**Hands On Lab 2**

**Configuring Test Code with Mockito Annotations**

## **1. Mockito Introduction**

During unit testing of the application, sometimes it is not possible to replicate exact production environment. Sometimes database is not available and sometimes network access is not allowed. There can be many more such restrictions. To deal with such limitations, we have to create mocks for these unavailable resources.

Mockito is an open source framework that allows you to easily create test doubles (mocks). Test Double is a generic term for any case where you replace a production object for testing purposes.

In Mockito, we generally work with following kind of test doubles.

* **Stubs** – is an object that has predefined return values to method executions made during the test.
* **Spies** – are objects that are similar to stubs, but they additionally record how they were executed.
* **Mocks** – are objects that have return values to method executions made during the test and has recorded expectations of these executions. Mocks can throw an exception if they receive a call they don’t expect and are checked during verification to ensure they got all the calls they were expecting.

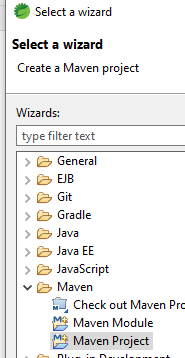
We can mock both interfaces and classes in the test class. Mockito also helps to produce minimum boilerplate code while using mockito annotations.

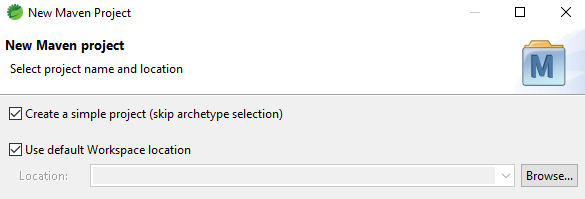
## **2. Mockito Setup**

To add mockito into the project, we can add the desired mockito version by any means i.e. [maven](https://mvnrepository.com/artifact/org.mockito/mockito-core), gradle or [jar file](http://central.maven.org/maven2/org/mockito/mockito-core/2.23.4/mockito-core-2.23.4.jar).

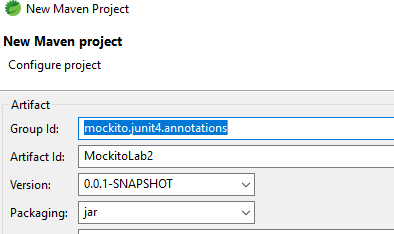
Create a simple Maven project.

Skip archetype selection.

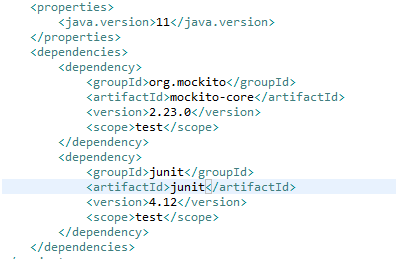




Enter in the groupId and artifactId:



Edit the pom.xml to include the proper JDK version, Mockito and JUnit dependencies.



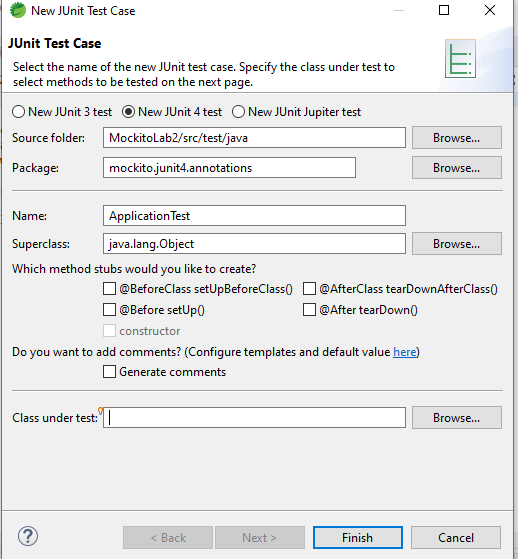
## **3. Mockito Annotations**

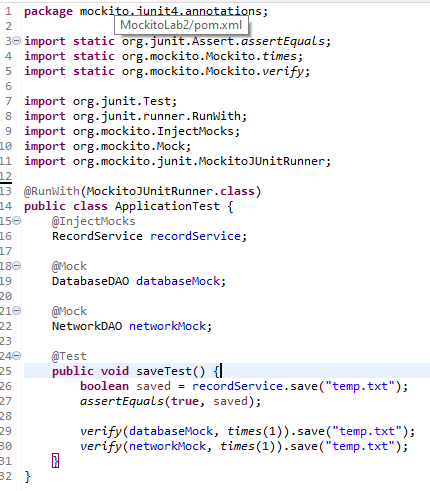
Before hitting the keyboard to write application and unit tests, let’s quickly overview the useful mockito annotations.

* **@Mock** is used for mock creation. It makes the test class more readable.
* **@Spy** is used to create a spy instance. We can use it instead of the spy(Object) method.
* **@InjectMocks** is used to instantiate the tested object automatically and inject all the *@Mock* or *@Spy* annotated field dependencies into it (if applicable).
* **@Captor** is used to create an argument captor

To process all the above annotations,  **MockitoAnnotations.initMocks(testClass);** must be used mat least once. To process annotations, we can use the built-in runner **MockitoJUnitRunner** or rule **MockitoRule**. We can also explicitly invoke initMocks() method in @Before annotated Junit method.

Under src/test/java package, create the following test class:

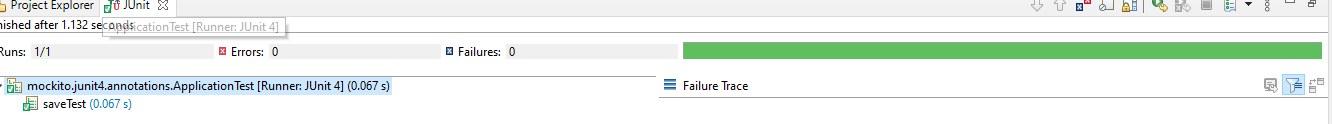




Right Click on your test class and run as a JUnit Test :



The test should run successfully:



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