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;

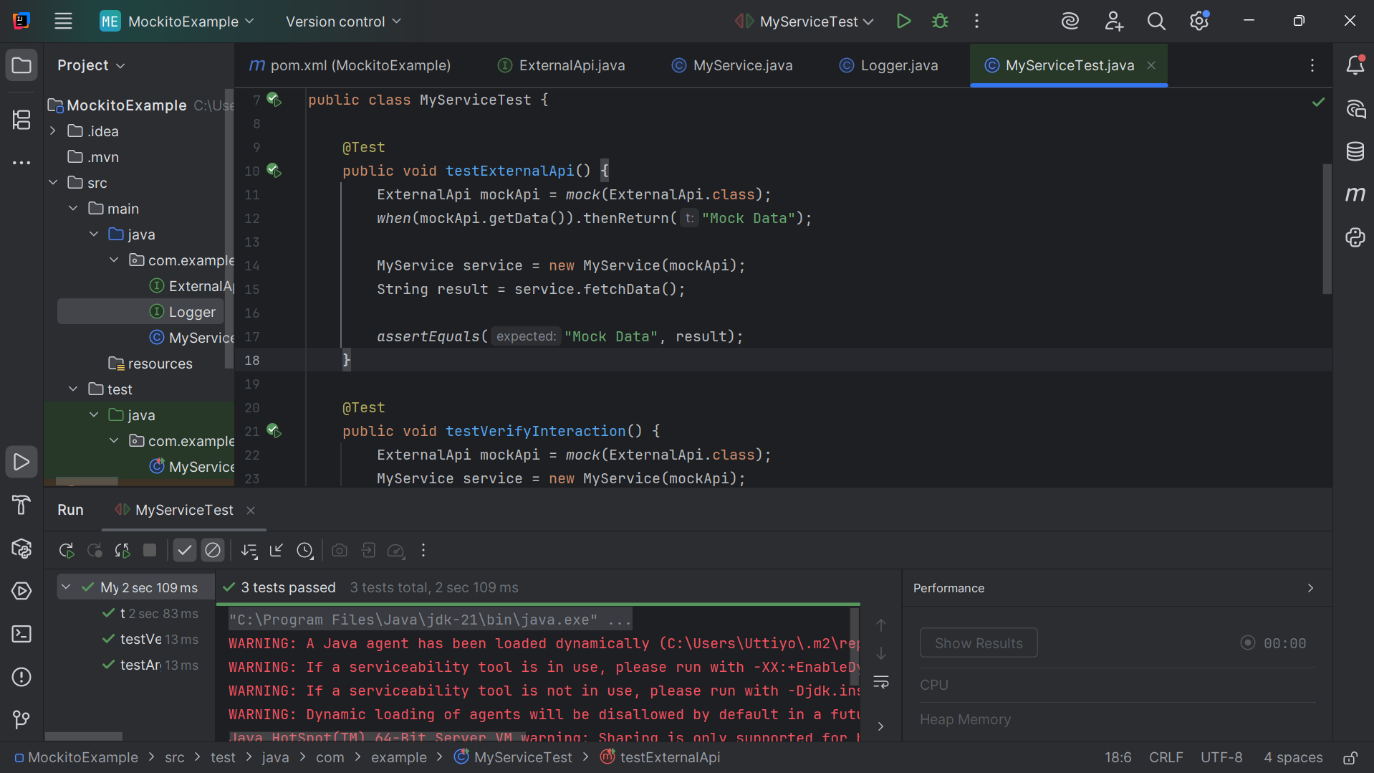
public class MyServiceTest { @Test

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);

}

}

# 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:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test; import org.mockito.Mockito;

