**Hands On Lab 1**

**Using Mockito with JUnit**

With Mockito, creating mock objects is very easy. It provides many simple annotations to do so. Internally, a mock is nothing but a proxy for the actual class instance.

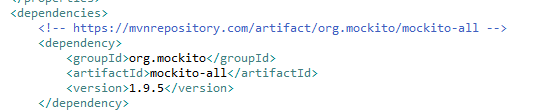
Follow these  steps to create Mockito JUnit project.

****Step 1:**** Create a simple java maven project.

### ****Maven Dependency****

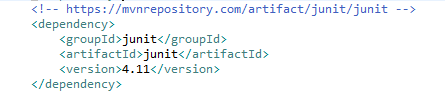
****Step 2:**** Add required dependencies to pom.xml

A Maven dependency is the fastest way to get started with Mockito:

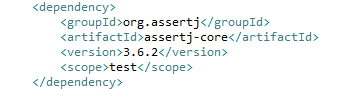


Just a single dependency which doesn’t bring any other libraries with it. See [here](https://mvnrepository.com/artifact/org.mockito/mockito-all" \t "https://java2blog.com/mockito-junit/_blank) for latest versions of the library.

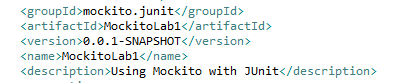
As we will also be using some JUnit functionality, we will also need it’s dependency. Let’s add it next,



Finally, we will also be using AssertJ.



Name your groupId and artifactId as follows:

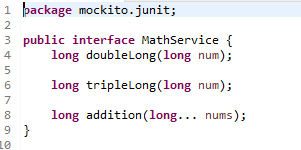


Don’t forget to specify the JDK version. Make sure you have set JDK 11 in the installed runtimes and set you compiler compliance level.

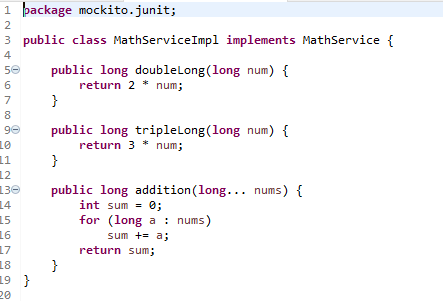


## ****Mockito and JUnit****

****Step 3:**** We will define an interface named as MathService. Here is our interface:



Next, we will be adding its implementation. These will be simple mathematical implementations:



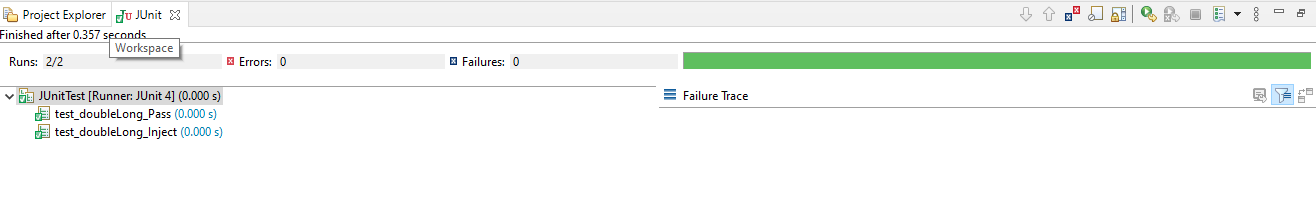
****Step 4:**** Let us focus on the steps we need to perform to get started with Mockito mocks:

1. Annotate the test class with @RunWith(MockitoJUnitRunner.class).
2. Annotate the test fields with either *@Mock* or *@Spy* annotation to have either a mock or spy object instantiated.
3. Annotate the system under test with @InjectMocks annotation.

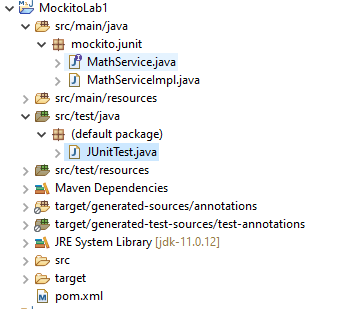
Let’s apply above three steps to provide our test demo.



Now, it is worth noting that when a Mock is used, the actual method implementation is not called. In the second test-case we wrote, the implementation of the method was actually called.  
When you run the above test, you will get this output:



The source code structure should look as follows:



\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*