おはよおう ございます

GOOD MORNING!

REAL WORLD MOCKING IN SWIFT

THE PROBLEM

YOU WANT TO WRITE TESTS

BUT YOU DON'T WANT THEM TO MESS UP YOUR REAL STUFF

OR BE SLOW

```
let session = NSURLSession()
let url = NSURL(string: "http://masilotti.com")!
let task = session.dataTaskWithURL(url) { (data, _, _) -> Void in
    if let data = data {
        let string = String(data: data, encoding: NSUTF8StringEncoding)
        print(string)
    }
}
task.resume()
```

OCMOCK CAN'T SAVE YOU NOW

A MOCK APPEARS

```
class HTTPClientTests: XCTestCase {
    var subject: HTTPClient!
    let session = MockURLSession()

    override func setUp() {
        super.setUp()
        subject = HTTPClient(session: session)
    }

    func test_GET_RequestsTheURL() {
        let url = NSURL(string: "http://masilotti.com")!

        subject.get(url) { (_, _) -> Void in }

        XCTAssert(session.lastURL === url)
    }
}
```

SOME HARD TRUTHS

IT WILL TAKE A LOT OF

TIME

YOU WILL WONDER IF IT'S

WORTH IT

WHY USE MOCKS

» Make tests faster (like 1000s of times faster!)

» Increase coverage of test suite

» Make tests more robust

TESTING

THE EFFECTIVE ENGINEER

THE TIME TO WRITE TESTS IS


NOW

DEPENDENCY INJECTION

“DEPENDENCY INJECTION MEANS GIVING AN OBJECT ITS INSTANCE VARIABLES. REALLY. THAT'S IT.”

James Shore

WHY NOT JUST USE A SINGLETON?

WHY USE DEPENDENCY INJECTION

» Clear declaration of dependencies

» Easy customization

» Clear ownership

» Testability

FORMS OF DEPENDENCY INJECTION

» Constructor injection

» Property injection

» Method injection

» Ambient context

TEST DOUBLES

TYPES OF TEST DOUBLES

» Stubs

» Mocks

» Partial mocks


STUBS

“FAKES A RESPONSE TO METHOD CALLS OF AN OBJECT”

Unit Testing Tutorial: Mocking Objects

```
struct StubTimeMachineAPI: TimeMachineAPI {
      var videoUrl = "https://www.youtube.com/watch?v=SQ8aRKG9660"

    func getVideoFor(year: Int) -> String {
        return videoUrl
    }
}
```


MOCKS

“LET YOU CHECK IF A METHOD CALL IS PERFORMED OR IF A PROPERTY IS SET”

Unit Testing Tutorial: Mocking Objects

“WHEN SOMETHING HAPPENS IN YOUR APP”

Unit Testing Tutorial: Mocking Objects

```
struct MockTimeMachine: TimeMachine {
    var timeTravelWasCalled = false

    mutating func timeTravelTo(year: Int) {
        timeTravelWasCalled = true
    }
}
```

PARTIAL MOCK

“ANY ACTUAL OBJECT WHICH HAS BEEN WRAPPED OR CHANGED ”

Justin Searls

“TO PROVIDE ARTIFICIAL RESPONSES TO SOME METHODS BUT NOT OTHERS”

Justin Searls

ATTENTION:
PARTIAL MOCKS ARE AN
ANTI-PATTERN

WHAT'S WRONG WITH PARTIAL MOCKS?

» Challenging to set up: they require instantiating a real thing, then altering or wrapping it, then providing it

» Decreases the comprehensibility of the test

WHAT'S REAL?
WHAT'S FAKE?

MOCKING

A BEST PRACTICE EMERGES

MOCKING IN SWIFT VIA PROTOCOLS

“THIS KIND OF TESTING IS REALLY SIMILAR TO WHAT YOU GET WITH MOCKS, BUT IT’S SO MUCH BETTER.”

Protocol-Oriented Programming in Swift

“MOCKS ARE INHERENTLY FRAGILE. YOU HAVE TO COUPLE YOUR TESTING CODE”

Protocol-Oriented Programming in Swift

“TO THE IMPLEMENTATION DETAILS OF THE CODE UNDER TEST.”

Protocol-Oriented Programming in Swift

PLAYS WELL WITH STRUCTS AND CLASSES

PROTOCOLS HELP WHEN THERE'S INTERNAL DEPENDENCIES

MORE BEST PRACTICES

THEY SAID DON'T MOCK TYPES YOU DON'T OWN

4.1 Only Mock Types You Own

Mock Objects is a design technique so programmers should only write mocks for types that they can change. Otherwise they cannot change the design to respond to requirements that arise from the process. Programmers should not write mocks for fixed types, such as those defined by the runtime or external libraries. Instead they should write thin wrappers to implement the application abstractions in terms of the underlying infrastructure. Those wrappers will have been defined as part of a need-driven test.

We have found this to be a powerful insight to help programmers understand the technique. It restores the pre-eminence of the design in the use of Mock Objects, which has often been overshadowed by its use for testing interactions with third-party libraries.

WHY IT'S BAD TO MOCK TYPES YOU DON'T OWN

- Have to be sure that the behavior you implement in a mock matches the external library
- The external library could change, breaking your mock

MOCKING APPLE FRAMEWORK CLASSES

```
class UserDefaultsMock: UserDefaultsProtocol {

    private var objectData = [String : AnyObject]()

    func objectForKey(key: UserDefaultsKey) -> AnyObject? {
        return objectData[key.name]
    }

    func dictionaryForKey(key: UserDefaultsKey) -> [NSObject : AnyObject]? {
        return objectData[key.name] as! [NSObject : AnyObject]?
    }

    func boolForKey(key: UserDefaultsKey) -> Bool {
        return objectData[key.name] as! Bool
    }

    func setObject(value: AnyObject?, forKey key: UserDefaultsKey) {
        objectData[key.name] = value
    }

    func setBool(value: Bool, forKey key: UserDefaultsKey) {
        objectData[key.name] = value
    }

    func removeObjectForKey(key: UserDefaultsKey) {
        objectData[key.name] = nil
    }

}
```

TIME TRAVELER TIMES

TEST THAT USES USERDEFAULTSMOCK

```
func testNotificationSettingsLoad() {
    let userDefaultsMock = UserDefaultsMock()
    mockUserDefaults.setObject("NEVER", forKey: "timeMachineBreakingNewsPushNotification")
    mockUserDefaults.setObject("NEVER", forKey: "timeMachineDailyDigestPushNotification")

    let dataProvider = PushNotificationsDataProvider(userDefaults: mockUserDefaults)
    let expectedFrequencies: [PushNotificationFrequency] = [.Never, .All, .All, .Never]

    XCTAssertEqual(expectedFrequencies, dataProvider.notificationSettings.values)

}
```

MOCKING NSNOTIFICATIONCENTER WAS NOT WORTH IT

» Complex

» Injected throughout entire codebase

» Limited functionality compared to real object

» Solves a problem that doesn’t really exist

THEY SAID DON'T MOCK VALUE TYPES

“WHEN YOU’RE DOING...SIMPLE DATA IN AND DATA OUT, YOU DON’T NEED MOCKING OR STUBBING. ”

Andy Matuschak

“YOU CAN PASS A VALUE INTO A FUNCTION, THEN LOOK AT THE RESULTING VALUE.”

Andy Matuschak

HOW?

“TRY MAKING YOUR REFERENCE TYPES IMMUTABLE”

Joe Groff

TRY IT OUT

GOOD MOCKS

WHAT MAKES A GOOD MOCK

» Quick and easy to write

» Relatively short and does not contain tons of information you don’t need

» Legitimate reason to not use real object

FELLOW TIME TRAVELERS


LET'S LOOK TOWARDS THE FUTURE

SWIFT

MOCKING FRAMEWORKS

» Dobby

» MockFive

» SwiftMock

» Cuckoo

A WORLD WITHOUT MOCKS

WRITE MOCKS AND USE THEM FOR GREAT GOOD

THANK YOU!