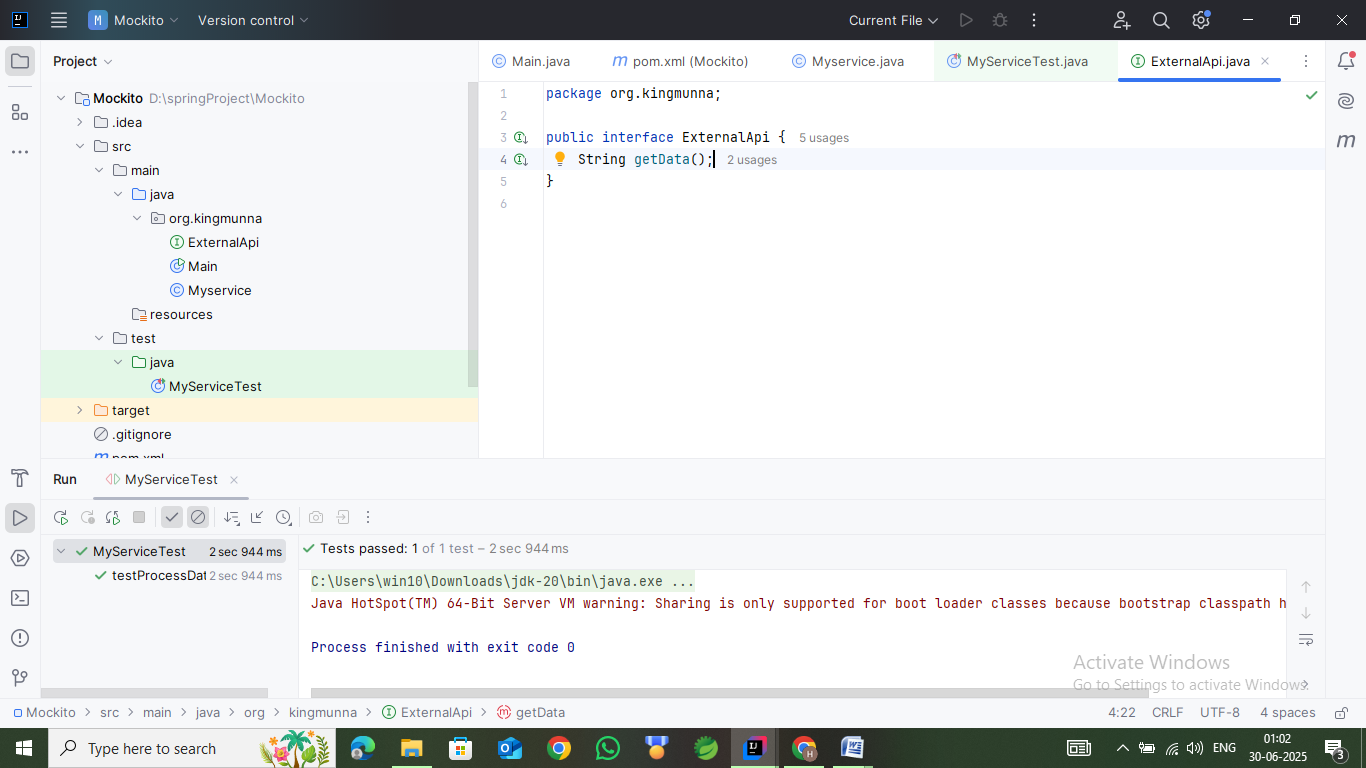
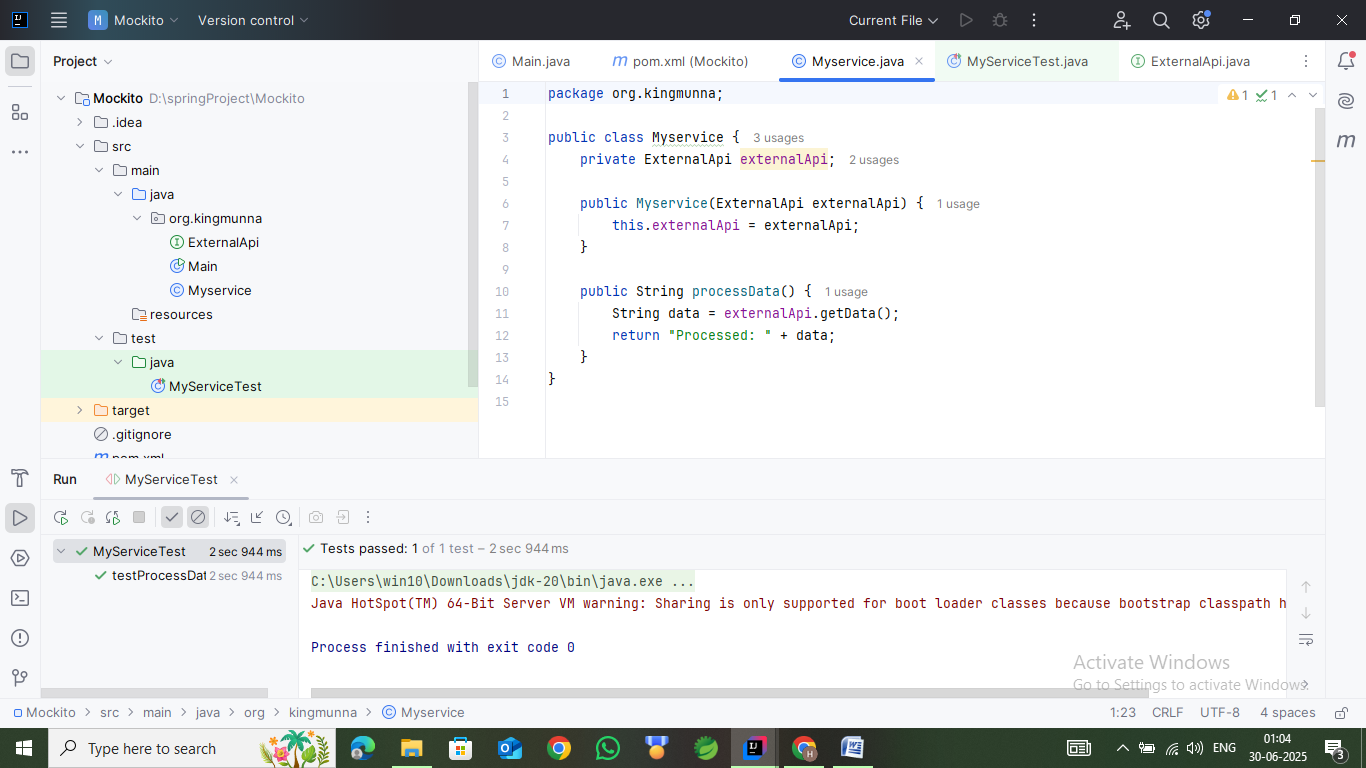
| ***Exercise 2: Verifying Interactions*** |
| --- |

