ATLS 4320: Advanced Mobile Application Development

Week 4: Unit Testing

See slides for more general info on Unit Testing!

Project set up

Either use your copy of the NPS Campsites demo app from last week, or pull my starter project down from the repo.

Add a Unit Test target if it isn't already in your project (my starter doesn't have it)
File->New->**Target**->Unit Testing Bundle
Name it *CampsitesTest*
Ensure that the Target to be Tested is the name of your main app target

It auto generates a Unit Test Case Class for you which is a subclass of XCTestCase
You also get the setUp() and tearDown() method stubs, a unit test stub, and a performance test stub
Delete the testExample() method
Delete the performance test because we won't be getting into that
We'll be writing some tests for our Model class since this is where most of the logic could go wrong.

Unit Test: Valid API Response

We'll start by writing a test to make sure we get a valid response when we make a valid call

Import the Campsites target underneath the top import statement for XCTest

```
//import the Campsites target to get access to private types
@testable import Campsites
```

Define our system under test as an optional class property

```
//our system under test is a URL session
var sut: URLSession!
```

Implement the setUp() method

```
override func setUp() {
    super.setUp()
    //init our session
    sut = URLSession(configuration: .default)
}
```

Implement the tearDown() method

```
override func tearDown() {
    //dereference sut
    sut = nil
    super.tearDown()
}
```

Write to make sure the dataTask completion handler is invoked and the response is valid. We **need** to use an expectation because we are testing asynchronous functionality. Essentially, we need to **make sure our test waits for the expectation to resolve before we can verify the results**.

```
func test_ValidCallToNPSGets200Response() {
    //MARK: given
    let url = URL(string:
"https://developer.nps.gov/api/v1/campgrounds?stateCode=co&api_key=KuzYWrSK
Hbuz6CBO8oc0pX35CeljxNSfgxane4IH")
    //we guarantee that the completion handler will be invoked
    let promise = expectation(description: "Completion handler invoked")
    //variables to hold parts of the response we're concerned with
    var statusCode: Int?
    var responseError: Error?

    //MARK: when
    let dataTask = sut.dataTask(with: url!, completionHandler: {data,
response, error in
        //simply assign the completionHandler parameters to our variables
        statusCode = (response as? HTTPURLResponse)?.statusCode
        responseError = error
        promise.fulfill()
    })

    //run the data task
    dataTask.resume()
    //wait for the promise to fulfill or 5 seconds whichever comes first
    wait(for: [promise], timeout: 5)

    //MARK: then
    //make sure we didn't get an error
    XCTAssertNil(responseError)
    //make sure that we got a 200 response back
    XCTAssertEqual(statusCode, 200)
}
```

Run the test and verify that it passes

Unit Test: Parsing Valid Mock HTTP Response

Next we'll write a test to make sure that our parsing function executes properly.
This test is more complicated because we want to make sure that it is independent and does not rely on an actual API response. Thus, we will need to do some **dependency injection** where we mock the response and provide some static data that we know is valid. Essentially, we are assuming we get a valid response and then we test how we handle that.

Write the Test

Create a new Unit Test Class
Ctrl-click on the CampsitesTests Target
New File...->Unit Test Case Class
Name it CampsitesDataControllerTests

Remove the auto-generated tests

Import our target

```
@testable import Campsites
```

Define the system under test as a class property

```
//define our system under test
var sut: CampsiteDataController!
```

Implement the setUp() and tearDown() methods

```
override func setUp() {
    super.setUp()
    //init our system under test in set up so it can be destroyed later
    sut = CampsiteDataController()
}

override func tearDown() {
    //de-reference sut
    sut = nil
    super.tearDown()
}
```

Write the test using a mock session and response with dummy data

```
func test_parseJSON() {
    //get access to the bundle for our test target
    let testBundle = Bundle(for: type(of: self))
    //get path of sample json static data file
    let path = testBundle.path(forResource: "sample-campground-response",
ofType: "json")
    //load contents of file into byte buffer
    let data = try? Data(contentsOf: URL(fileURLWithPath: path!), options:
.alwaysMapped)

    sut.parseJson(rawData: data!)

    XCTAssertEqual(sut.currentCampsites.data.count, 17, "Didn't parse 17
items from fake JSON response")
}
```

Run the test to make sure it succeeds

Next steps

Let's **check** our code coverage
Go to Product->Scheme->Edit Scheme
Select *Test* in the side panel
Select the Options tab near the top
Check the box next to Code Coverage (Gather coverage for all targets)

Now we can **run** all the tests in our suite.

To **view** Coverage results **go** to the Report navigator in the left side panel. In the most recent test report, **select** the Coverage item and we can see how much of our code was utilized in our tests.

Also, you can **select** Editor->Code Coverage to see what code was called in a specific file.

Our tests aren't great because we only test our success cases. We would want to write some more tests that make sure our logic is sound when something goes wrong retrieving or parsing the data (boundary cases).

Write UI Tests to test common UI workflows (button changing labels, segmented controls, etc.)