**Exercise 1: Mocking and Stubbing**

1. Understand the Scenario

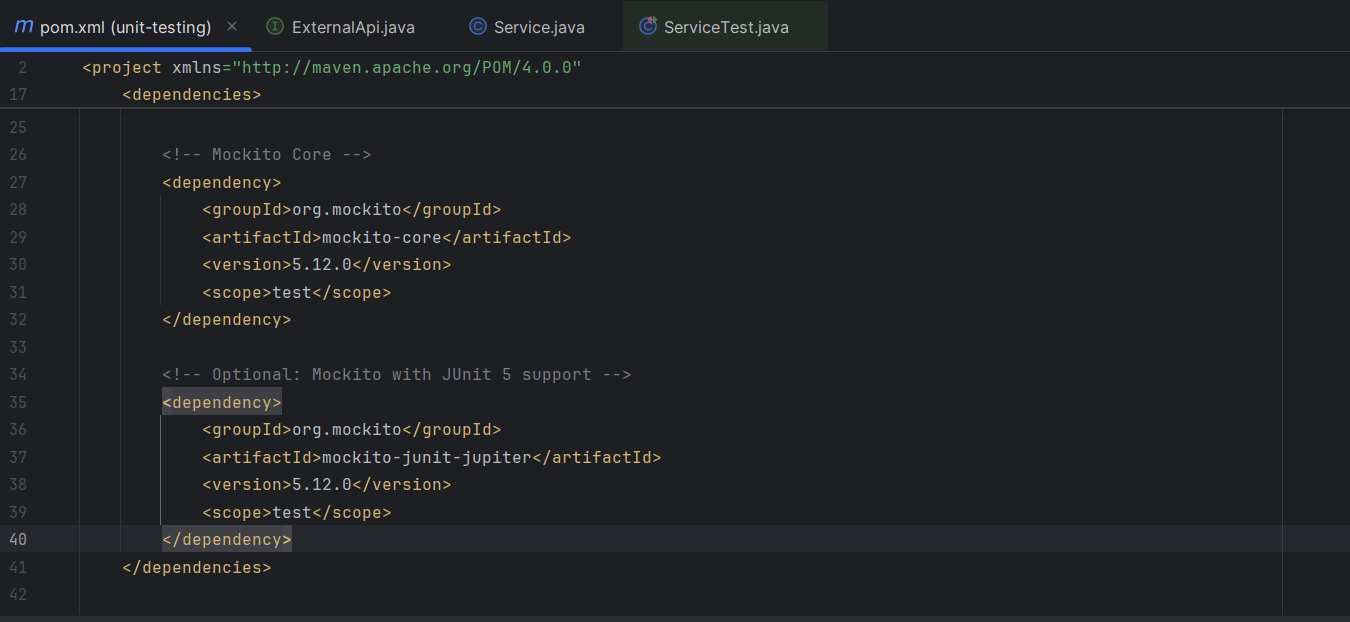
In real-world applications, a service class often relies on external systems like APIs, databases, or other services. During unit testing, calling these real systems is not ideal because:

* They may be slow or unavailable.
* They can return inconsistent data.
* They introduce dependencies not related to the code we are testing.

To solve this, we use Mockito to create mock objects—fake versions of the dependencies—so we can simulate their behavior and focus testing only on our service logic.

