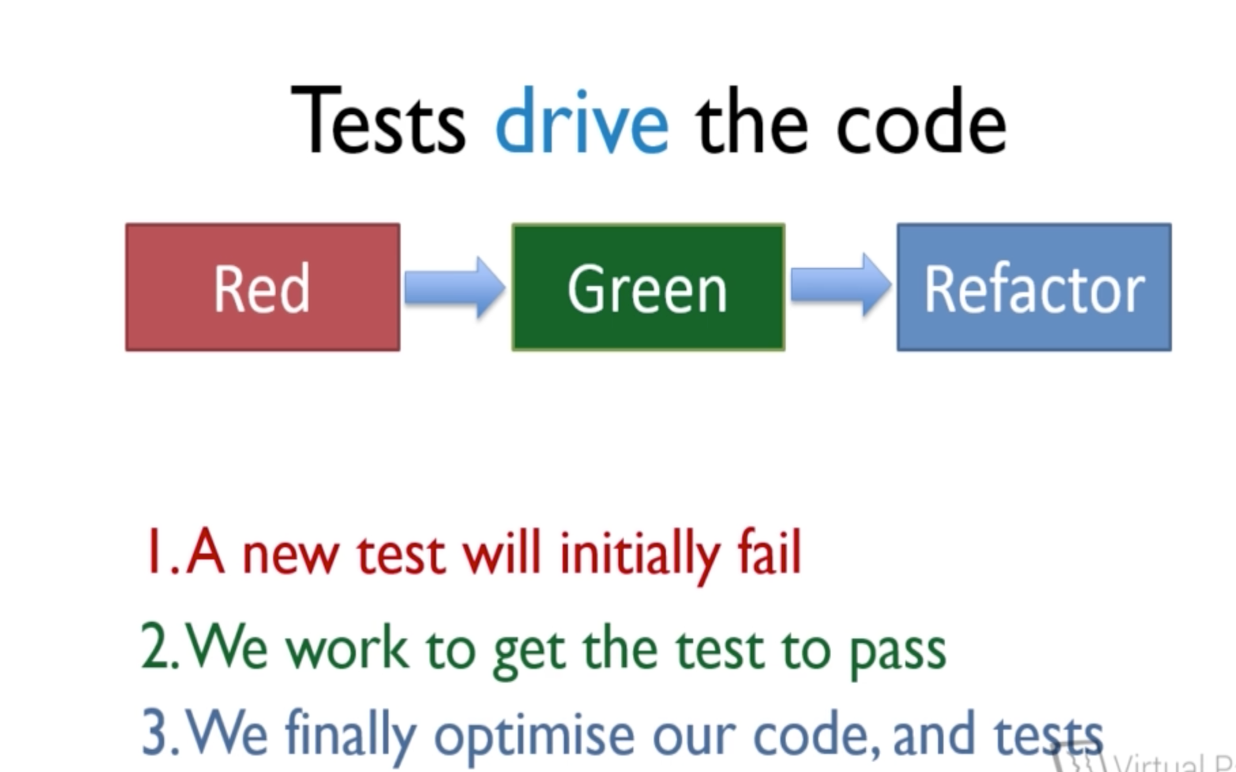
**Junit – Udemy**

* **Introduction:**

**Clean code that works.**

**Tests drive the code.**

****

* **What to test?**

Each test will cover a single piece of logic.

Each test will cover a single scenario for a single piece of logic.

* **What we are going to test?**

Project – I – ISBN Validation

ISBN – International Standard Book Number

Valid and Invalid ISBN:



* **Environment Setup:**

1. Create a new project in Eclipse with name “ISBNValidator”
2. Create a new class inside the source folder with name as “ValidateISBN” and package name as “ com.virtualpairprogrammers.isbntools”.
3. Create new source folder named “test” to keep all our test cases.
4. Create a new test class with name as “ValidateISBNTest” and the package name as “com.virutalpairprogrammers.isbntools” within “test” folder.

