Test Doubles

- What?
- Why?
- Types
- Some strategy
- Guidelines
- Exercise

Test Doubles: What?

If you have read the “gang of four” book *Design Patterns*, one of the guiding principles mentioned in the introduction is “program to an interface, not an implementation”. One advantage is that it opens the doors for using *Test Doubles*.

A *Test Double* is any kind of object used in place of a real object for testing purposes

Test Doubles: Why?

- They became very popular at the beginning as an instrument for isolating tests from third-party libraries.
- They guide the composition of a coherent, well-encapsulated system of types within a code base, by focusing on behavior rather than data.
- They lead developers to think about object interactions early, identifying public interfaces first.

Test Doubles: Types

- **Dummy objects:** Only needed to complete the parameters' list of a method, but never actually used.
- **Stubs:** Stubs respond to calls made during the test with some pre-programmed output specific for each test. Used to provide synthetic replacement for queries
- **Fake objects:** Fake objects are handmade Stubs (also known as "poor man Stubs")
- **Mocks:** Mocks are set up with expectations of the calls they are expected to receive. They can throw an exception if they receive a call they don't expect and/or are queried during the assertion phase to verify they received all the calls they were expecting. Used to provide a way of confirming Commands
- **Spies:** Handmade Mocks (also known as "poor man Mocks") **

CQS or Command-Query Separation is a principle that states that a software interface abstraction is designed with 2 types of methods, one known as **Command method**, and the other known as **Query method.**

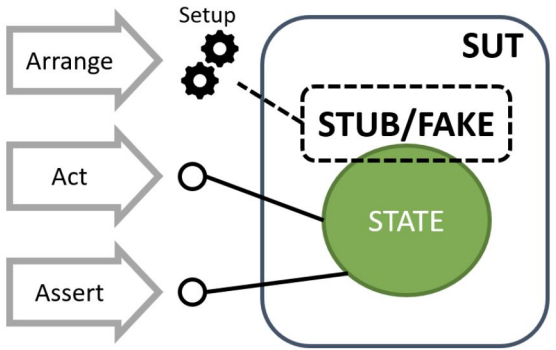
- A Command method modifies or mutates the state underneath the interface, but does not answer any portion of state
- A Query method answers the current state beneath the interface, but must not modify that state before answering it.

Keeping in mind this principle, we can describe a usage pattern for Test Doubles.

Queries

A query is a method call that returns data and shouldn't have any effect on the state of an object (should be idempotent)

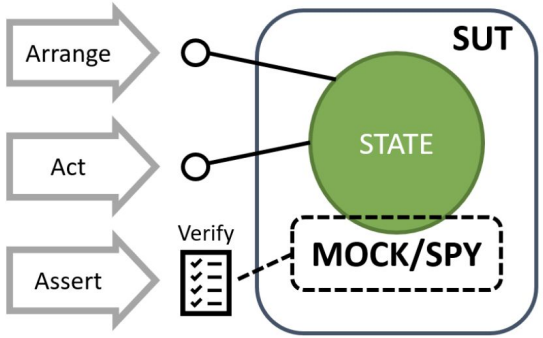
Use **Stubs** for queries in the Arrange section



Commands

A command is a method call that changes the state of the system; hence, it has a side effect, and it should return void.

Use **Mocks** for Commands in the Assert section



Test Doubles: Guidelines

- Only use Test Doubles for classes that you own
- Verify as little as possible in a test
- Don’t use Test Doubles for isolated objects
- Don’t add behavior in Test Doubles
- Only use Test Doubles for your immediate neighbors
- (Avoid) Too many Test Doubles
- (Be aware of) CQS Principle trade-offs

Note: Be aware that when you mock 3rd parties (via a client) that you also ensure the integration of the 3rd party, as mocks ‘assume’ the 3rd party work as expected.

Bank Kata

Instructions: Write a class named Account that implements the following public interface

```
public interface AccountService{
    void deposit(int amount)
        void withdraw(int amount)
        void printStatement()
    }
```

Rules: You cannot change the public interface of this class.
Desired Behavior: Here's a specification for an acceptance test

Given a client makes a deposit of 1000 on 10-01-2012
And a deposit of 2000 on 13-01-2012
And a withdrawal of 500 on 14-01-2012
When they print their bank statement
Then they would see:

```
Date       || Amount || Balance
14/01/2012 || -500   || 2500
13/01/2012 || 2000   || 3000
10/01/2012 || 1000   || 1000
```