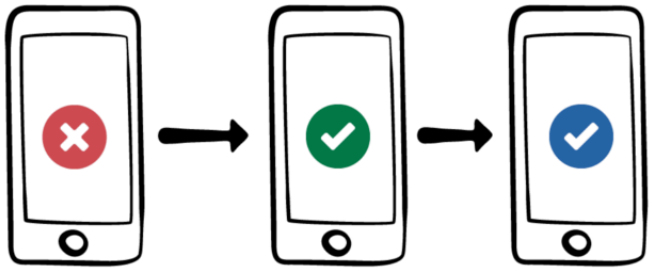
BEGINNING iOS UNIT & UI TESTING



HANDS-ON CHALLENGES

Beginning iOS Unit and UI Testing

Joshua Greene

Notice of Rights

Notice of Liability

Trademarks

3 Async & Mocking - Challenge

By Joshua Greene

There are two ways to create mock objects in Swift:

1. By conforming to a protocol
2. By subclassing a real object and overriding methods/properties/etc

Where possible, you should prefer creating mocks by conforming to a protocol, as this will give you the most flexibility and be less likely to unexpectedly break over time.

If you create a mock by subclassing and overriding a real class, there's a chance the real implementation may change and unintentionally break your mock.

Either way, you also have to "inject" your mock at runtime. This is done by either **initiailizer injection**, where you pass your mock object as a parameter to an `init` method, or by **property injection**, where you set a property on your test class to a mock object.

Challenge

Your challenge is to create a mock object that returns a canned response whenever `loadPancakeHouses(success:, failure:)` is called.

You'll then write a test that uses this mock to verify `collection.loadPancakesFromCloud(completion:)` calls its completion handler, expects `didReceiveData` to be `true` and sets the collection's `pancakeHouses` array to the returned array.

Hints

Here's an overview of what you'll need to do:

1. Create a new protocol called `CloudNetworkService`. This should declare the same two methods already on `CloudNetworkManager`.
2. Make `CloudNetworkManager` conform to `CloudNetworkService`.
3. In `PancakeHouseCollection`, explicitly declare the type of `_cloudNetworkManager` to be `CloudNetworkService` instead of having it inferred by the compiler.
4. Create a `MockCloudNetworkService` class that conforms to `CloudNetworkService` and add this to the `StackReviewIntegrationTests` target.
5. On your `MockCloudNetworkService`, stub out the response for `loadPancakeHouses(success:, failure:)` to return a `PancakeHouse` array created from `test_pancake_houses.plist`.

   You'll also need to make sure you add this plist as a member of the `StackReviewIntegrationTests` target.
6. Add a new test to the `PancakeHouseCollectionIntegrationTests` class that sets an instance of `MockCloudNetworkService` on the `collection` property and verifies that `collection.loadPancakesFromCloud(completion:)` calls the completion handler with `didReceiveData` as `true` and that the `collection`'s `pancakeHouses` array is set to the array returned by the mock.

Whew, this is going to take a bit of work...!

See how much you can do yourself first. If you get stuck, check out the solution on the next page for walkthrough instructions.

Solution

Select the **Networking** group, and press ⌘**n** to create a new file. Select **Swift File** from the list, and name this **CloudNetworkService.swift**.

Replace the entire contents with the following:

```
public protocol CloudNetworkService {

  func login(userName: String, password: String,
             success: ()->(), failure: (Error)->())

  func loadPancakeHouses(
                success: @escaping ([PancakeHouse])->(),
                failure: @escaping (CloudNetworkError)->())
}
```

Next, open **CloudNetworkManager.swift** and replace `public class CloudNetworkManager` with `public class CloudNetworkManager: CloudNetworkService`.