public class MyServiceTest { @Test

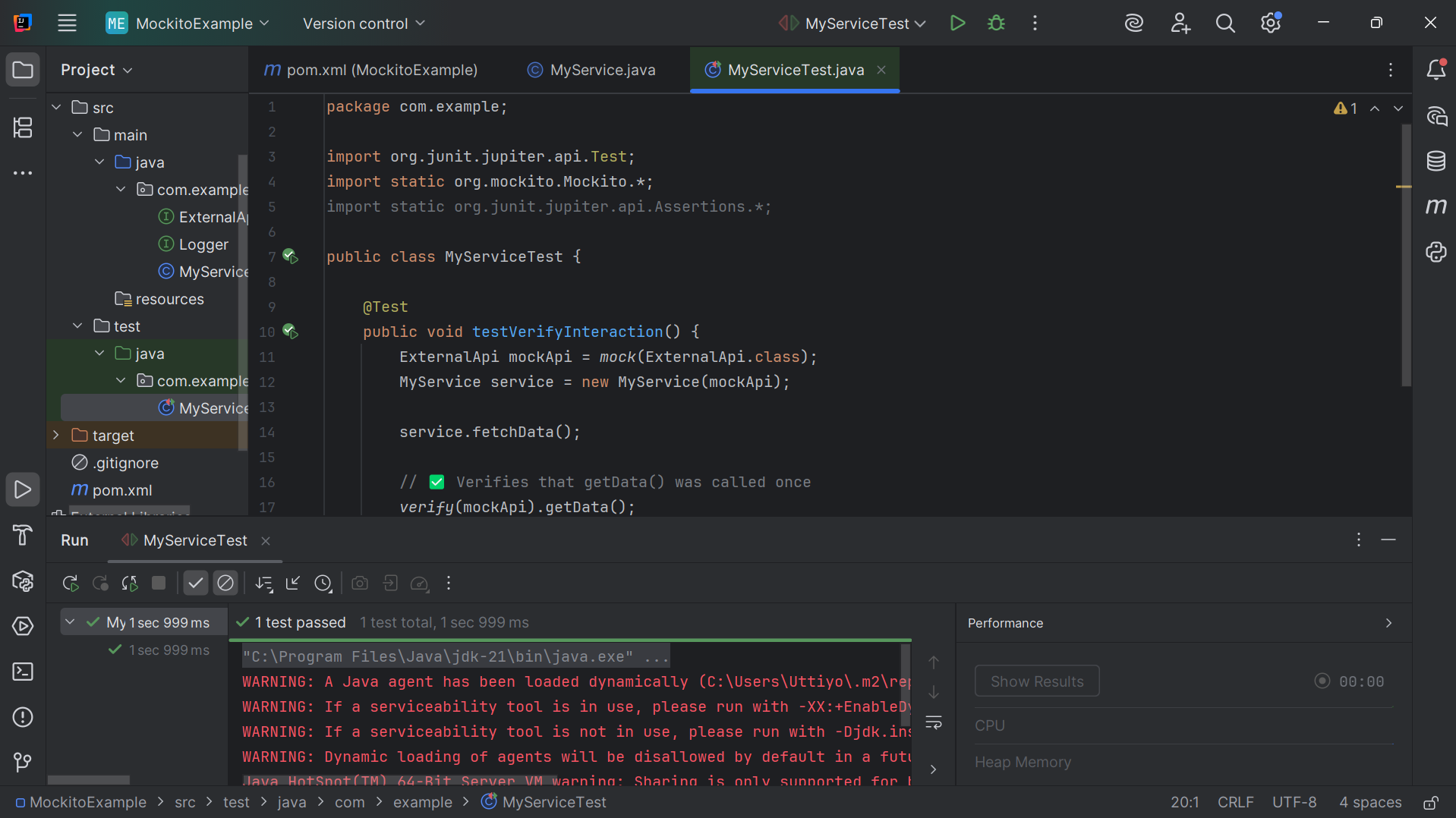
public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); MyService service = new MyService(mockApi); service.fetchData();

verify(mockApi).getData();

