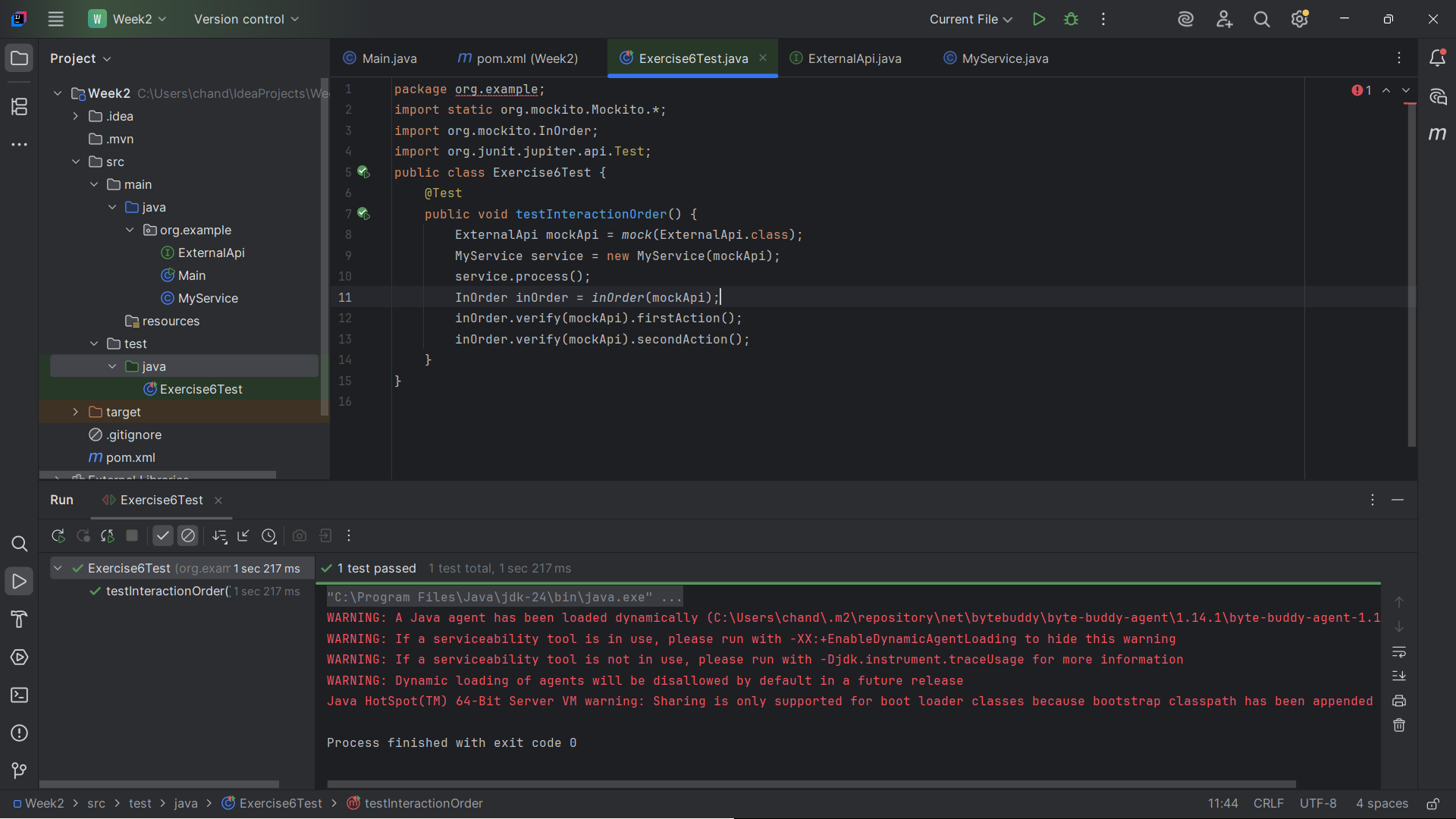


**Exercise 6: Verifying Interaction Order**

**CODE :**

**Exercise6Test.java**

package org.example;  
import static org.mockito.Mockito.\*;  
import org.mockito.InOrder;  
import org.junit.jupiter.api.Test;  
public class Exercise6Test {  
 @Test  
 public void testInteractionOrder() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
 service.process();  
 InOrder inOrder = *inOrder*(mockApi);  
 inOrder.verify(mockApi).firstAction();  
 inOrder.verify(mockApi).secondAction();  
 }  
}

**OUTPUT:** 

**Exercise 7: Handling Void Methods with Exceptions**

**CODE :**

**Exercise7Test.java**

import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
import org.junit.jupiter.api.Test;  
public class Exercise7Test {  
 @Test  
 public void testVoidMethodThrowsException() throws Exception {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *doThrow*(new RuntimeException("Simulated Failure"))  
 .when(mockApi).dangerousAction();  
 MyService service = new MyService(mockApi);  
 Exception ex = *assertThrows*(RuntimeException.class, () -> {  
 service.performDangerous();  
 });  
 *assertEquals*("Simulated Failure", ex.getMessage());  
 *verify*(mockApi).dangerousAction();  
 }  
}

**OUTPUT:**