### 

### **1. Understand the Purpose**

In this exercise, we are testing whether a particular method — in this case, getData() — was actually **called** when expected. This is important when we want to **validate the behavior** of our code, especially when the method being tested does not return a value or when we care more about the **effect** or **interaction** than the result itself.





### **2. Create a Mock Object**

We start by creating a **mock** of the external dependency. Since ExternalApi is an interface, Mockito can easily mock it.

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

Here, we are not creating a real object. Instead, we are creating a **test double** — a fake version that **records interactions** with it but doesn’t run any actual logic.

### **3. Inject the Mock into the Service**

We now inject this mocked dependency into the class we want to test — in this case, MyService.

MyService service = new MyService(mockApi);

This setup allows us to test MyService in **isolation**, without needing the actual implementation of ExternalApi. Any calls MyService makes to ExternalApi will go through our mock.

### **4. Call the Method Being Tested**

Next, we invoke the method on MyService that internally uses ExternalApi.

service.fetchData();

We assume that fetchData() internally calls mockApi.getData(). This is the **interaction** we want to verify. At this point, Mockito keeps track of whether or not getData() was actually invoked.

### 

### **5. Verify the Interaction**

Finally, we use verify() from Mockito to **assert that the method was indeed called**

verify(mockApi).getData();

This line checks:

* Was getData() called on the mockApi object?
* Was it called with the expected arguments (in this case, no arguments)?

If the call did not happen, or happened with different arguments, **the test will fail**.

### 