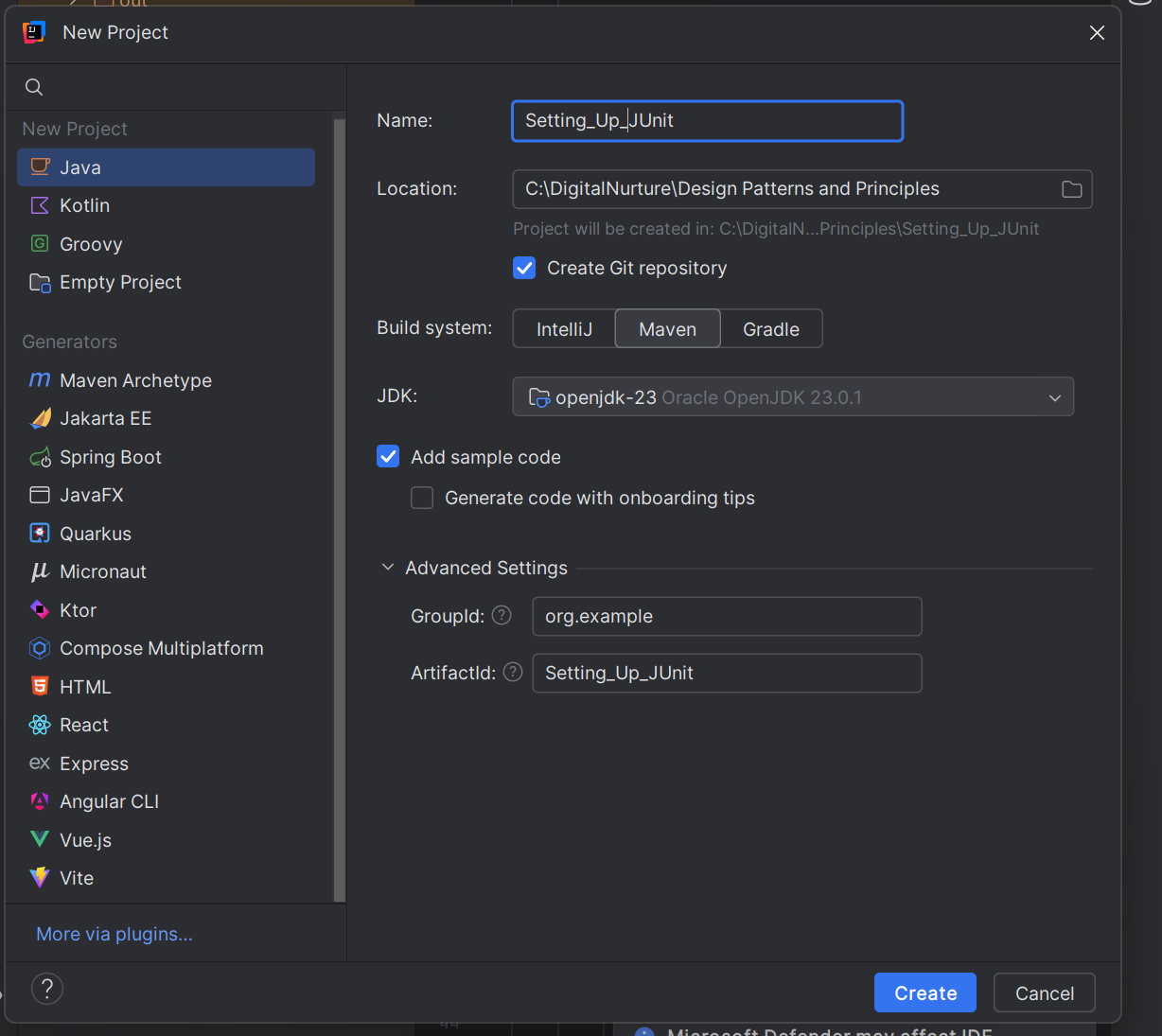
**JUnit Testing Exercises**

**Exercise 1: Setting Up Junit**

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

**Step1: Initial project configurations in Intellij.**



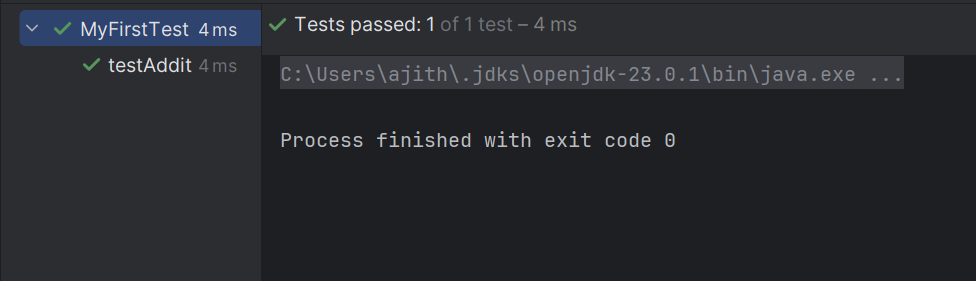
**Step2: Add dependencies to pom.xml .**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Setting\_Up\_JUnit</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
  
 <properties>  
 <maven.compiler.source>23</maven.compiler.source>  
 <maven.compiler.target>23</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
  
  
</project>