}

}



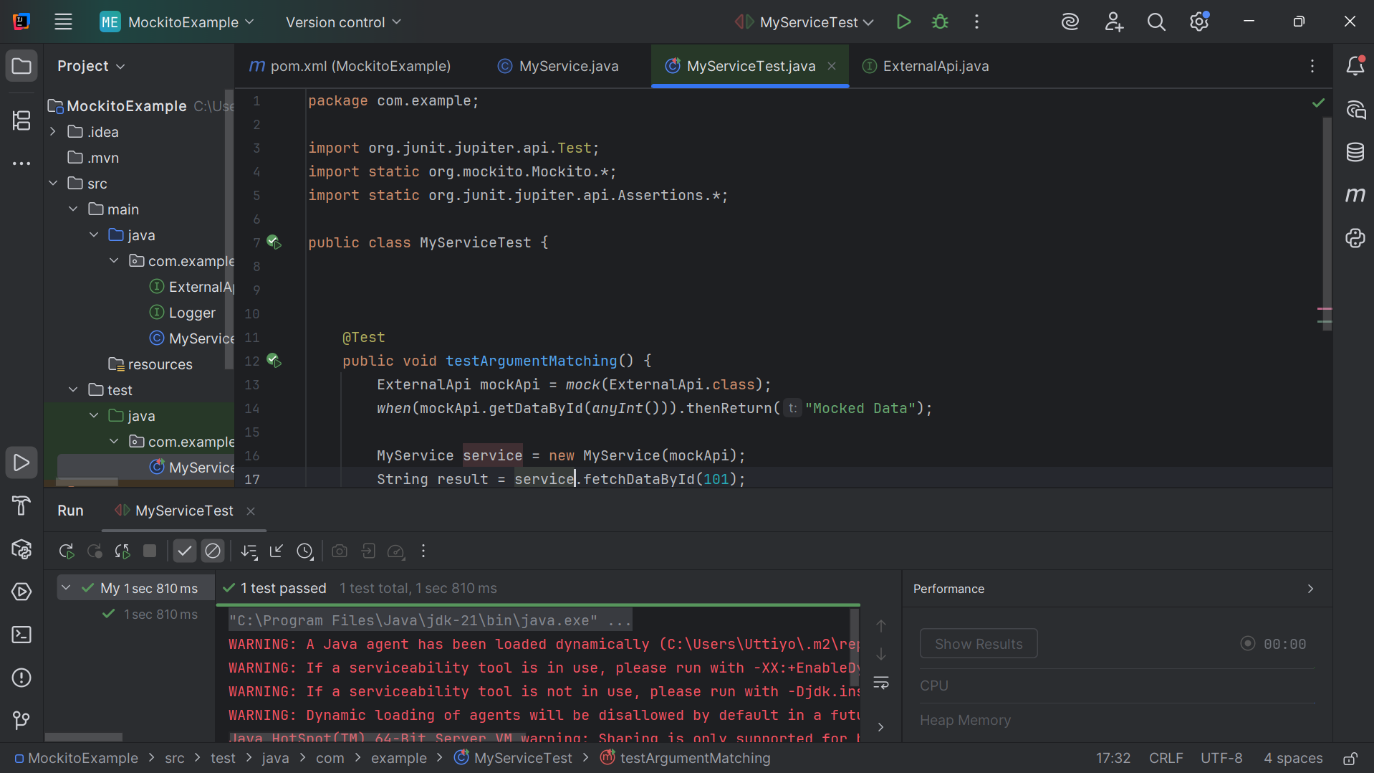
# Exercise 3: Argument Matching

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.

**package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MyServiceTest {  
  
  
  
 @Test  
 public void testArgumentMatching() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getDataById(*anyInt*())).thenReturn("Mocked Data");  
  
 MyService service = new MyService(mockApi);  
 String result = service.fetchDataById(101);  
  
 *assertEquals*("Mocked Data", result);  
 *verify*(mockApi).getDataById(*eq*(101));  
 }  
  