* **Rules of TDD:**

1. Test the expected outcome of an example. Not how its implemented, design, architecture etc.
2. Don’t pre-judge design… let your tests drive it.
3. Write the minimum code required to get your tests to pass.
4. Each test should validate one single piece of logic. (Which mean our test should contain only one assert). If it has more than one assert, then it must be of the below scenario.

If you are setting many variables in the class and if you want to test it, then we can go for multiple assert in a test.

If you are checking more than one valid scenario, then we can have multiple assert statement in a test. Example, in our test we are check two different valid ISBN.

* **Getting Red:**

Always keep in mind the workflow of a unit test.

Red -> Green -> Refactor

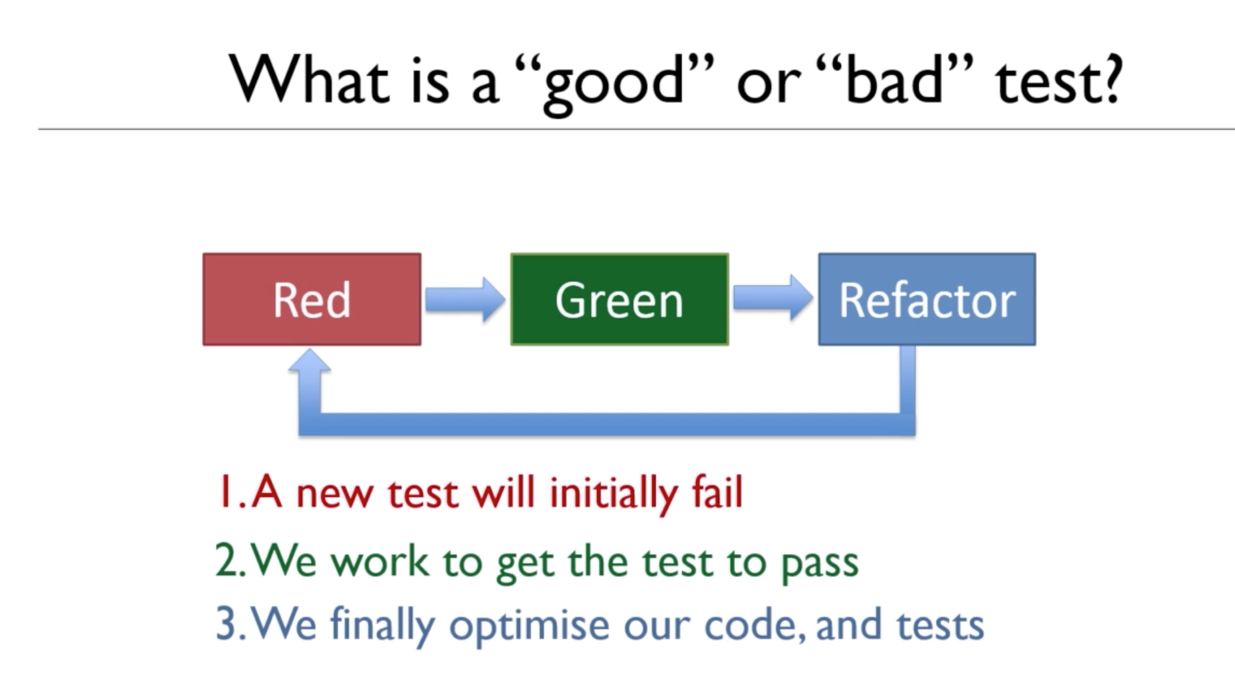
Which means, fail -> make the test pass -> refactor.

1. Change the test name from test to checkAValidISBN and call the fail method as it doesn’t have the implementation.
2. Now we got the Red in the workflow. Next steps are to get the Green in the workflow.
3. Now change the checkAValidISBN(), it has to take the ISBN as an input.
4. Create an instance for ValidateISBN and then call the method which checks the validation.
5. The method has to return boolean.
6. Create a method in the ValidateISBN class as per the signature.
7. Add a assertTrue statement to check the result.
8. Rerun the test again it will fail as we are returning false as a result by default.