**3.Create a new test class in the project.**

**MyFirstTest.java**

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class MyFirstTest {  
 @Test  
 public void testAddition() {  
 int result = 2 + 3;  
 assertEquals(5, result);  
 }  
}



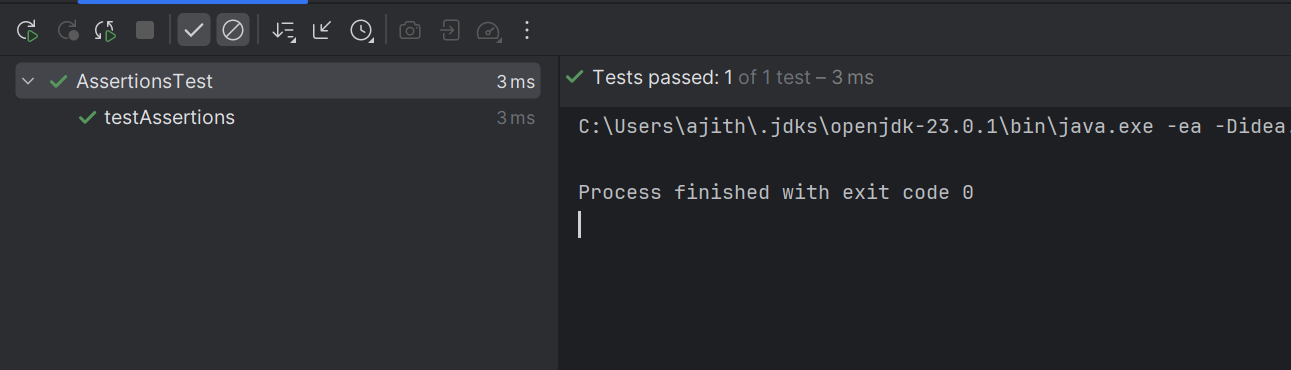
**Exercise 3: Assertions in Junit**

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

**AssertionsTest.java**

import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class AssertionsTest {  
 @Test  
 public void testAssertions() {  
 // 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**