}**

# Exercise 4: Handling Void Methods

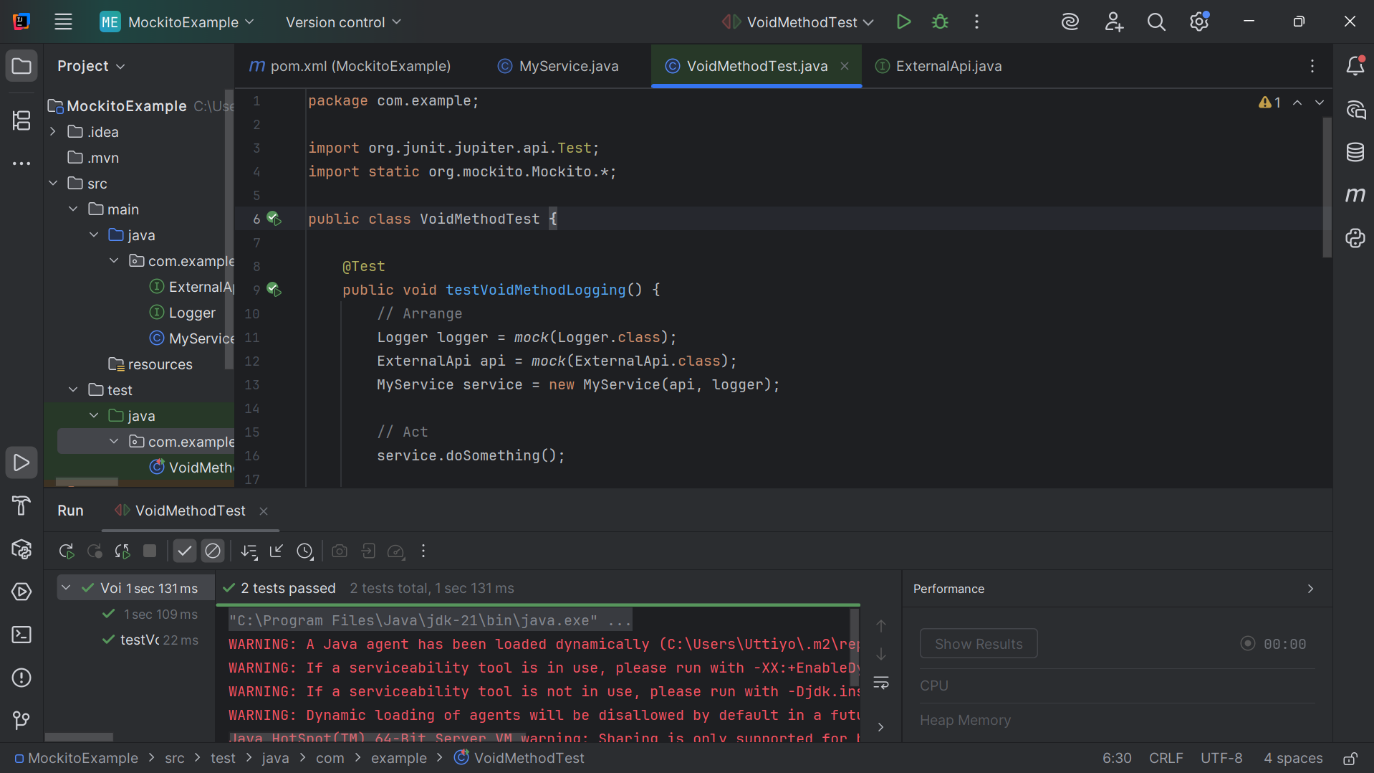
Scenario:

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.
4. package com.example;  
     
   public class MyService {  
    private final ExternalApi api;  
    private final Logger logger;  
     
    // Constructor with logger  
    public MyService(ExternalApi api, Logger logger) {  
    this.api = api;  
    this.logger = logger;  
    }  
     
    // Method to test  
    public void doSomething() {  
    logger.log("Processing...");  
    }  
   }

**2. package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
  
public class VoidMethodTest {  
  
 @Test  
 public void testVoidMethodLogging() {  
   
 Logger logger = *mock*(Logger.class);  
 ExternalApi api = *mock*(ExternalApi.class);  
 MyService service = new MyService(api, logger);  
  
   
 service.doSomething();  
  
   
 *verify*(logger).log("Processing...");  
 }  
  
 @Test  
 public void testVoidMethodThrowsException() {  
   
 Logger logger = *mock*(Logger.class);  
  
   
 *doThrow*(new RuntimeException("Log failed")).when(logger).log("fail");  
  
   
 org.junit.jupiter.api.Assertions.*assertThrows*(RuntimeException.class, () -> {  
 logger.log("fail");  
 });  
  