* **Step\_02: Getting Green:**

1. To get the green, just return true always for the checkISBN method. This is not the rite approach, we will refactor it once again.

* **What is a good or bad test?**



* **Rules of testing:**

Test one item of functionality only.

Test business logic, not methods.

Tests must be repeatable and consistent.

Tests must be thorough.

* **What tests should I write?**

What should the logic be?

What is the opposite to that logic?

Are there any edge cases?

Are there any error conditions?

* **Junit Asserts:**

@Test

public void valueIsNotNull(){

MySystem mySystem = new MySystem();

String value = mySystem.getValue();

assertNotNull(value);

}

assertThat():

@Test

public void isAValidEmailAddress(){

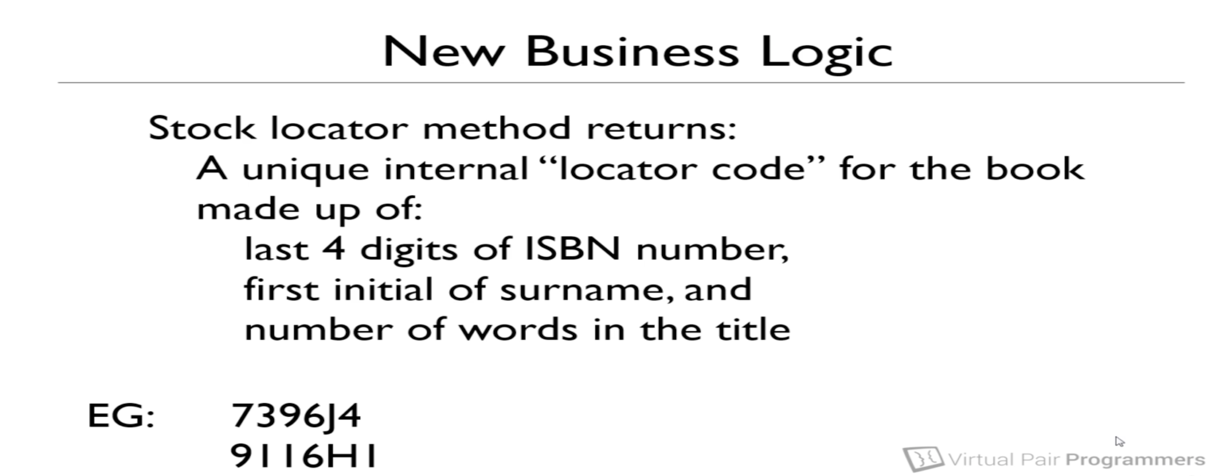
MySystem mySystem = new MySystem();