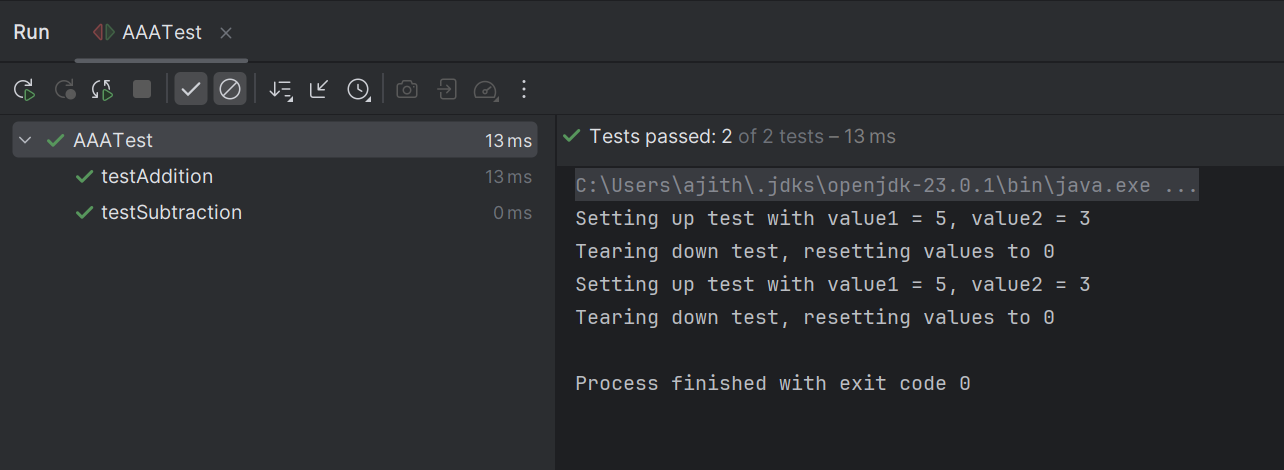
****

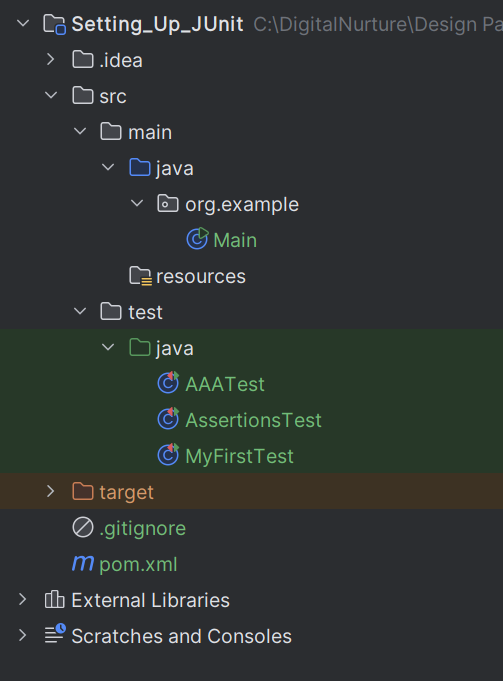
**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Scenario: You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.**

**AAATest.java**

import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class AAATest {  
 private int value1;  
 private int value2;  
  
 @Before  
 public void setUp() {  
 value1 = 5;  
 value2 = 3;  
 System.*out*.println("Setting up test with value1 = " + value1 + ", value2 = " + value2);  
 }  
  
 @After  
 public void tearDown() {  
 value1 = 0;  
 value2 = 0;  
 System.*out*.println("Tearing down test, resetting values to 0");  
 }  
  
 @Test  
 public void testAddition() {  
   
 int result = value1 + value2;  
 *assertEquals*(8, result);  
 }  
  
 @Test  
 public void testSubtraction() {  
 int result = value1 - value2;  
 *assertEquals*(2, result);  
 }  
}