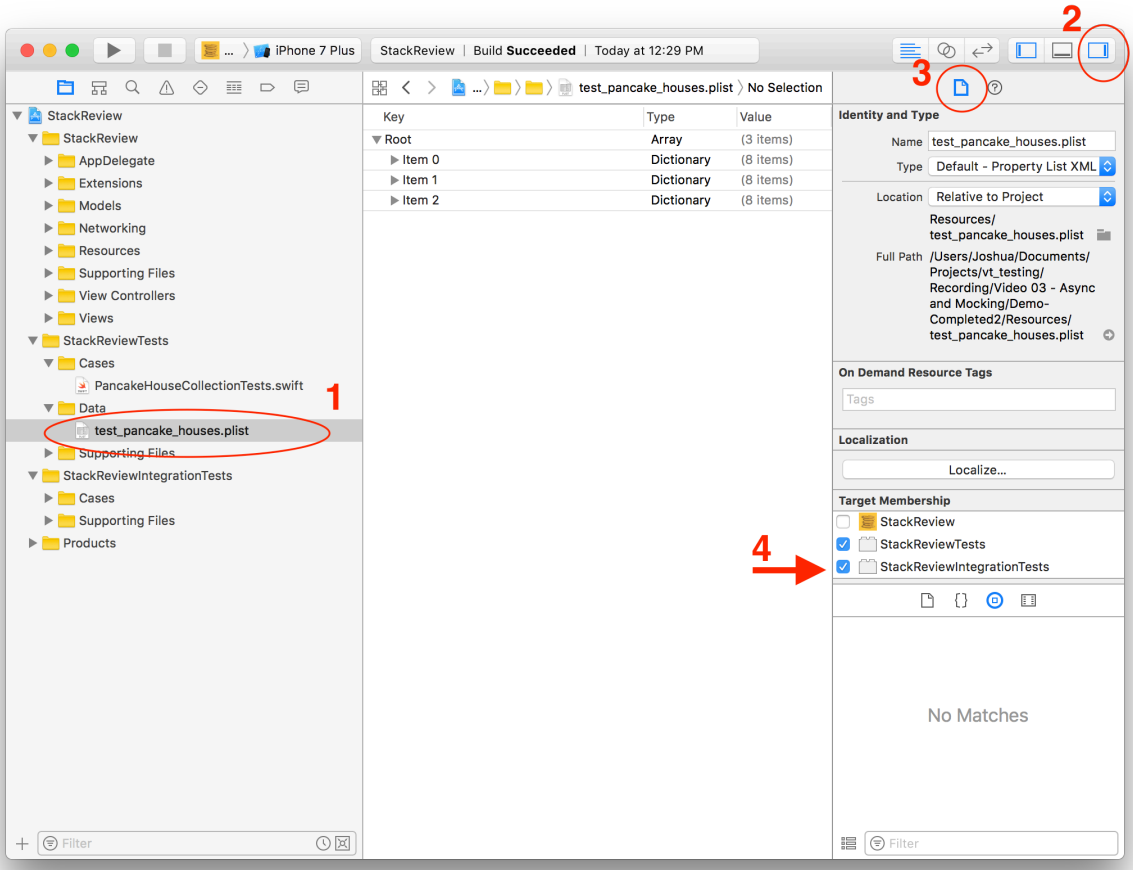
Since this class already conforms to `CloudNetworkService`, you don't need to make any other changes.

Open **PancakeHouseCollection.swift** and replace `internal var _cloudNetworkManager = CloudNetworkManager()` with `internal var _cloudNetworkManager: CloudNetworkService = CloudNetworkManager()`.

This explicitly sets the type of `_cloudNetworkManager` instead of having the compiler infer it.

Change the `test_pancake_houses.plist` Target membership by doing the following:

1. Select `test_pancake_houses.plist` under the **StackReviewTests\Data** group.
2. Open **Utilities** (the right pane).
3. Select the **File Inspector** (the "page" tab).
4. Under **Target Membership**, check the box next to **StackReviewIntegrationTests**.



Select the **StackReviewIntegrationTests** group, and press ⌘**n** to create a new file. Select **Swift File** from the list, and name this **MockCloudNetworkService.swift**.

Replace the entire contents of the file with the following:

```
import Foundation
@testable import StackReview

public class MockCloudNetworkService: CloudNetworkService {

  public lazy var pancakeHouses: [PancakeHouse] = {
    let bundle = Bundle(for: type(of: self))
    let path = bundle.path(forResource: "test_pancake_houses",
                           ofType: "plist")!
    let array = NSArray(contentsOfFile: path)
    return PancakeHouse.from(array as! [[String : Any]])
  }()

  public func login(userName: String, password: String,
                    success: ()->(),
                    failure: (Error)->()) {
    success()
```

```
  }

  public func loadPancakeHouses(
                  success: @escaping ([PancakeHouse])->(),
                  failure: @escaping (CloudNetworkError)->()) {
    success(pancakeHouses)
  }
}
```

Next, add the following test to the end of `PancakeHouseCollectionIntegrationTests`:

```
func testGivenMockCloudNetworkServiceLoadPancakesFromCloudSucceeds() {

  // given
  let mockCloudNetworkService = MockCloudNetworkService()
  collection._cloudNetworkManager = mockCloudNetworkService

  let expecation = self.expectation(description:
    "Expected load pancakes from cloud to succeed")

  // when
  collection.loadPancakesFromCloud { (didReceiveData) in
    expecation.fulfill()
    XCTAssertTrue(didReceiveData)
    XCTAssertEqual(self.collection._pancakeHouses,
                   mockCloudNetworkService.pancakeHouses)
  }

  // then
  waitForExpectations(timeout: 0.1, handler: nil)
}
```

Run the `StackIntegrationTests`, and verify the new test passes. :]

Über challenge

We've used the term "mock" in a very general sense. However, you're likely to come across terms like "test double," "spy," "stub," "fake" and others.

For the Über challenge, read Martin Fowler's brief article about "Test Doubles" to learn what each of these actually mean:

http://www.martinfowler.com/bliki/TestDouble.html