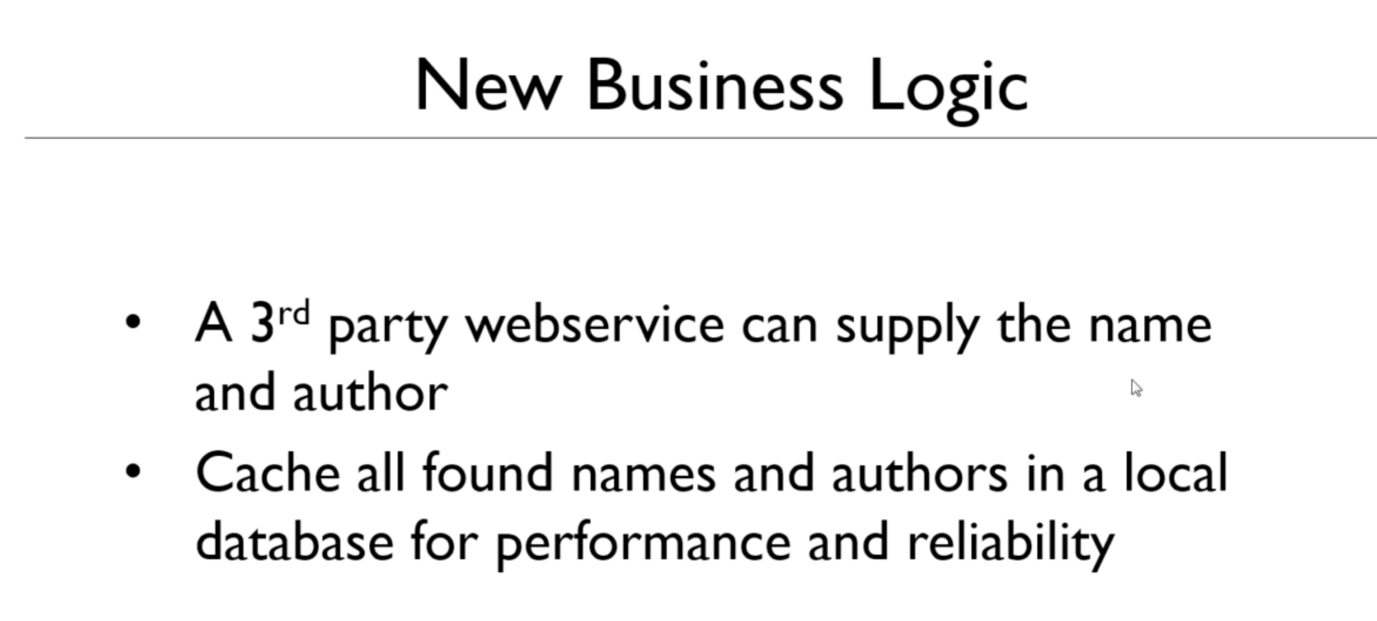
String email = mySystem.getEmail();

assertThat(email, StringContains.containsString(“@”));