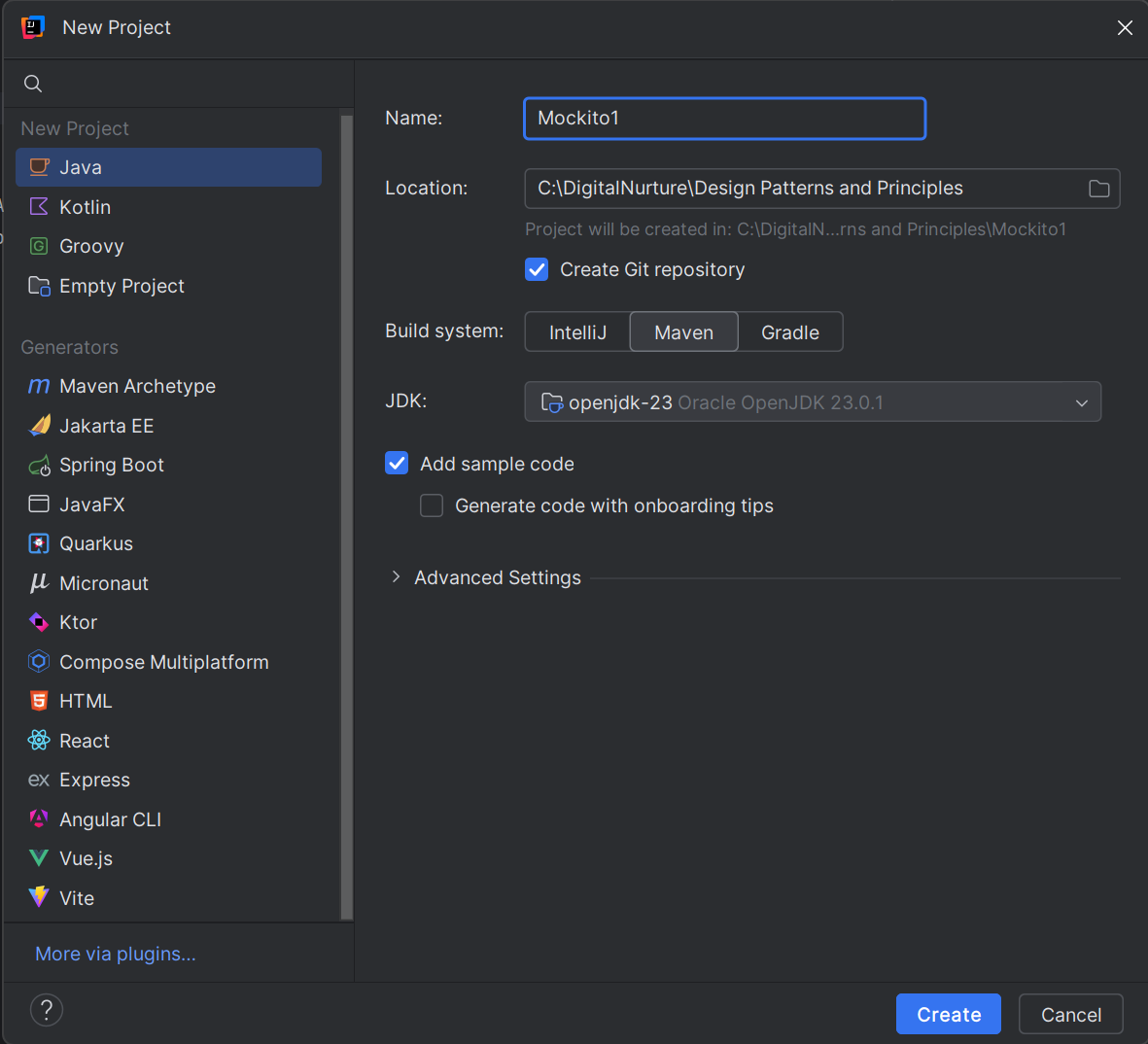
****

****

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing  
You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.**

**Step1: Initial project configurations in Intellij.**

****

**Step2: Add dependencies to pom.xml .**

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>Mockito</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <dependencies>  
 <!-- JUnit 5 -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.11.0</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-engine</artifactId>  
 <version>5.11.0</version>  
 <scope>test</scope>  
 </dependency>  
 <!-- Mockito -->  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.14.2</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-junit-jupiter</artifactId>  
 <version>5.14.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
 <properties>  
 <maven.compiler.source>23</maven.compiler.source>  
 <maven.compiler.target>23</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
</project>

