Mocking and Stubbing using Mockito

**🔷 What is Mocking?**

**Mocking** is a unit testing technique where we create **fake objects** that simulate the behavior of real objects. This is useful when:

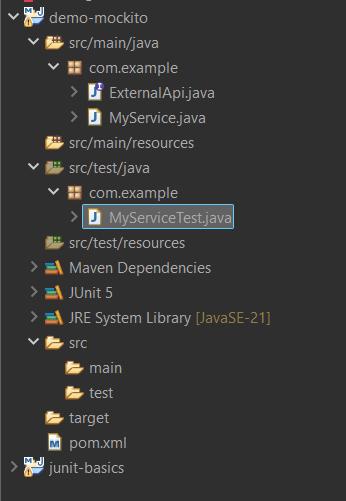
* A class depends on **external APIs**, **databases**, or **network services**
* We want to **isolate** the class under test
* We want **predictable outputs** without actual external calls

In Java, we use libraries like **Mockito** to create mock objects.

**🔷 Objective of This Exercise**

* Create a mock for an external API
* Stub the API method to return predefined data
* Inject the mock into a service class
* Verify the output using **JUnit 5**

**🔷 Project Structure**



**Depenedecies :**

**<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>com.example</groupId> <!-- You can change this -->**

**<artifactId>mockito-demo</artifactId> <!-- You can change this -->**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<!-- JUnit 5 -->**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter</artifactId>**

**<version>5.10.0</version>**

**<scope>test</scope>**

**</dependency>**

**<!-- Mockito -->**

**<dependency>**

**<groupId>org.mockito</groupId>**

**<artifactId>mockito-core</artifactId>**

**<version>5.11.0</version>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<plugins>**

**<!-- Compiler Plugin for Java 21 -->**

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-compiler-plugin</artifactId>**

**<version>3.11.0</version>**

**<configuration>**

**<source>21</source>**

**<target>21</target>**

**</configuration>**

**</plugin>**

**<!-- JUnit 5 Surefire Plugin -->**

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

**<artifactId>maven-surefire-plugin</artifactId>**

**<version>3.0.0</version>**

**<configuration>**

**<includes>**

**<include>\*\*/\*Test.java</include>**

**</includes>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

**🔷 Code Snippets**

**🔹 ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

📌 This is a simple interface simulating a third-party or external API.

**🔹 MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

📌 MyService depends on ExternalApi. Instead of calling a real API, we’ll inject a mock.

**🔹 MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Step 1: Create mock

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

// Step 2: Stub method

when(mockApi.getData()).thenReturn("Mock Data");

// Step 3: Inject mock into service

MyService service = new MyService(mockApi);

// Step 4: Verify result

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

📌 This test:

* Creates a mock for ExternalApi
* Stubs getData() to return "Mock Data"
* Verifies that MyService.fetchData() returns the expected value

**🔷 Screenshot: Output**