   
 *verify*(logger).log("fail");  
 }  
}**



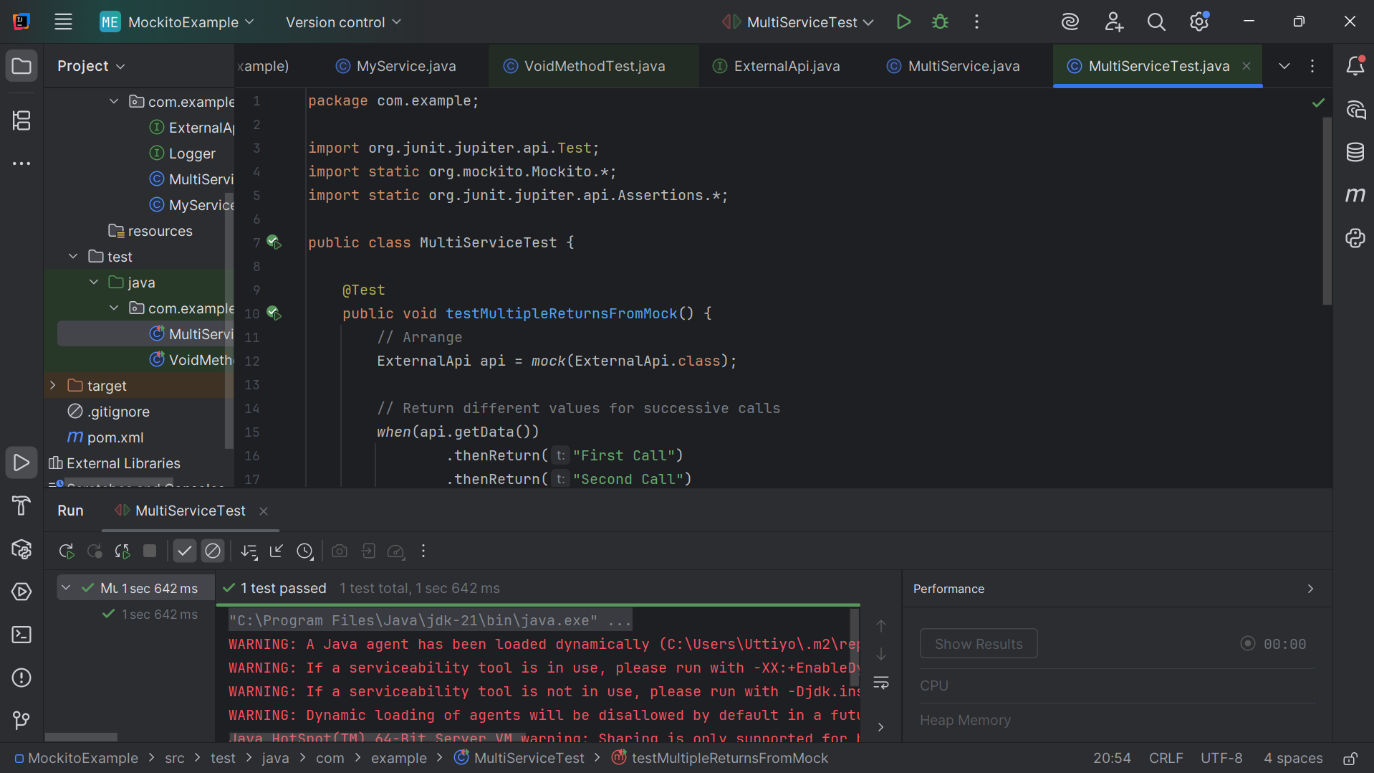
# Exercise 5: Mocking and Stubbing with Multiple Returns

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.
4. **package com.example;  
     
   public class MultiService {  
    private final ExternalApi api;  
     
    public MultiService(ExternalApi api) {  
    this.api = api;  
    }  
     
    public String getNextMessage() {  
    return api.getData();  
    }  
   }**
5. package com.example;  
     
   import org.junit.jupiter.api.Test;  
   import static org.mockito.Mockito.\*;  
   import static org.junit.jupiter.api.Assertions.\*;  
     
   public class MultiServiceTest {  
     
    @Test  
    public void testMultipleReturnsFromMock() {  
    // Arrange  
    ExternalApi api = *mock*(ExternalApi.class);  
     
    // Return different values for successive calls  
    *when*(api.getData())  
    .thenReturn("First Call")  
    .thenReturn("Second Call")  
    .thenReturn("Third Call");  
     
    MultiService service = new MultiService(api);  
     
    // Act + Assert  
    *assertEquals*("First Call", service.getNextMessage());  
    *assertEquals*("Second Call", service.getNextMessage());  
    *assertEquals*("Third Call", service.getNextMessage());  
    }  
   }

****

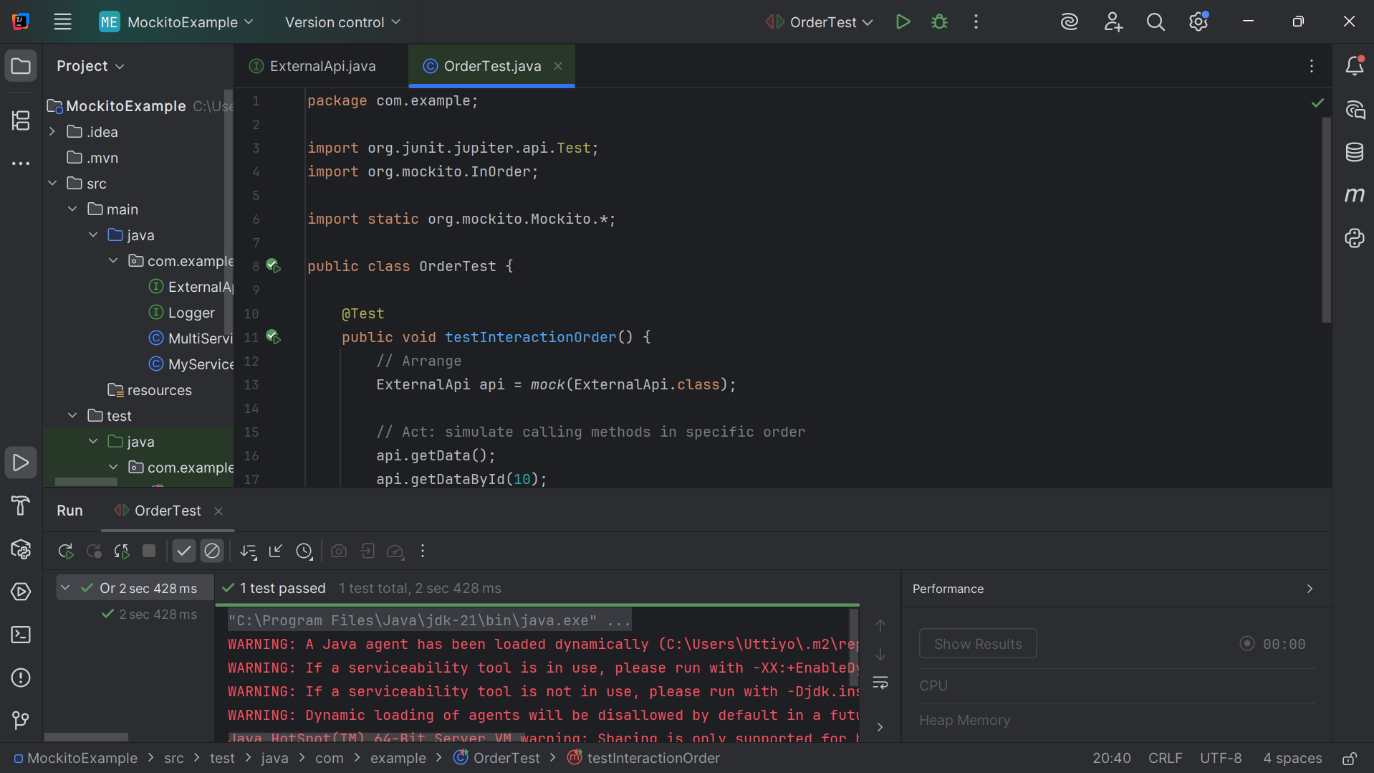
# Exercise 6: Verifying Interaction Order

Scenario:

You need to ensure that methods are called in a specific order.

Steps:

1. Create a mock object.
2. Call the methods in a specific order.
3. Verify the interaction order.

**package com.example;  
  
import org.junit.jupiter.api.Test;  
import org.mockito.InOrder;  
  
import static org.mockito.Mockito.\*;  
  
public class OrderTest {  
  
 @Test  
 public void testInteractionOrder() {  
 // Arrange  
 ExternalApi api = *mock*(ExternalApi.class);  
  
 // Act: simulate calling methods in specific order  
 api.getData();  
 api.getDataById(10);  
  
 // Assert: verify order  
 InOrder inOrder = *inOrder*(api);  
 inOrder.verify(api).getData();  
 inOrder.verify(api).getDataById(10);  
 }  
}**