**Step3:Initalise the nesscary classes**

**ExternalApi.java**

package org.example;  
  
public interface ExternalApi {  
 String getData();  
}

**MyService.java**

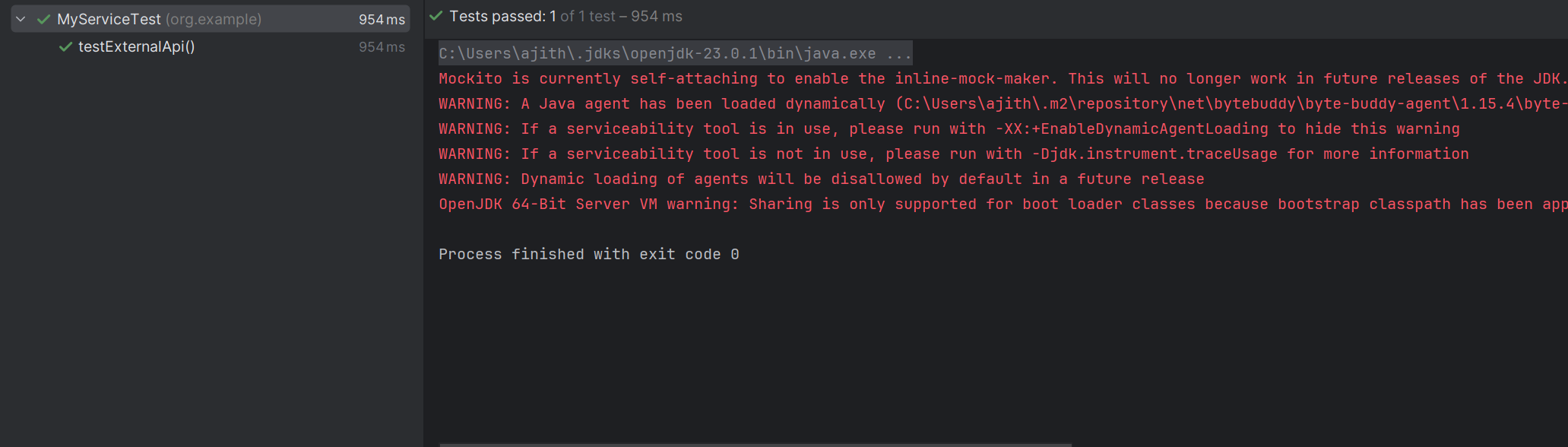
package org.example;  
  
public class MyService {  
 private final ExternalApi externalApi;  
  
 public MyService(ExternalApi externalApi) {  
 this.externalApi = externalApi;  
 }  
  
 public String fetchData() {  
 return externalApi.getData();  
 }  
}

**Step4:Initalise code for Service Testing**

**MyServiceTest.java**

package org.example;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.*when*;  
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); // Use traditional syntax  
 }  
}