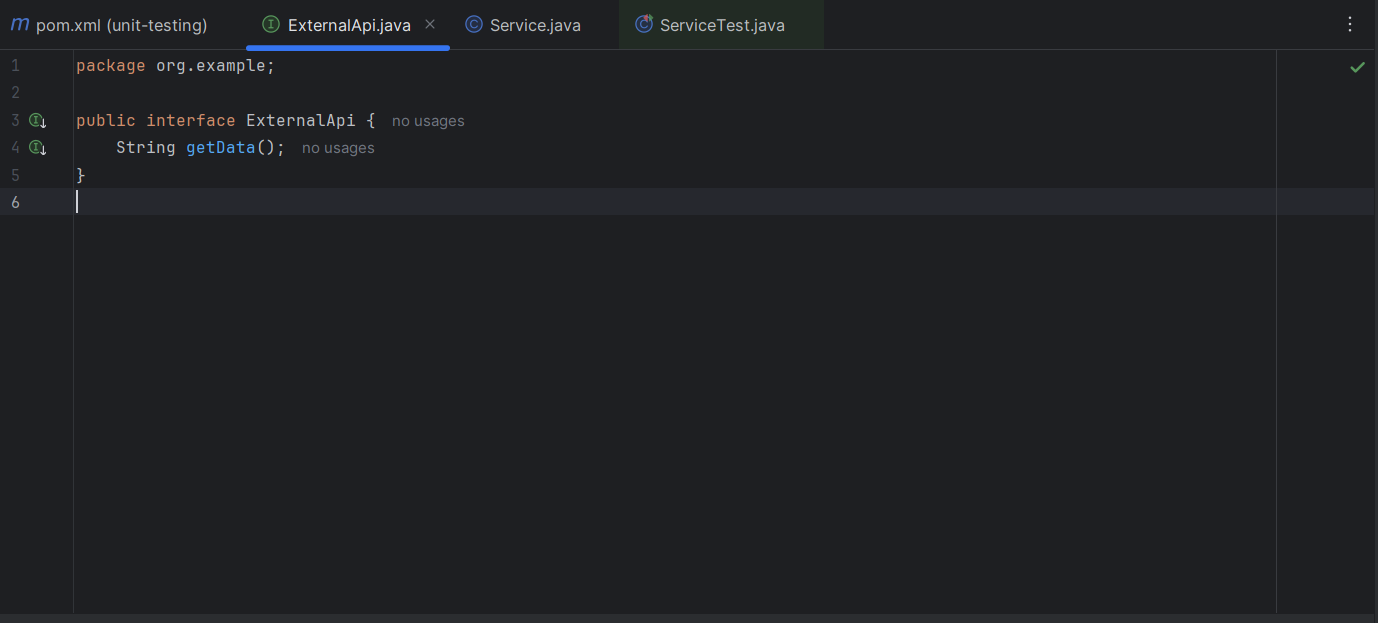
2. Add Mockito to the Project

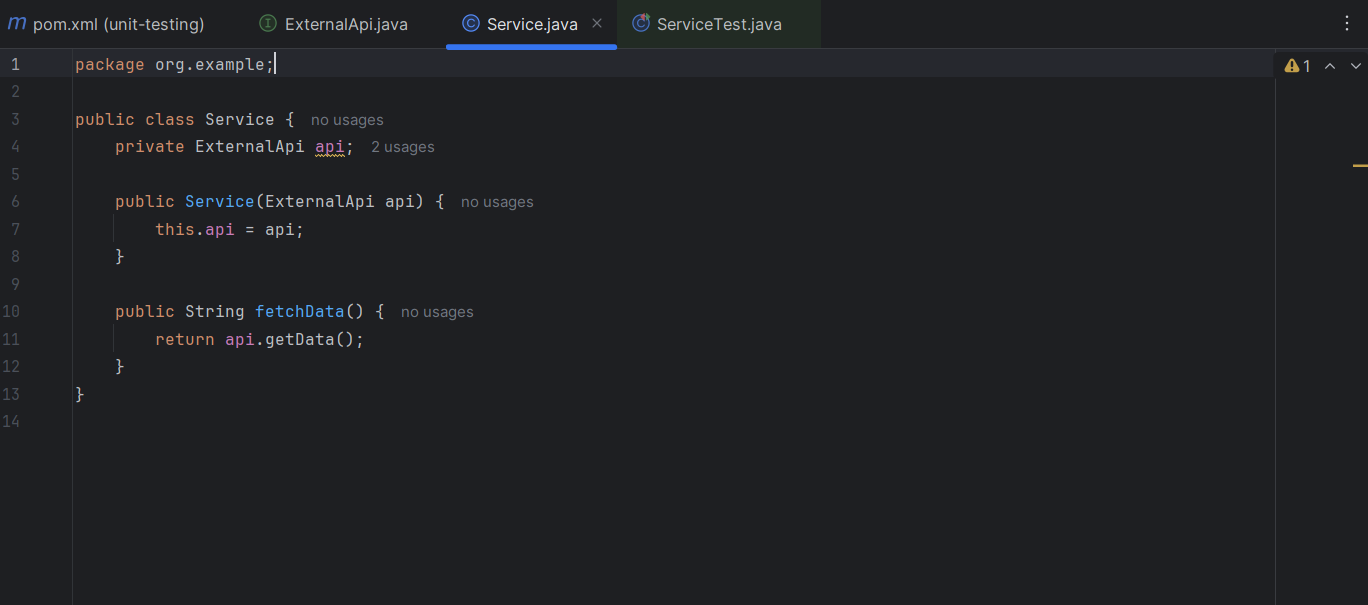
* Added dependency to pom.xml

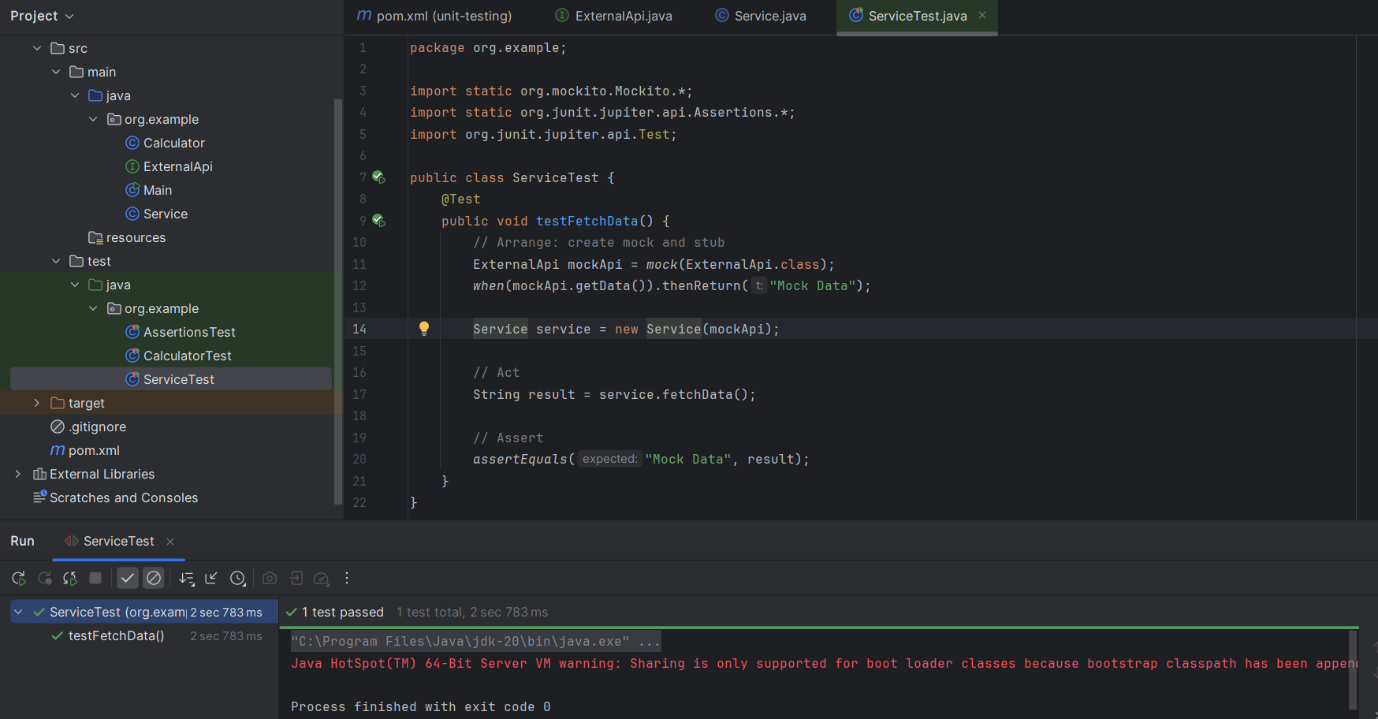
****

**Working Structure**

* Create a Mock Object
* Use mock() to create a dummy version of the external dependency.
* mockApi behaves like ExternalApi but has no real logic inside. We can define what it should return using stubbing.
* To control the behavior of the mock, we stub its methods using when(...).thenReturn(...). This sets up what the mock should return when called.
* This ensures that every time getData() is called on the mock, it will return "Mock Data".
* Inject the Mock into the Service
* we are testing only the logic inside Service, using controlled mock behavior, not real API calls.
* Call the Method and Assert the Result
* Now that the service has a mock dependency, we call the service method and verify its output using assertions
* This checks that the service **uses the mock correctly** and **returns the expected result**.

****

****

****