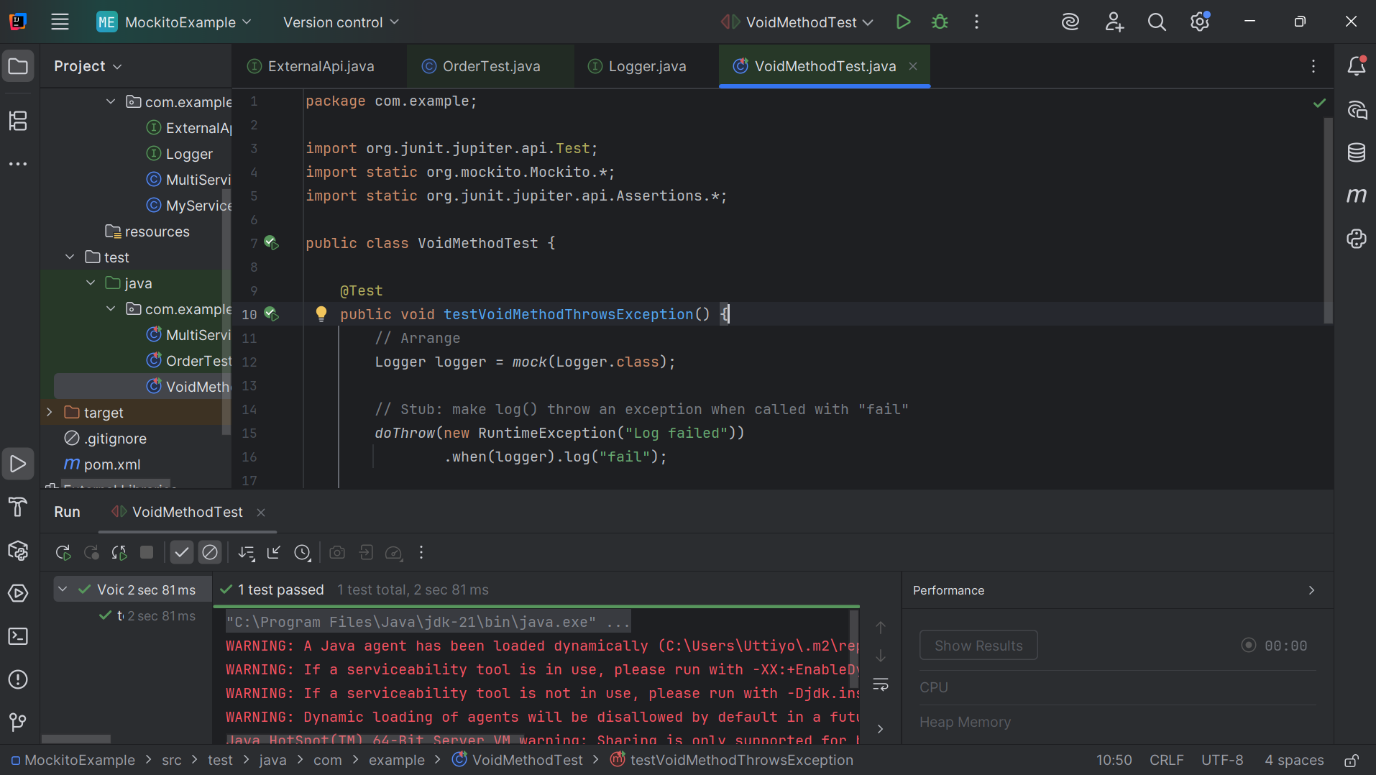
# Exercise 7: Handling Void Methods with Exceptions

Scenario:

You need to test a void method that throws an exception.

Steps:

1. Create a mock object.
2. Stub the void method to throw an exception.
3. Verify the interaction.

**package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class VoidMethodTest {  
  
 @Test  
 public void testVoidMethodThrowsException() {  
 // Arrange  
 Logger logger = *mock*(Logger.class);  
  
 // Stub: make log() throw an exception when called with "fail"  
 *doThrow*(new RuntimeException("Log failed"))  
 .when(logger).log("fail");  
  
 // Act + Assert: verify exception is thrown  
 *assertThrows*(RuntimeException.class, () -> logger.log("fail"));  
  
 // Verify interaction  
 *verify*(logger).log("fail");  
 }  
}**