Testing with Dependencies: Parts I, II, III, & IV

Part I

- The more mocks we use, the poorer our design is.
    - Use mocks selectively.
- Unit testing is fairly easy if there are no dependencies.

What's a mock?

- For credit card, the charging company gets a private key + a test key.
    - You can use the test key with a proxy server to simulate real transactions/errors etc.
    - A credit card with all 42's is considered a test credit card. With the test card & test, you can do testing etc.
    - A credit card with all 42's with a 1 at the end is considered an invalid credit card (expired eg). So you can test those conditions also.
- Fake → You talk to an object/class/instance like you'd talk to a real one, and you'd be getting fake responses. Your code won't have a clue if it's talking to the real service or the fake service. Eg. In-memory database.
- Software should be designed from the testability point of view; testing should be made simple. Don't rush to write code.
- **A mock object is an object that stands in for a real object.**

Stub vs. Mock

- **Fake** → a service that takes a shortcut and may provide you with a fake response and may not be used in real production.

    - In case you don’t have internet (if you are in an airplane eg), you can’t use this, as this works in the same way as a production code.

- **Stub** → stands in for another object, but simply returns responses you key in. You cook up some responses and that’s what you get. Used to test the state of an object after an interaction.
- **Mock** → will tattle tale. It keeps specific interaction between the code being tested and the code you depend on (replaced by mock). Useful for behaviour and interaction.
- **Spy** → stands in for a real object and lets you talk to the real object, except takes away some methods of the real object. It acts as a proxy in front of an object you depend on — it lets some calls pass through and provides fake responses for others.

What can mocks do for you?

- Useful for simulating the behaviour of a code.
- Ill-behaviour: simulating for when things go wrong.

Part II

Ways to inject a mock object

1. Constructor based
    a. We pass the mock object as a value to the constructor.
    b. Relatively static and inflexible. Hard to change the value after constructing.
    c. Hard to pass multiple values — will have to pass as a list.
2. Setter based
    a. Use a setter to pass the mock object.
    b. Easy to vary dynamically.
3. Factory based

    a. Within your class, you'd go out to a 'factory' object to get it. Eg. `serviceFactory.getService()`. This could return a mock object.
    b. Adds a bit more complexity.
4. Overriding

Part III

Don't grow mocks

- If you have to write unit tests for your mocks, your mock is too complex.

Keep mock under tight leash

- Do not increase the complexity of mocks.
- Each test can have a small, little mock object for itself.
    - The mock should just return what you need; there's no need of all the things that the class/interface offers.
- A mock *is not* a representative of an object; it is standing in only for that very narrow window of interaction between the code and the dependent object enough to get that test to pass.
- Mockito is a tool used to create a mock for Java.

Part IV

To mock or not to mock?

- The purpose of mocking is to make the test easier to write and faster to run.
    - If you can do the above without the use of mocks, no need for mocks.
- Use mocks very reluctantly.
- The fewer dependencies you have, the fewer mocks you need.

- Use mocks as an exception rather than a norm.
- A lot of mocks is a sign of poor design.
- Don’t mock code that is fast, predictable, easy to work with, or that you can easily set up.
- Mock if things are slow, or unpredictable, or you want to test stuff that isn’t easy to set up (failure etc.), or if you want to test ill-behaviour (when the service isn’t working properly).

What to mock?

- Mock code that you’re dependent on *directly*.
- Mock only if it is going to speed up testing and make it more reliable and dependable.

When not to mock?

- Do not mock multiple levels below.

Knock out before your mock out

- Removing dependencies is better than to mock.