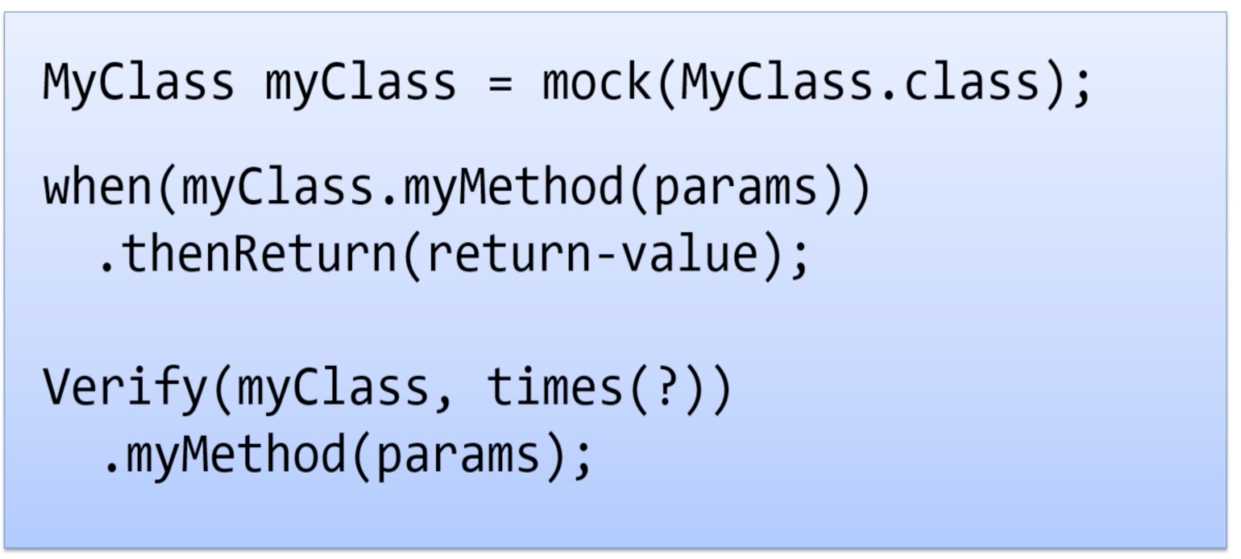
}

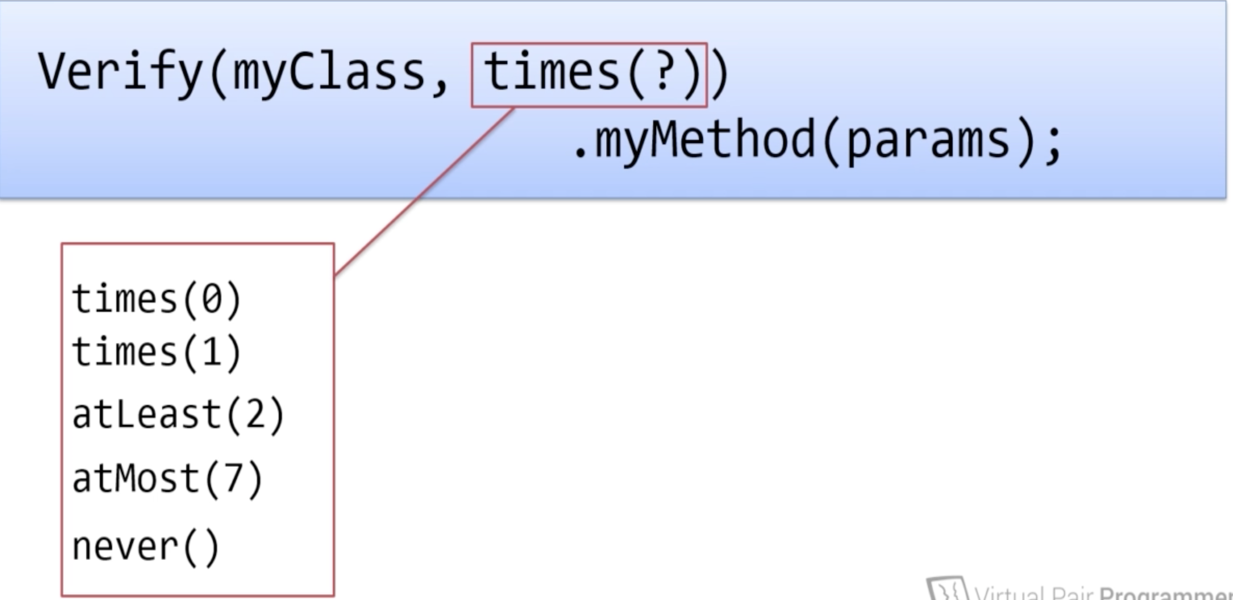
* **Stubs:**

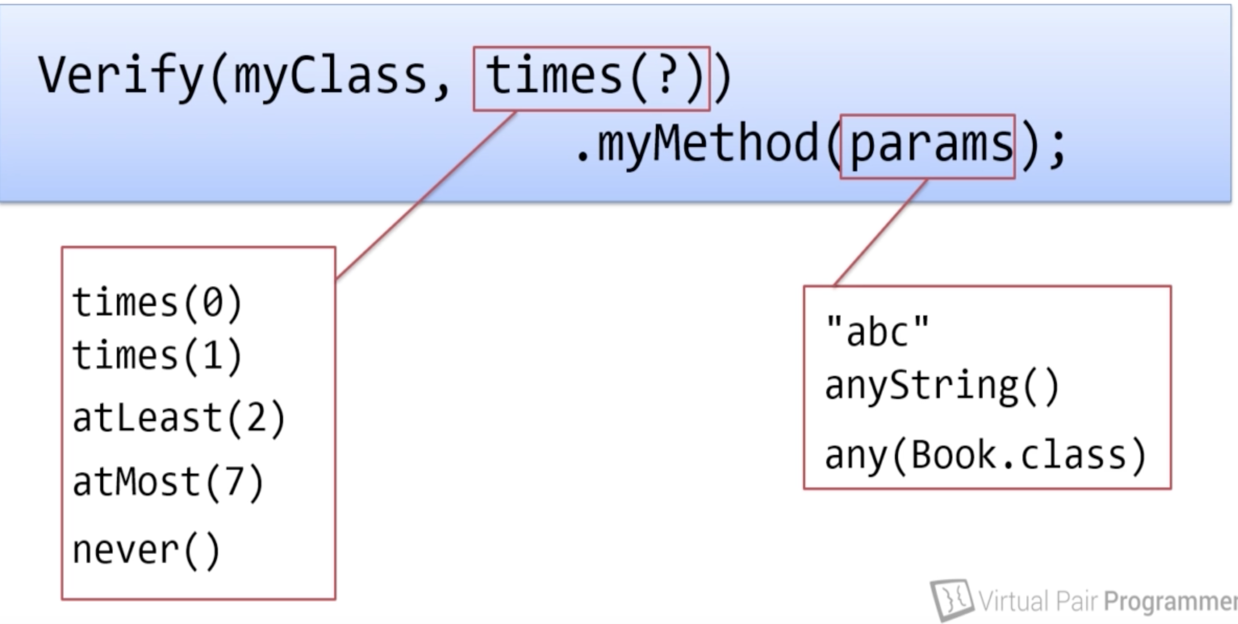


****

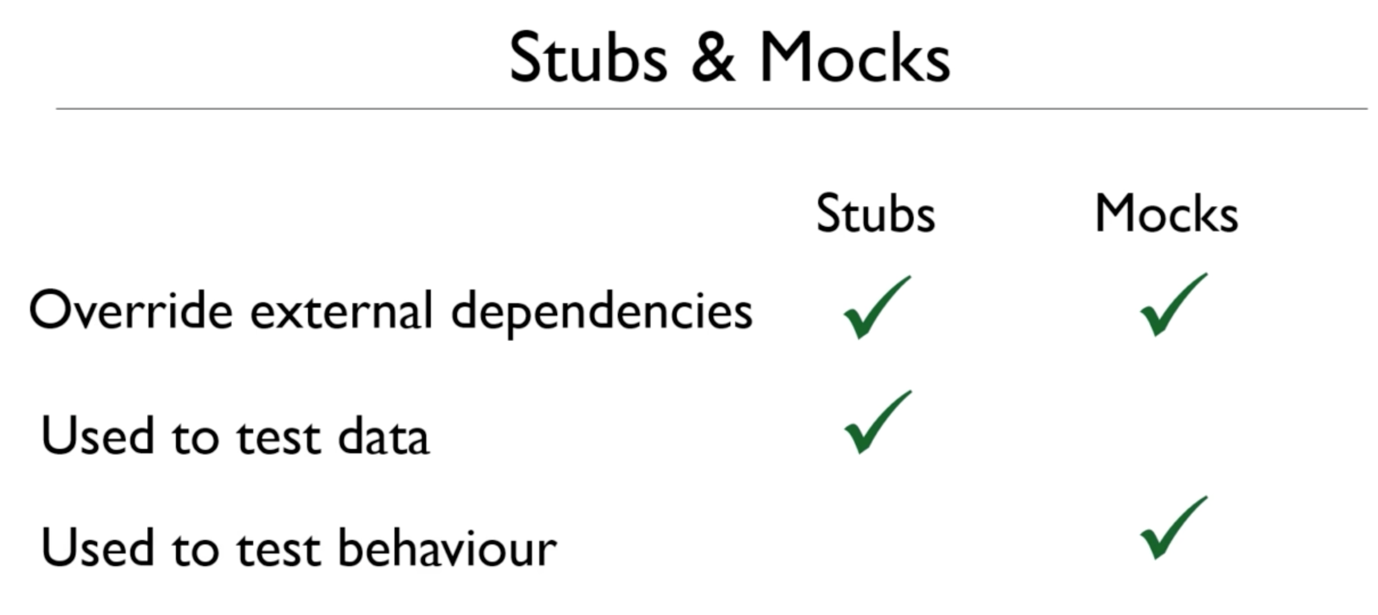
* **Mockito Methods:**

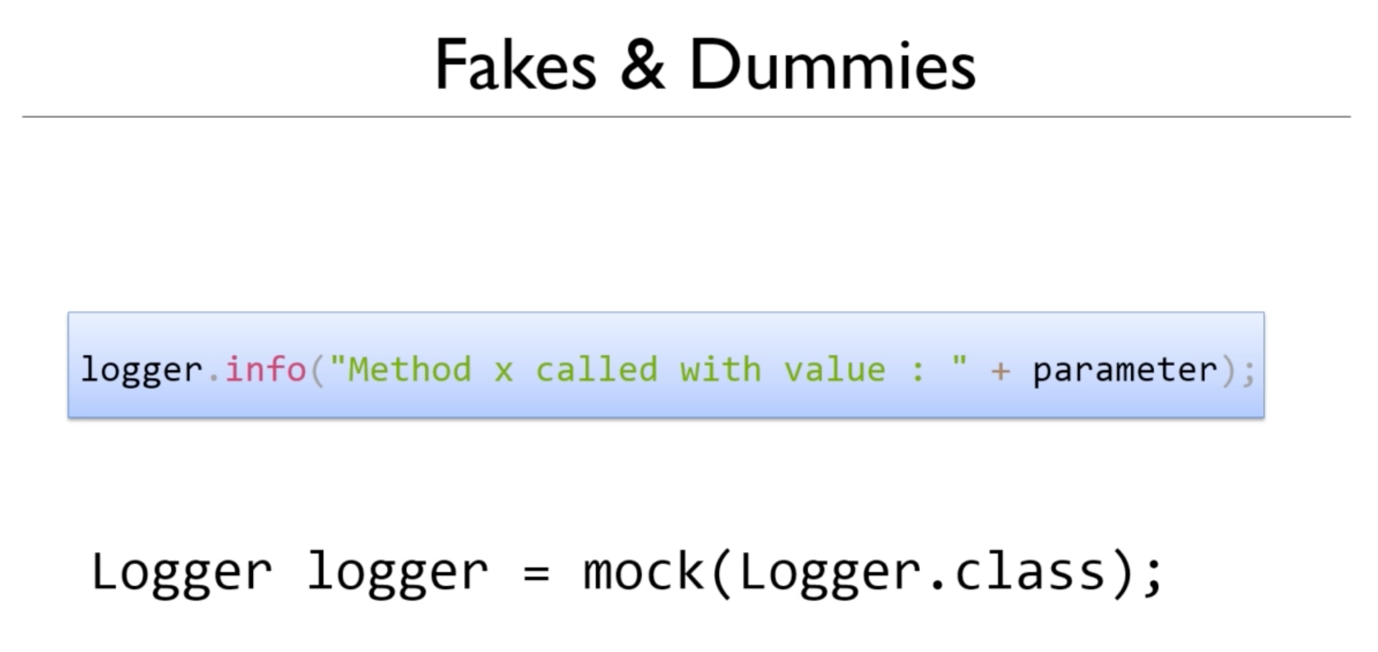




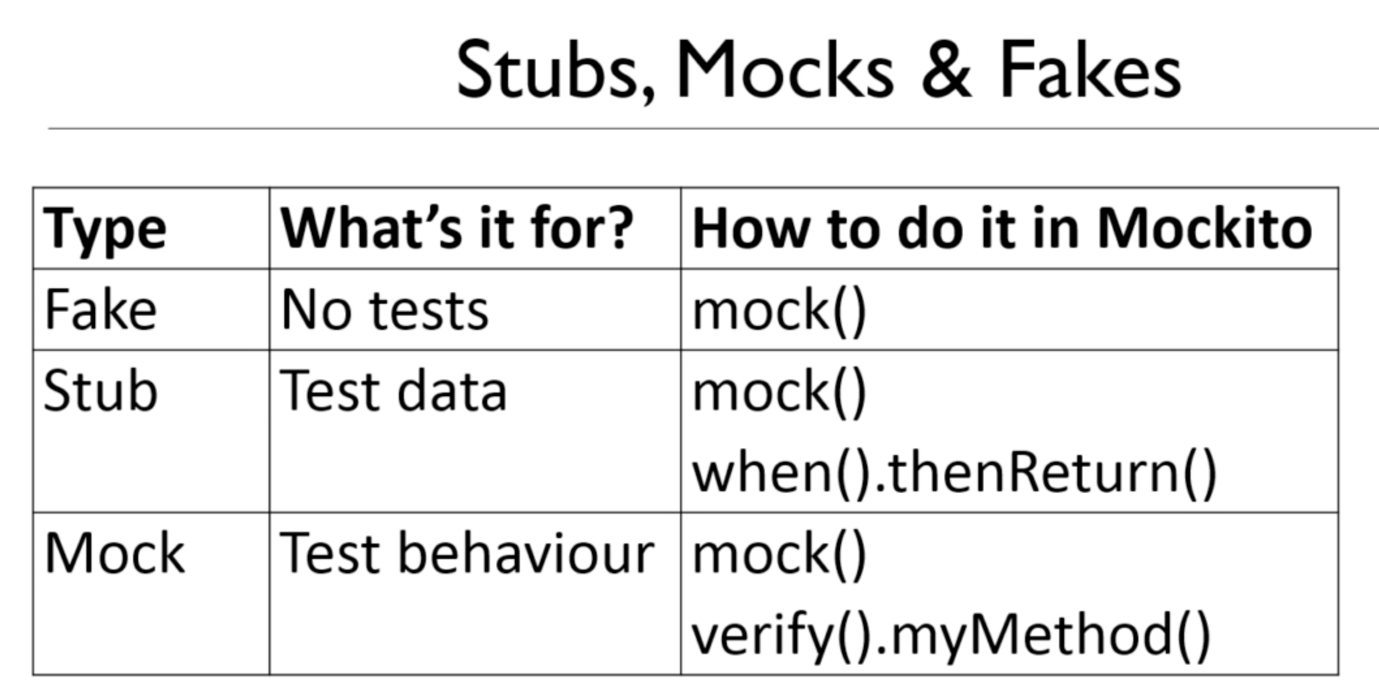


* **What are fakes and dummy?**

****

****

**Note: We have to use the Fake objects wherever we are not bothered about the implementation.**

****

* **setup() and tearDown() methods:**

In Junit every test methods runs on independent, in the example we are using, we are calling the mock and the stub in all the test classes which is a duplicate code, we can overcome this problem by putting those calls in a single method.

The methods are setUp() and tearDown()

These methods will be called before and after running each test case.

These methods act like a constructor and destructor for the test class.

Move all the commonly used code to the setup method, so that all the duplicate codes would be at single place.

* **How to avoid tautologies:**

**Tautology: repetition, reiteration, duplication etc.**

If the class method and the test class method contain the same logic, then it’s not a correct test.

The main problem with this kind of test is that, if there is a mistake in one place that will affect in the other place too. That is, if the implementation class is wrong that affects the test and if the test is wrong that affects the implementation.

Tautology in a single point, if you do any calculation or logic in your test, then definitely it will lead to tautology.