**Output**

****

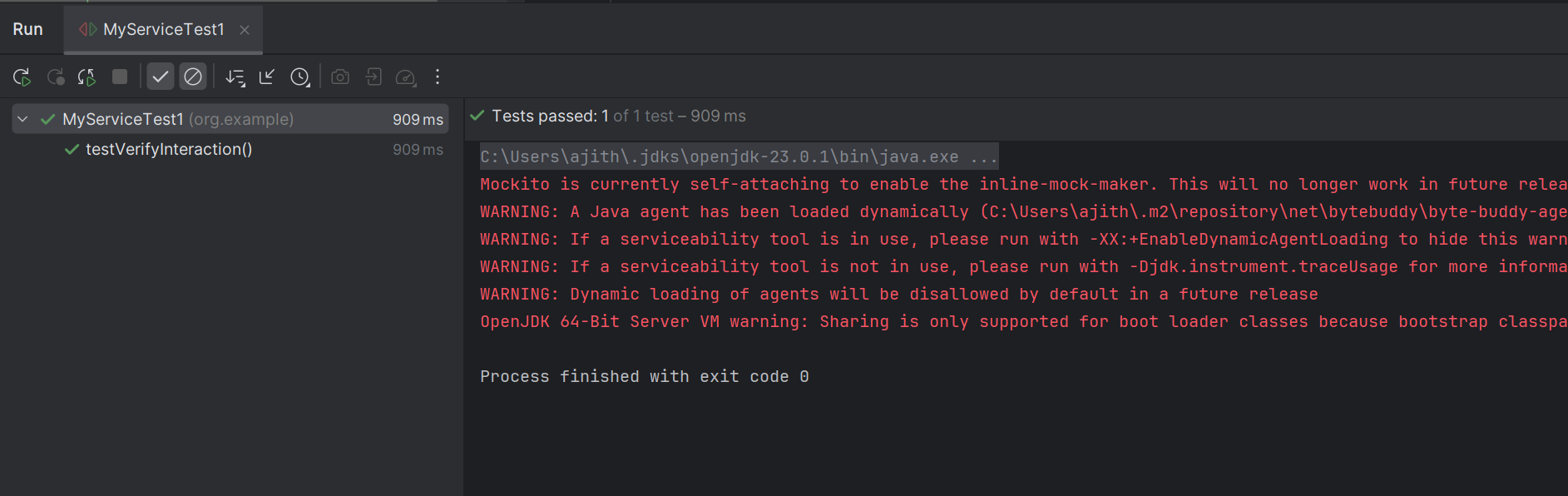
**Exercise 3: Argument Matching**

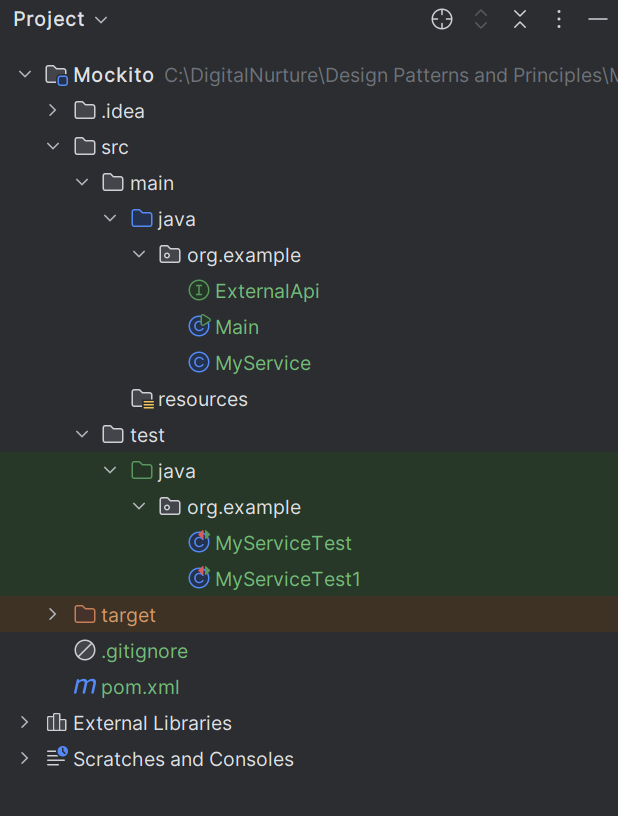
**Scenario: You need to verify that a method is called with specific arguments.**

**Step1:Define the Test code**

package org.example;  
  
import static org.mockito.Mockito.*verify*;  
import static org.mockito.Mockito.*mock*;  
import org.junit.jupiter.api.Test;  
  
public class MyServiceTest1 {  
 @Test  
 public void testVerifyInteraction() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
 *verify*(mockApi).getData();  
 }  
}

**Output**

****

****