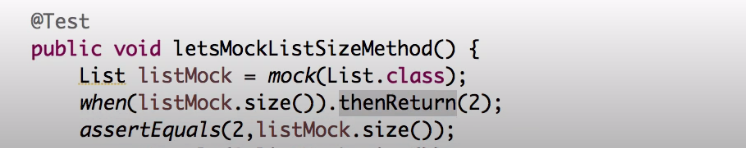
**Junit by 28minute:**

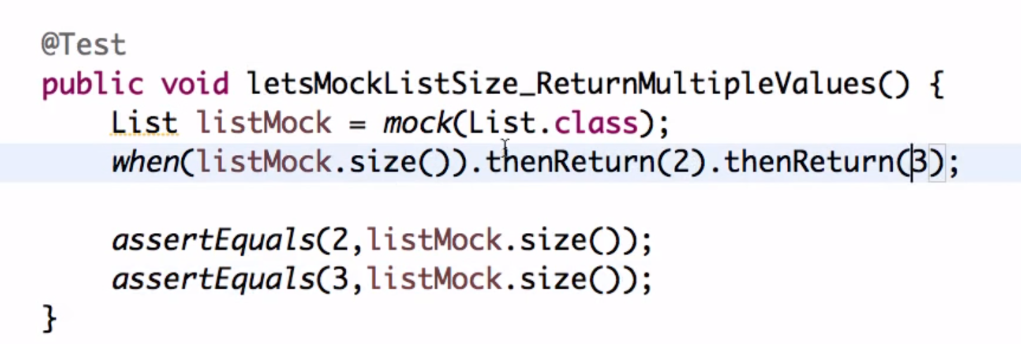
* **SUT:** System Under Test, this is the class for which we are going to write test.
* **Dependency:** This is class/interface which methods are getting used in SUT. Even though we are calling some methods of dependency class/interfaces, there might be a chance that these classes/Interfaces are from 3rd party and we don’t know what is there in their implementation.
* For the methods and interfaces which don’t have implementation, we can create stub for them to do unit testing.
* This stub class will be implementing method of our interface which is kind a dependency of SUT.
* **Stubs:** Dummy implementations of interfaces.
* **Mock:** Mocking is creating Objects that simulate the behavior of real objects. Unlike stubs, mock can be dynamically created from code at Runtime.

Mock provide more functionality than stubbing. We can verify method call using mock.

* We can mock predefined interfaces/classes given by Java as well. For example, mock List interface:

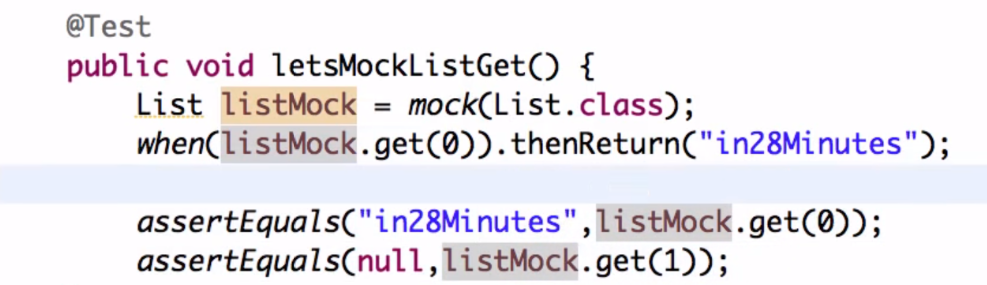


* We can write a method in such a way that it returns multiple values as below:



Here first time once listMock.size() is getting called it will be returned 2 and next time it is returning 3 so out assertEquals will work in same way. This way a single method can return multiple values.

* If we call a non-stubbed method which means a method which we haven’t handled it will return null for that as below:



* Argument matcher: If we want to return same values irrespective of parameter values that been passed to any method then we use the concept of argument matcher. Like if parameter type is integer then we can use anyInt(), similar way we can write of other data type as well. anyInt() is a static method from Mockito framework.



* Mockito for exception handling, in below example we are explicitly throwing an exception which is getting caught by our jUnit framework as we wrote condition in @Test() as below: