Swift by Practical Example

Justin Miller • Mapbox • @incanus77

Introduction

- Mobile lead at Mapbox
- Eleven years of Swift experience™
  - Not really :-)
- But Cocoa, yes; I started with Project Builder in 10.2, pre-Xcode
- Have shipped three standalone Swift libraries, several Objective-C SDKs with Swift compatibility, and Swift test apps
- So I've been an intermediary between Swift coders and real-world use-cases, as well as been experimenting

Disclaimer

- We’re all new here
- I may be wrong
- But I’ve been playing around (and shipping) a fair amount
- *caveat emptor*

Practical?

- I’m not a computer scientist
- I’m also not a functional programming adherent
- I’m a tool builder and tinkerer
- So I wanted to think about how to present Swift in a practical light

Practicality

- I’m going to focus on two overarching topics:
  - Usefulness of Swift in existing projects
  - Usefulness of some of Swift’s features

What is Swift good for…

- … in the context of integration into existing projects?
- Adding Swift files to Objective-C projects
  - Especially extensions
- REST service wrappers
- Command-line utilities
- Prototyping algorithms, especially visual ones
  - Playgrounds are in a good state these days

What is Swift good for…

- … in the context of language features that can save programmers some pain?
- Type aliases
- Nested functions
- Nil coalescing
- Lazy loading
- Closures
- Optional chaining

Integration

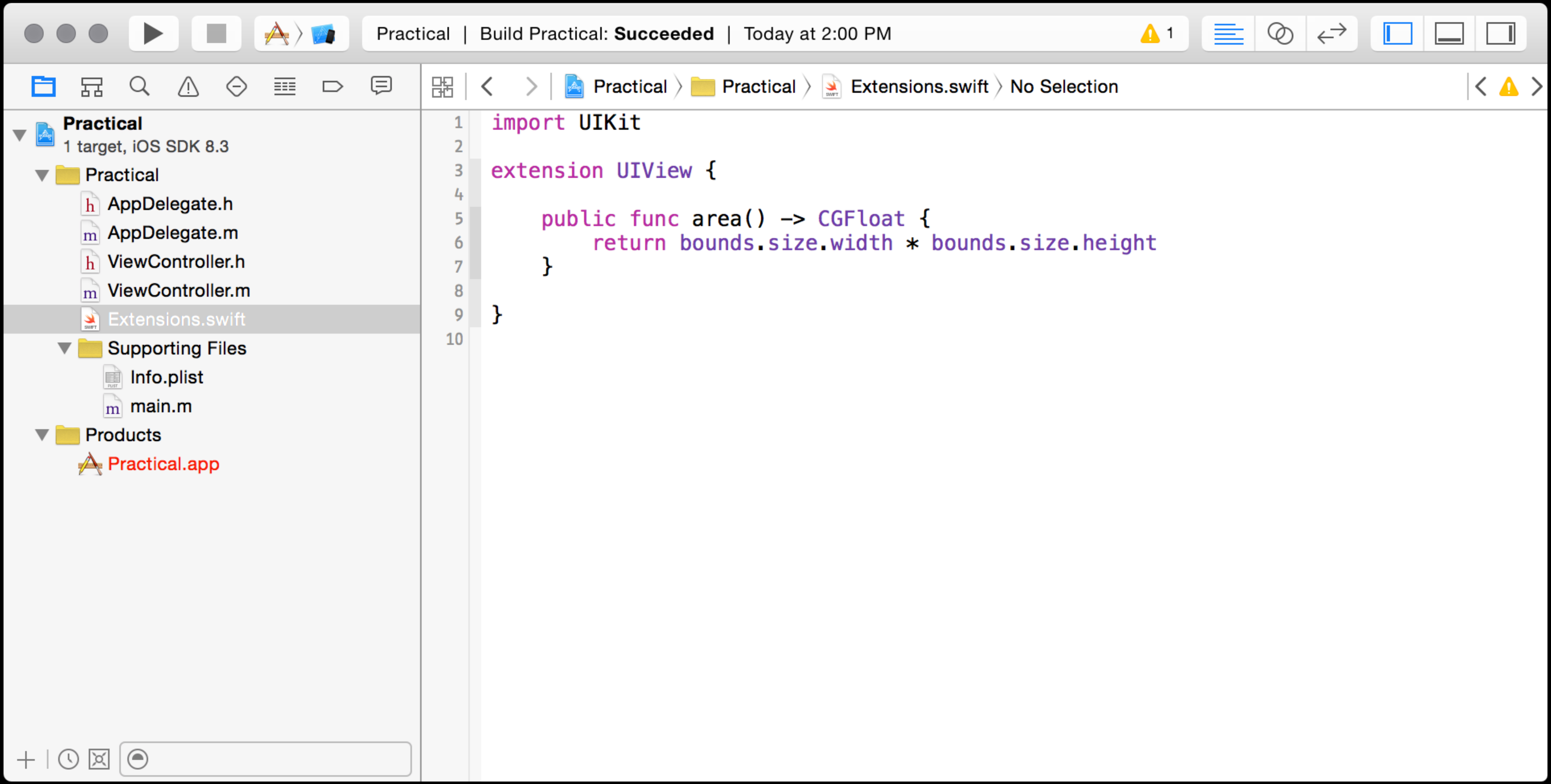
Easy Integration

- *“[Swift …] uses the Objective-C runtime, allowing C, Objective-C, C++ and Swift code to run within a single program.”*
- A lot of potential for piecemeal integration
- You can start Swifting today with just one file

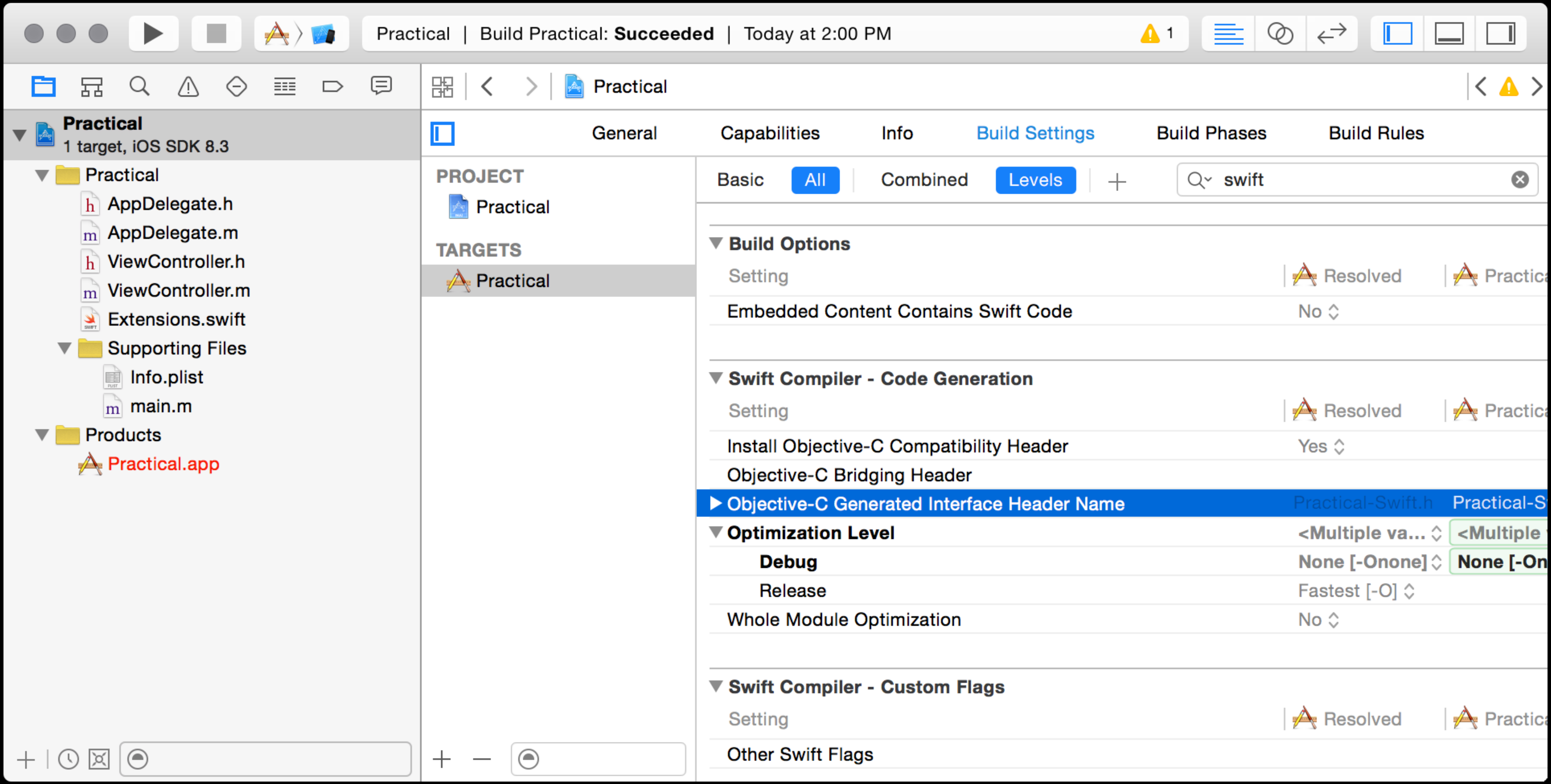
https://en.wikipedia.org/wiki/Swift_(programming_language)

Easy Integration

- You can use Objective-C from Swift
  - But if you already had a Swift app, you probably wouldn’t be here
- But it’s even easier to use Swift from Objective-C
  - Create **`foo.swift`**
  - **`#import “<Target>-Swift.h”`**
  - That’s it!



```swift
import UIKit

extension UIView {

    public func area() -> CGFloat {
        return bounds.size.width * bounds.size.height
    }

}
```


Practical | Build Practical: **Succeeded** | Today at 2:00 PM

Practical
1 target, iOS SDK 8.3
- Practical
  - AppDelegate.h
  - AppDelegate.m
  - ViewController.h
  - ViewController.m
  - Extensions.swift
  - Supporting Files
    - Info.plist
    - main.m
- Products
  - Practical.app

PROJECT
Practical
TARGETS
Practical

General | Capabilities | Info | Build Settings | Build Phases | Build Rules

Basic | All | Combined | Levels | swift

Build Options

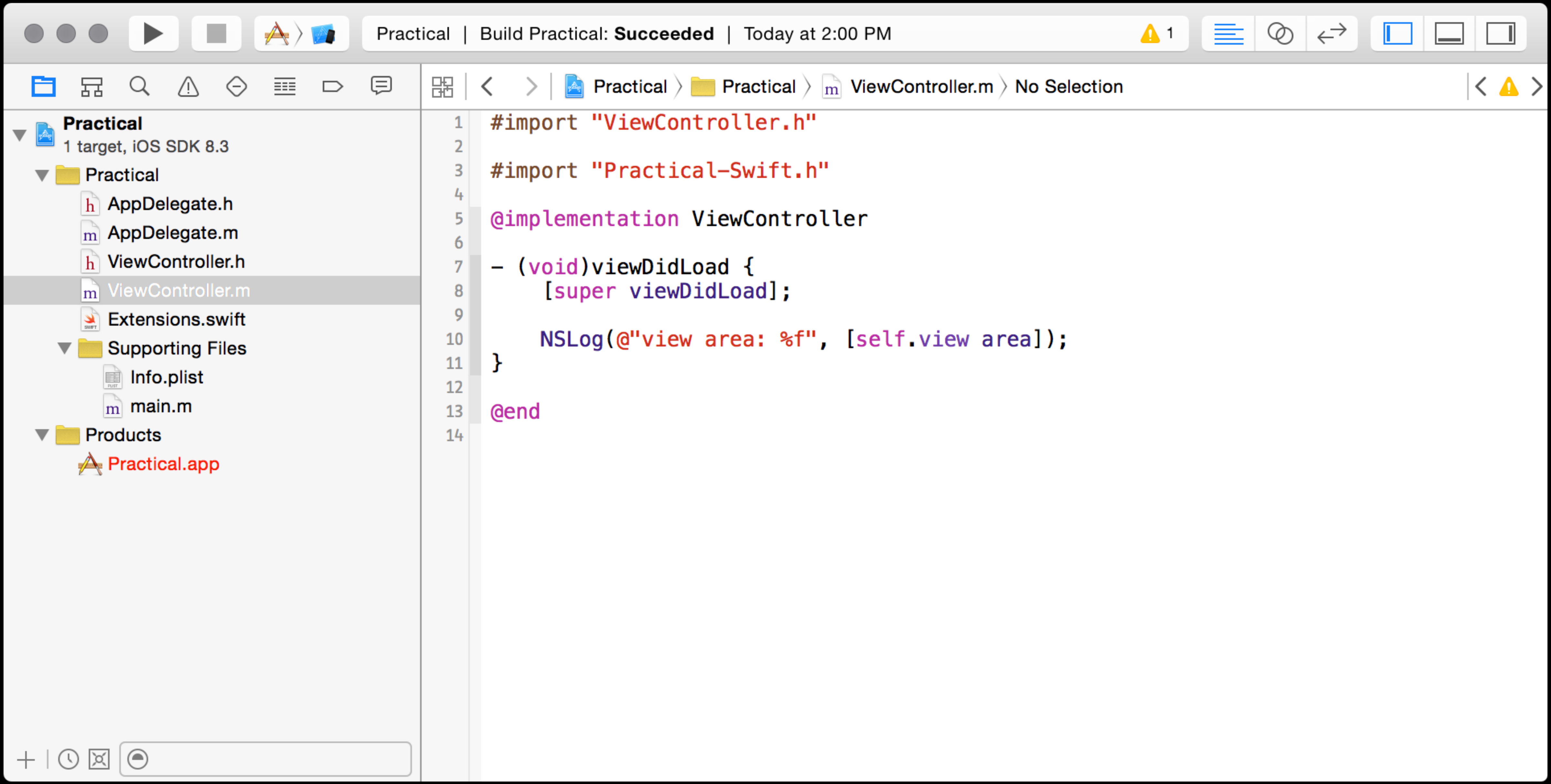
Setting	Resolved	Practica
Embedded Content Contains Swift Code	No	

Swift Compiler - Code Generation

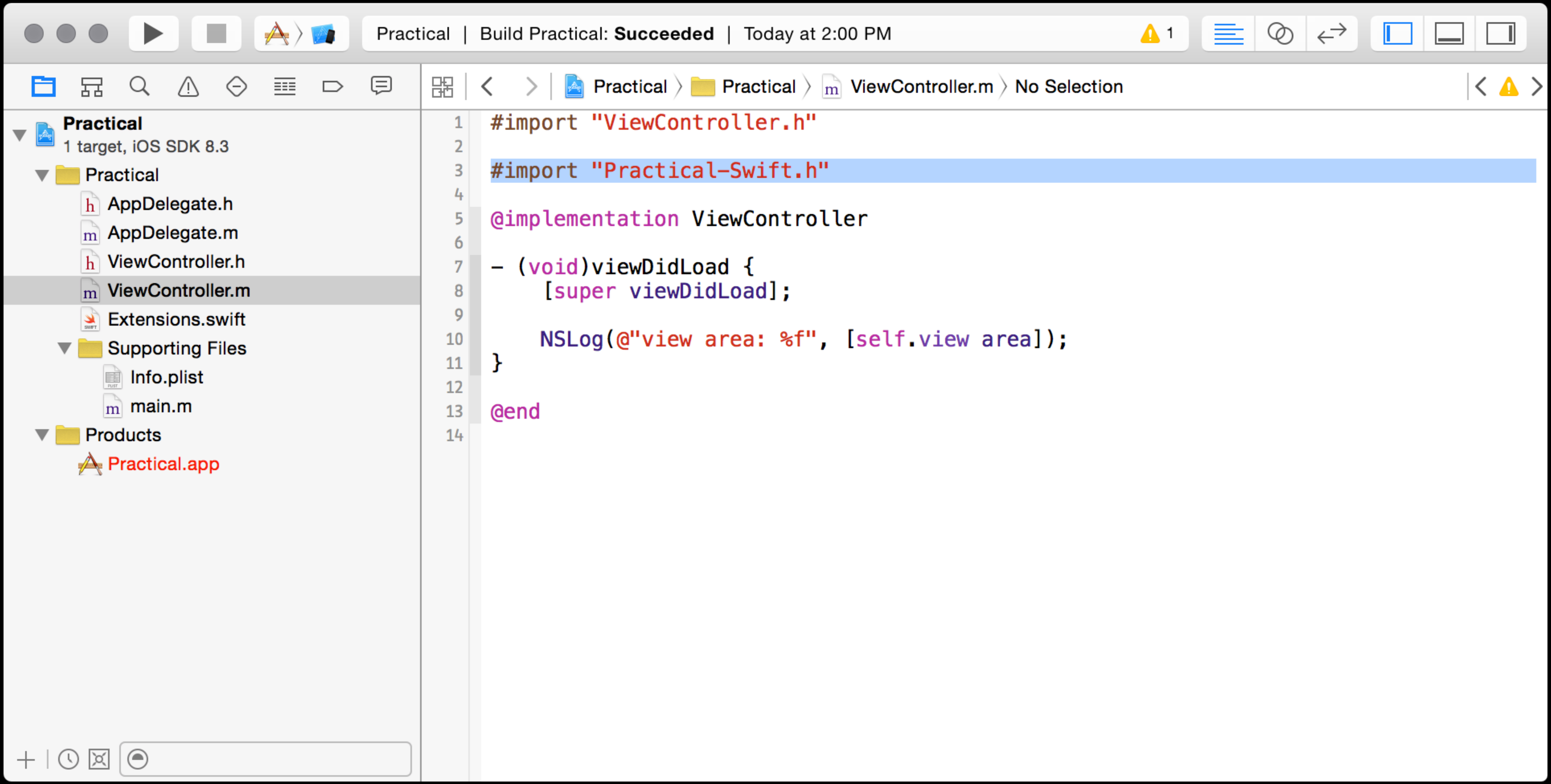
Setting	Resolved	Practica
Install Objective-C Compatibility Header	Yes	
Objective-C Bridging Header		
Objective-C Generated Interface Header Name	Practical-Swift.h	Practical-S
Optimization Level	<Multiple va...	<Multiple
Debug	None [-Onone]	None [-On
Release	Fastest [-O]	
Whole Module Optimization	No	

Swift Compiler - Custom Flags

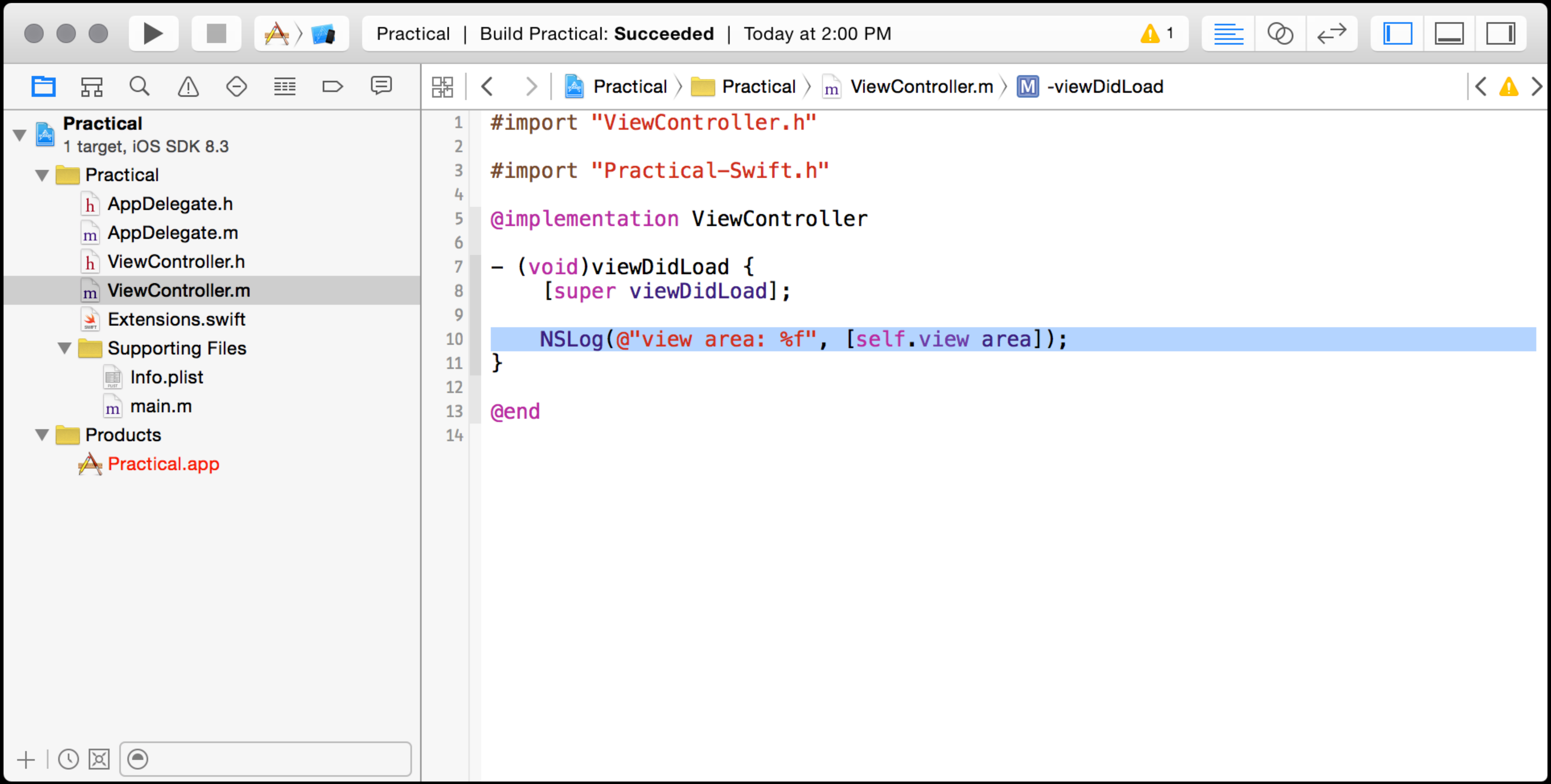
Setting	Resolved	Practica
Other Swift Flags		



```
#import "ViewController.h"

#import "Practical-Swift.h"

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"view area: %f", [self.view area]);
}

@end
```



```objc
#import "ViewController.h"

#import "Practical-Swift.h"

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"view area: %f", [self.view area]);
}

@end
```



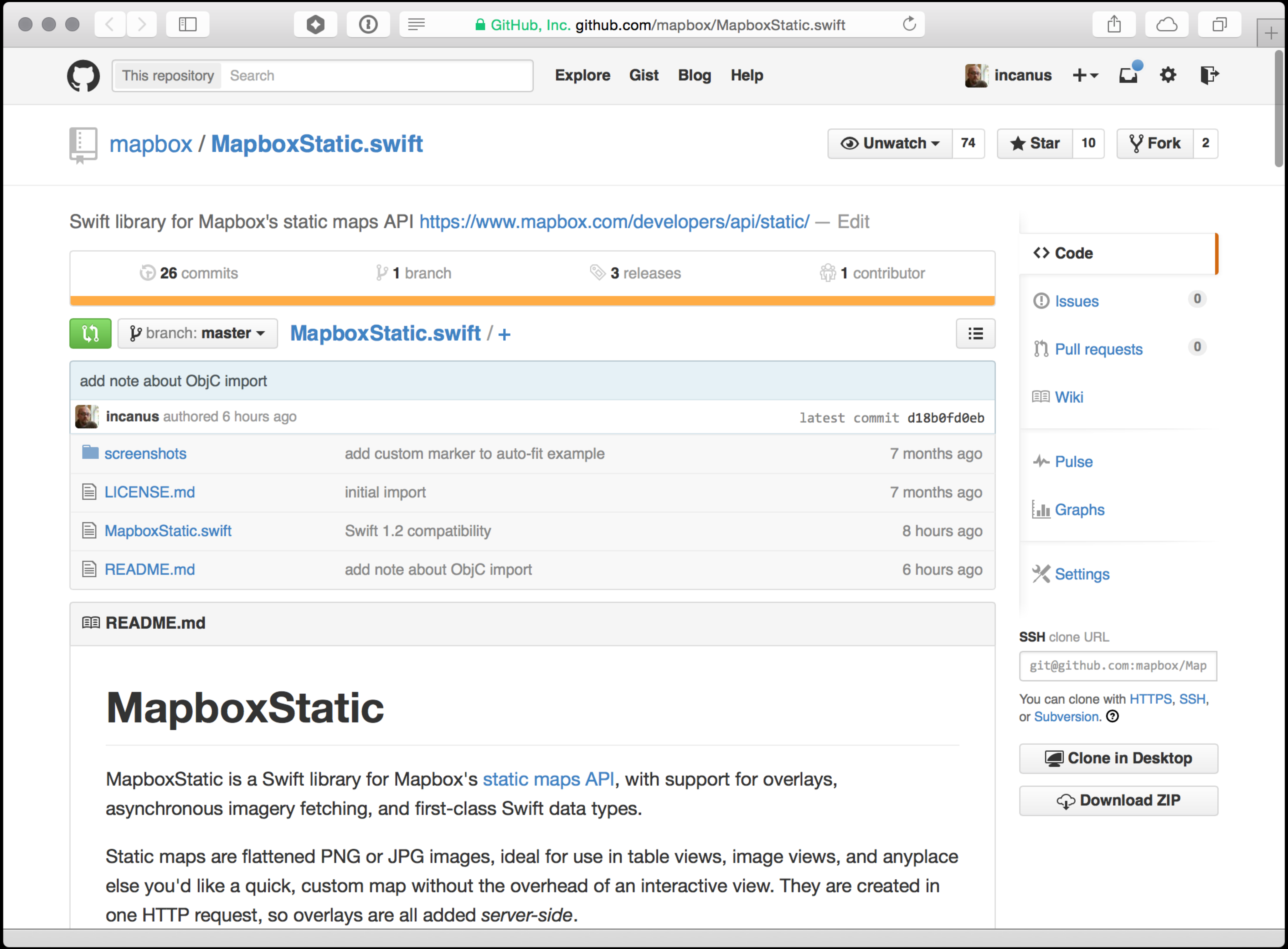
```
#import "ViewController.h"

#import "Practical-Swift.h"

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    NSLog(@"view area: %f", [self.view area]);
}

@end
```

REST Wrappers

- Well-suited to all-in-one file style of Swift
- We've found them easier read by web services folks
- Essentially three pieces:
  - A request URL constructor (but nice)
  - An **`NSURLConnection`** / **`NSURLSession`** manager
  - A closure caller and a raw URL provider

Example Web Service

- **`let service = MyGreatService(foo, bar)`**
- Then, three action options:
  - Do things with **`service.requestURL`** (e.g. AFNetworking)
  - Obtain e.g. **`service.image`** (blocking)
  - **`service.imageWithHandler { image in`** … **`}`** (non-blocking)



mapbox / MapboxStatic.swift

Swift library for Mapbox's static maps API https://www.mapbox.com/developers/api/static/ — Edit

26 commits 1 branch 3 releases 1 contributor

MapboxStatic.swift / +

add note about ObjC import

incanus authored 6 hours ago latest commit d18b0fd0eb

screenshots	add custom marker to auto-fit example	7 months ago
LICENSE.md	initial import	7 months ago
MapboxStatic.swift	Swift 1.2 compatibility	8 hours ago
README.md	add note about ObjC import	6 hours ago

SSH clone URL

`git@github.com:mapbox/Map`

You can clone with HTTPS, SSH, or Subversion.

README.md

MapboxStatic

MapboxStatic is a Swift library for Mapbox's static maps API, with support for overlays, asynchronous imagery fetching, and first-class Swift data types.

Static maps are flattened PNG or JPG images, ideal for use in table views, image views, and anyplace else you'd like a quick, custom map without the overhead of an interactive view. They are created in one HTTP request, so overlays are all added *server-side*.

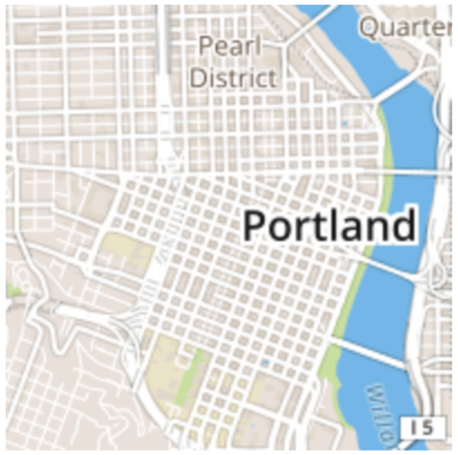
GitHub, Inc. github.com/mapbox/MapboxStatic.swift

The main map class is `MapboxStaticMap`. To create a basic map, specify the center, zoom level, and pixel size:

```swift
let map = MapboxStaticMap(
    mapID: <your map ID>,
    center: CLLocationCoordinate2D(latitude: 45.52, longitude: -122.681944),
    zoom: 13,
    size: CGSize(width: 200, height: 200),
    accessToken: <your API token>
)
```

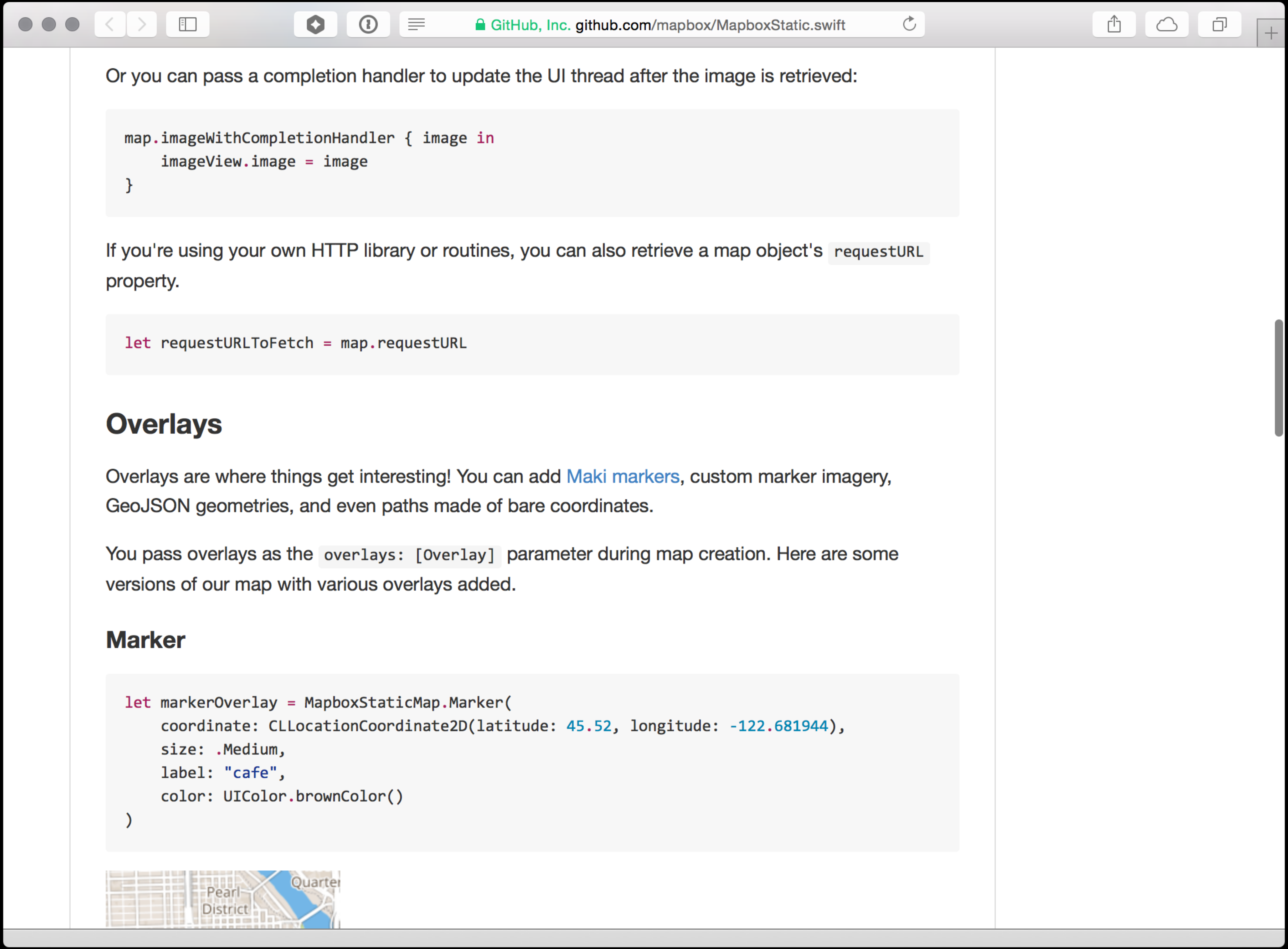
Then, to retrieve an image, you can do it either synchronously (blocking the calling thread):

```swift
self.imageView.image = map.image
```



Or you can pass a completion handler to update the UI thread after the image is retrieved:

```swift
map.imageWithCompletionHandler { image in
    imageView.image = image
}
```



Or you can pass a completion handler to update the UI thread after the image is retrieved:

```
map.imageWithCompletionHandler { image in
    imageView.image = image
}
```

If you're using your own HTTP library or routines, you can also retrieve a map object's `requestURL` property.

```
let requestURLToFetch = map.requestURL
```

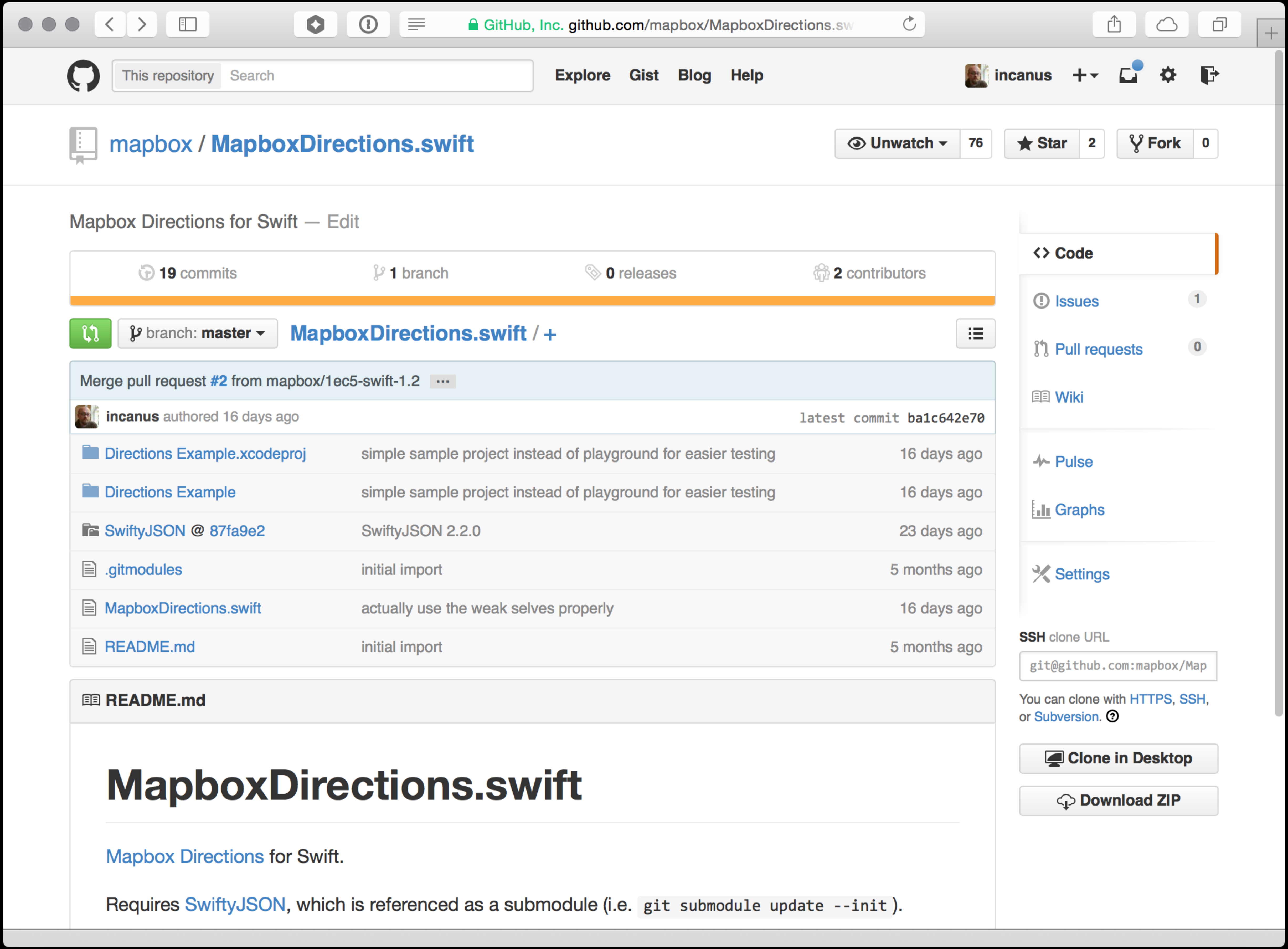
Overlays

Overlays are where things get interesting! You can add Maki markers, custom marker imagery, GeoJSON geometries, and even paths made of bare coordinates.

You pass overlays as the `overlays: [Overlay]` parameter during map creation. Here are some versions of our map with various overlays added.

Marker

```
let markerOverlay = MapboxStaticMap.Marker(
    coordinate: CLLocationCoordinate2D(latitude: 45.52, longitude: -122.681944),
    size: .Medium,
    label: "cafe",
    color: UIColor.brownColor()
)
```



mapbox / MapboxDirections.swift

Mapbox Directions for Swift — Edit

19 commits | 1 branch | 0 releases | 2 contributors

branch: master | MapboxDirections.swift / +

Merge pull request #2 from mapbox/1ec5-swift-1.2 ···

incanus authored 16 days ago — latest commit ba1c642e70

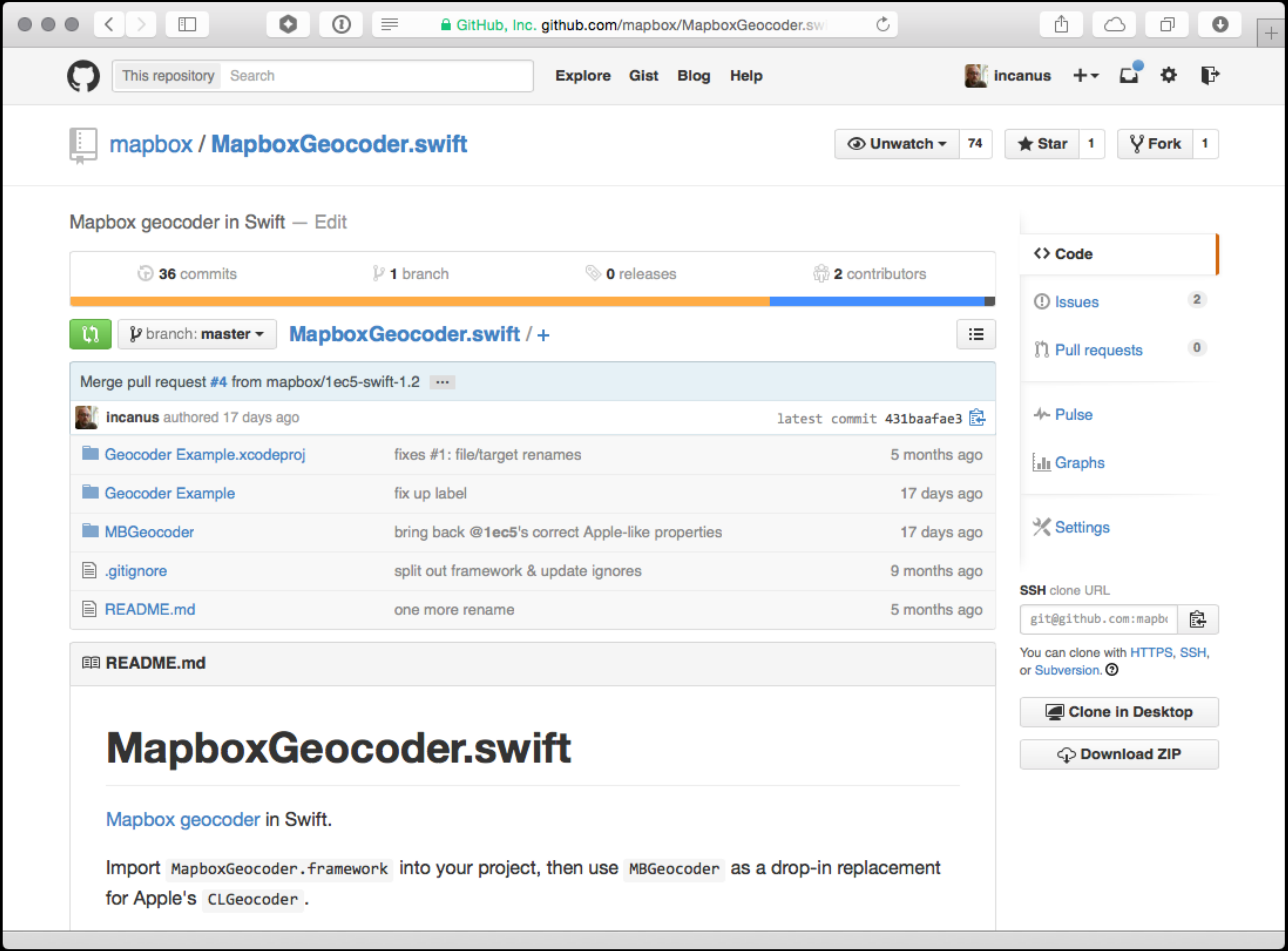
Directions Example.xcodeproj	simple sample project instead of playground for easier testing	16 days ago
Directions Example	simple sample project instead of playground for easier testing	16 days ago
SwiftyJSON @ 87fa9e2	SwiftyJSON 2.2.0	23 days ago
.gitmodules	initial import	5 months ago
MapboxDirections.swift	actually use the weak selves properly	16 days ago
README.md	initial import	5 months ago

README.md

MapboxDirections.swift

Mapbox Directions for Swift.

Requires SwiftyJSON, which is referenced as a submodule (i.e. `git submodule update --init`).



mapbox / MapboxGeocoder.swift

Mapbox geocoder in Swift — Edit

36 commits · 1 branch · 0 releases · 2 contributors

branch: master · MapboxGeocoder.swift / +

Merge pull request #4 from mapbox/1ec5-swift-1.2 ···

incanus authored 17 days ago · latest commit 431baafae3

Geocoder Example.xcodeproj	fixes #1: file/target renames	5 months ago
Geocoder Example	fix up label	17 days ago
MBGeocoder	bring back @1ec5's correct Apple-like properties	17 days ago
.gitignore	split out framework & update ignores	9 months ago
README.md	one more rename	5 months ago

SSH clone URL

`git@github.com:mapb`

You can clone with HTTPS, SSH, or Subversion.

README.md

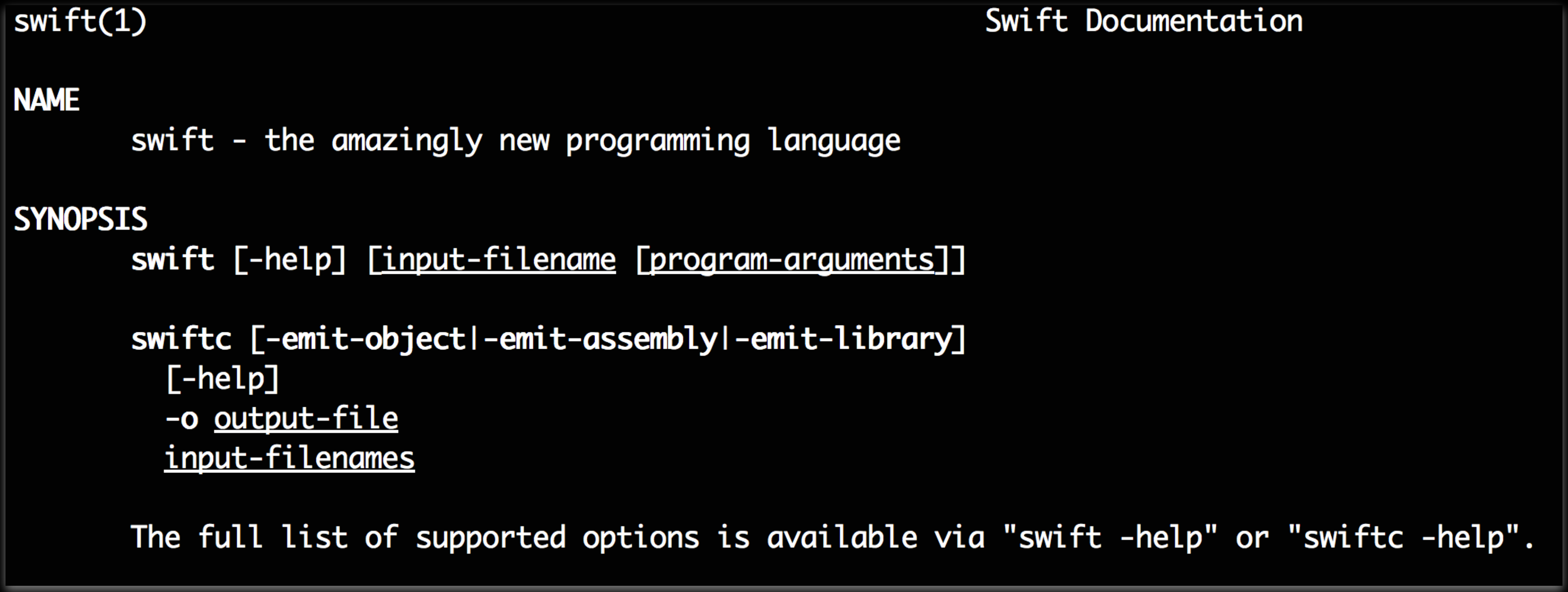
MapboxGeocoder.swift

Mapbox geocoder in Swift.

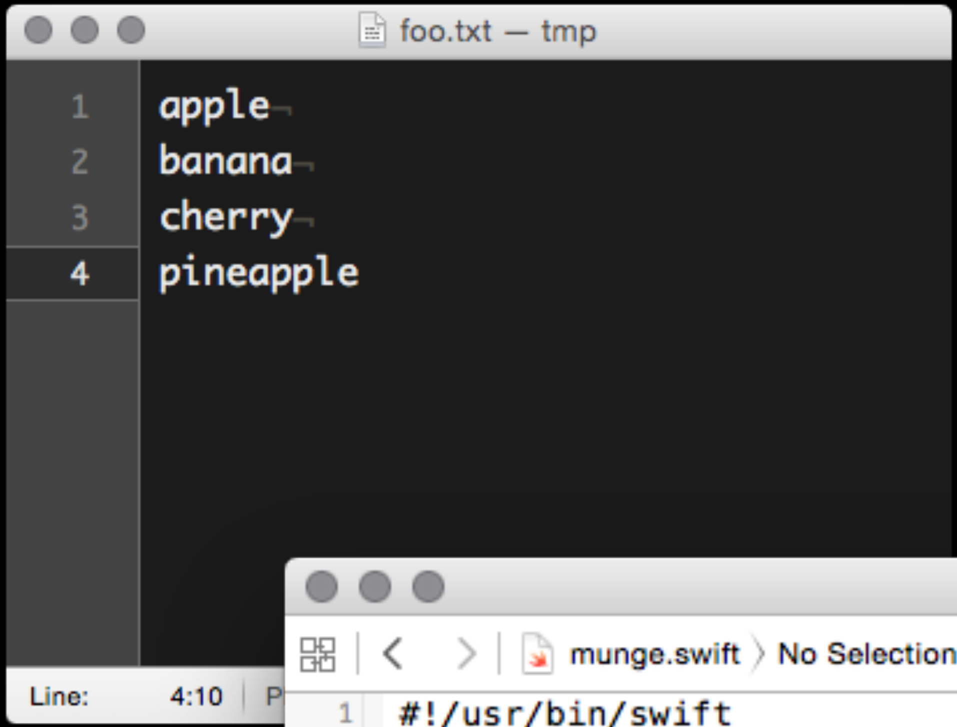
Import `MapboxGeocoder.framework` into your project, then use `MBGeocoder` as a drop-in replacement for Apple's `CLGeocoder`.

Command Line Swift (!)

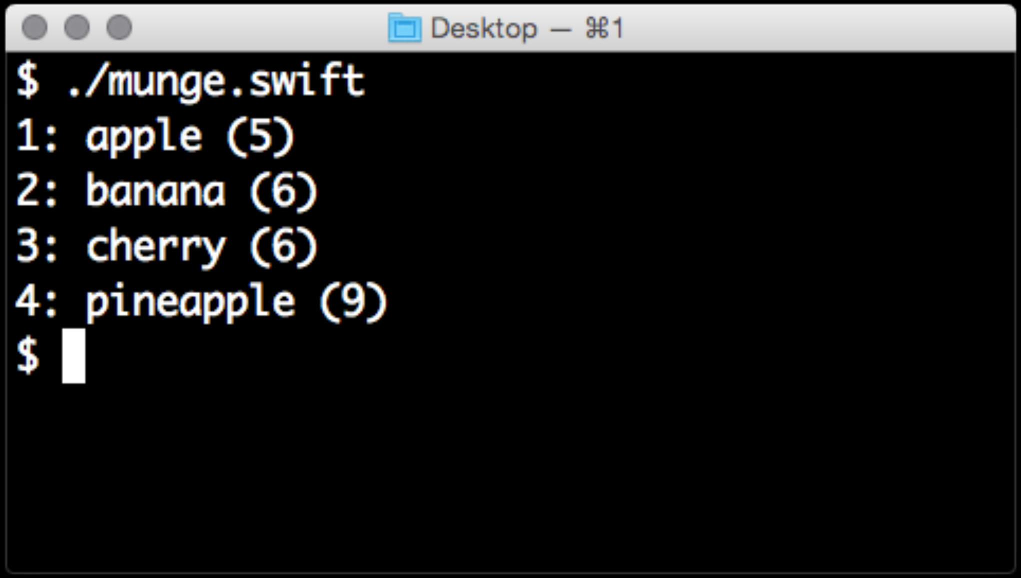
- So, Swift has a REPL (read-eval-print loop)
- By extension, it also just has **`/usr/bin/swift`**
- Use it like Bash, Ruby, Python, Perl, Node…



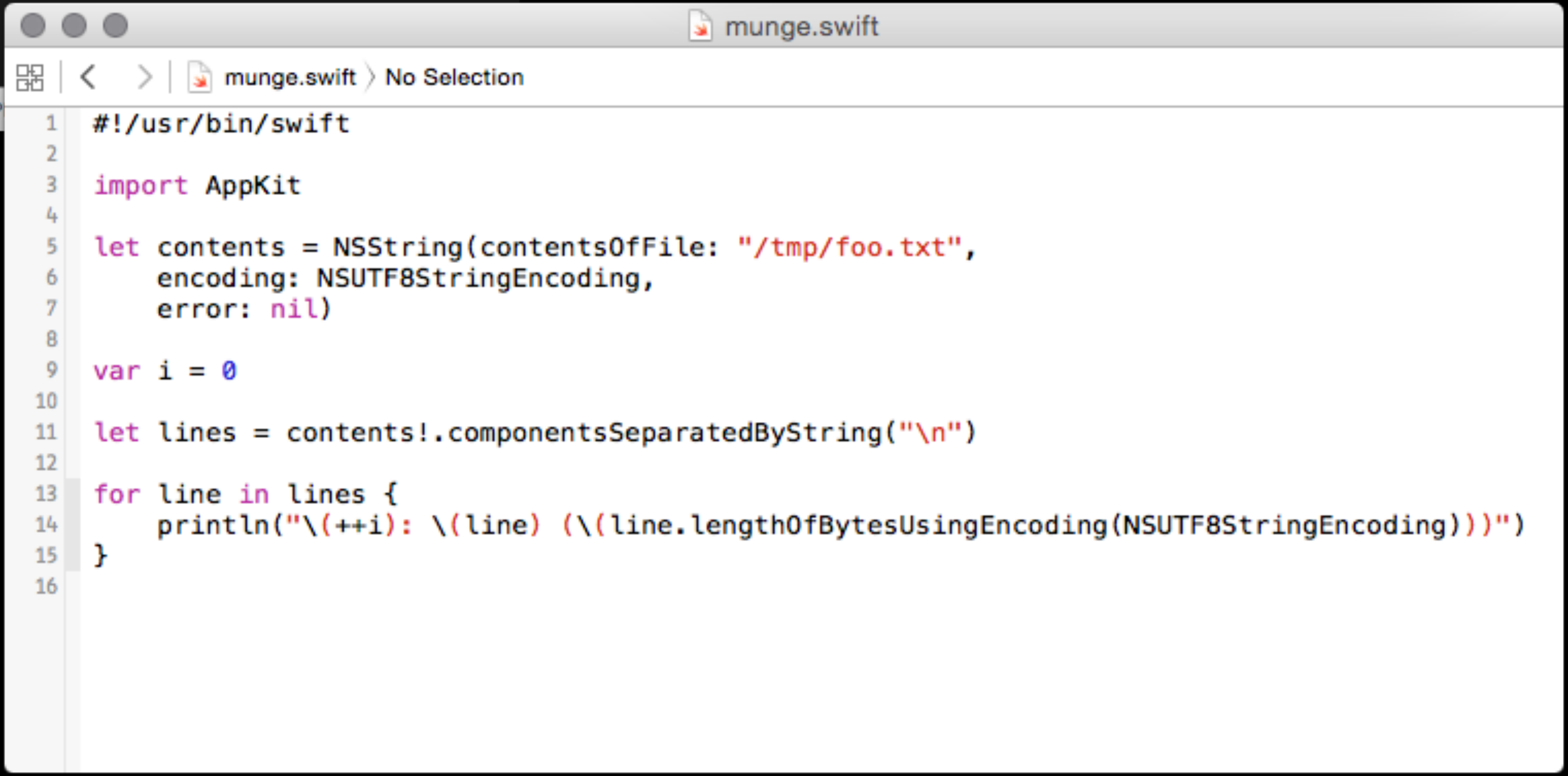
```
swift(1)                                              Swift Documentation

NAME
       swift - the amazingly new programming language

SYNOPSIS
       swift [-help] [input-filename [program-arguments]]

       swiftc [-emit-object|-emit-assembly|-emit-library]
         [-help]
         -o output-file
         input-filenames

       The full list of supported options is available via "swift -help" or "swiftc -help".
```



```
apple
banana
cherry
pineapple
```



```
$ ./munge.swift
1: apple (5)
2: banana (6)
3: cherry (6)
4: pineapple (9)
$
```



```swift
#!/usr/bin/swift

import AppKit

let contents = NSString(contentsOfFile: "/tmp/foo.txt",
    encoding: NSUTF8StringEncoding,
    error: nil)

var i = 0

let lines = contents!.componentsSeparatedByString("\n")

for line in lines {
    println("\(++i): \(line) (\(line.lengthOfBytesUsingEncoding(NSUTF8StringEncoding)))")
}
```

Example Uses

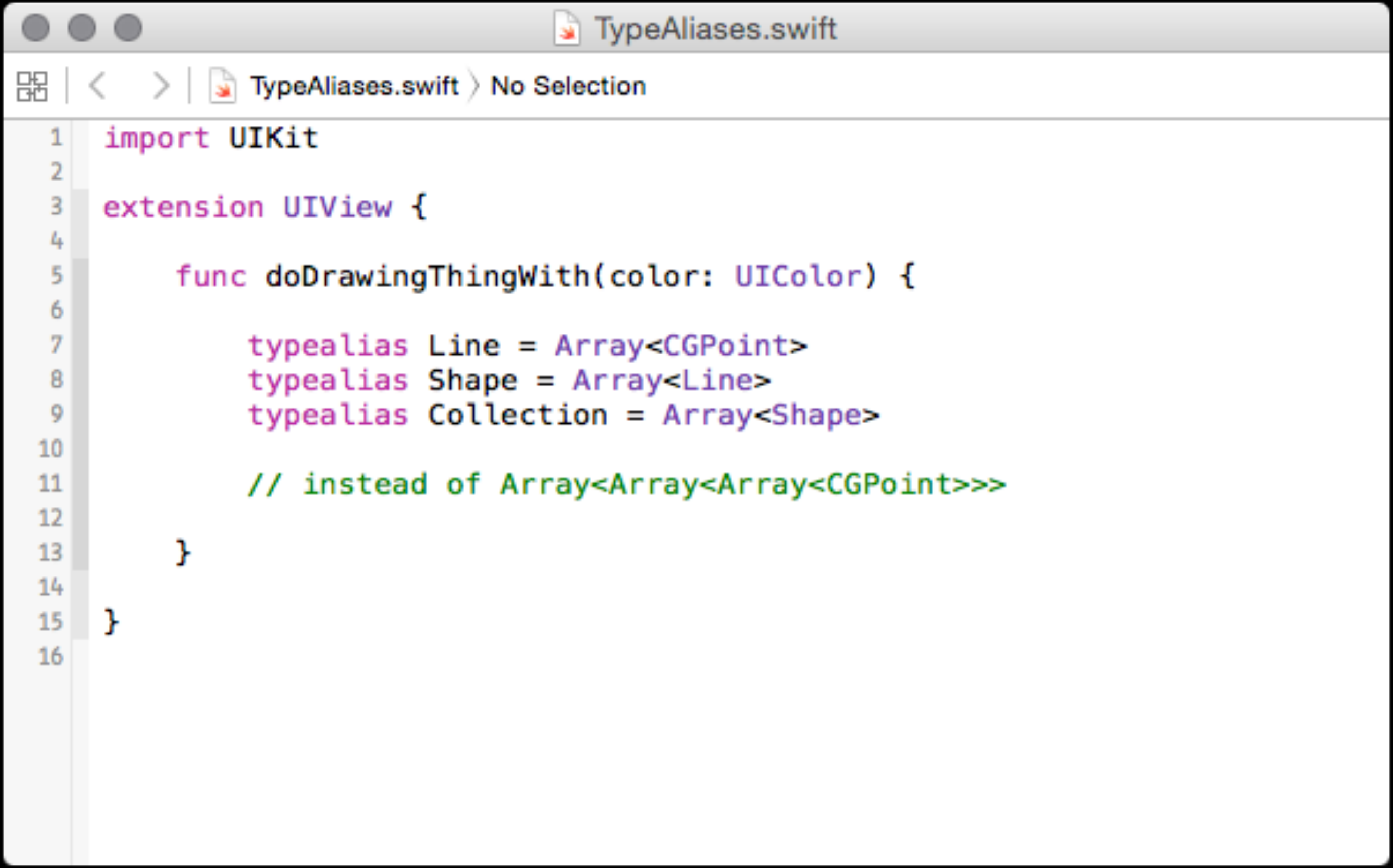
- General housekeeping scripts
- Xcode build phase scripts
    - Great way to start playing with Swift today

Language Features

Type Aliases

- Alias one type to another (obvs)
- Can be created in local scope
- Especially great for typed containers
  - This is commonly used in C++ and is a nice tradeoff
    - Safe, typed containers, but lower verbosity

Local Scope Type Aliases



```swift
import UIKit

extension UIView {

    func doDrawingThingWith(color: UIColor) {

        typealias Line = Array<CGPoint>
        typealias Shape = Array<Line>
        typealias Collection = Array<Shape>

        // instead of Array<Array<Array<CGPoint>>>

    }

}
```

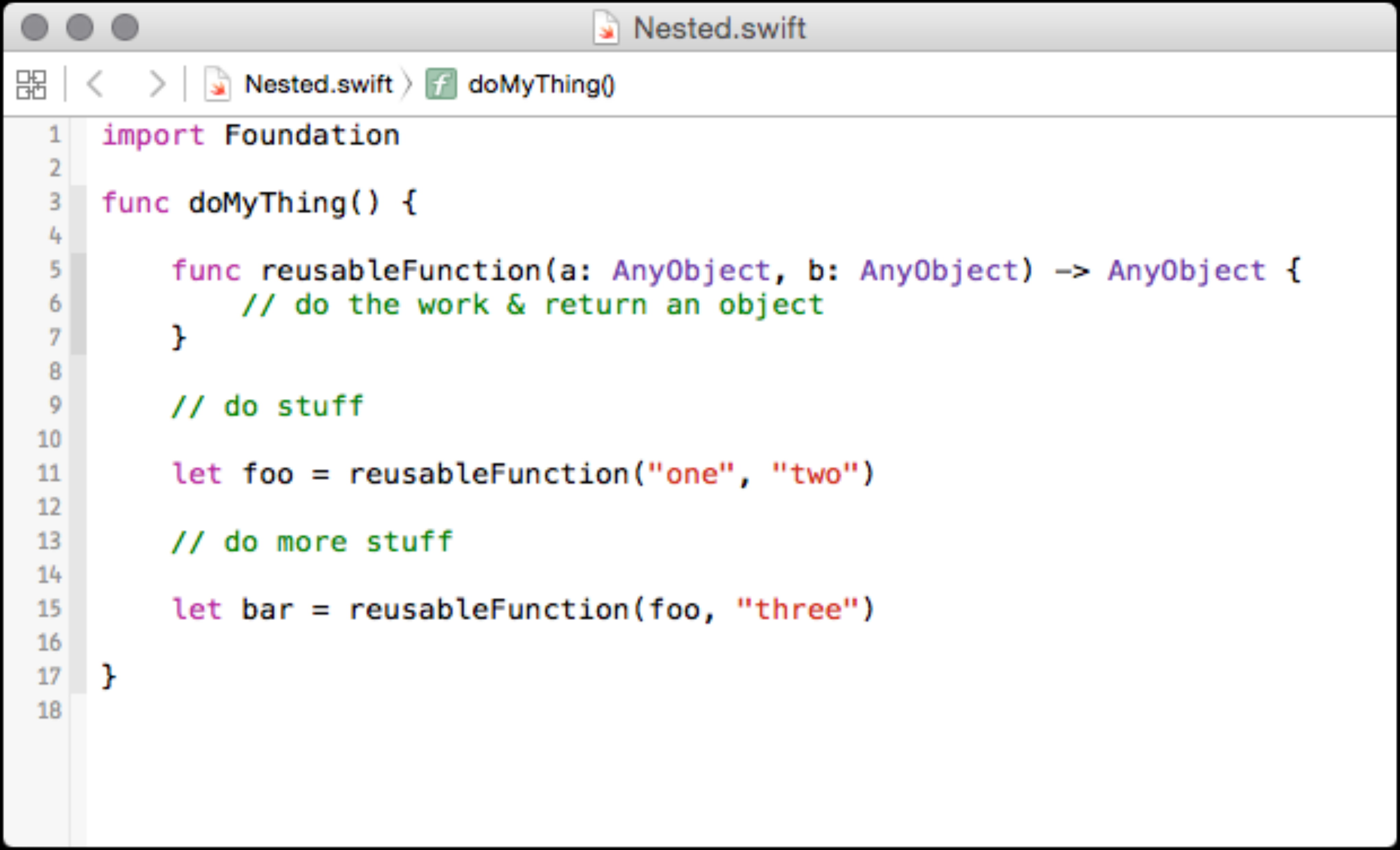
Type Aliases

- Also useful when mimicking an existing class
  - Like, say, a custom version of one of Apple's
  - **`typealias MBGeocodeCompletionHandler = CLGeocodeCompletionHandler`**

Nested Functions

- Like type aliases, can be scoped locally
- Handy for externally non-reusable routines

Nested Functions



```swift
import Foundation

func doMyThing() {

    func reusableFunction(a: AnyObject, b: AnyObject) -> AnyObject {
        // do the work & return an object
    }

    // do stuff

    let foo = reusableFunction("one", "two")

    // do more stuff

    let bar = reusableFunction(foo, "three")

}
```

Nil Coalescing

- Objective-C & Swift both allow **`nil`** values
    - Swift does this through the use of optionals
- If/else control flow is useful for checking **`nil`** for assignment purposes
- Can be shortcutted with the *ternary operator* (borrowed from C)
    - **`condition ? true expr : false expr;`**

Nil Coalescing

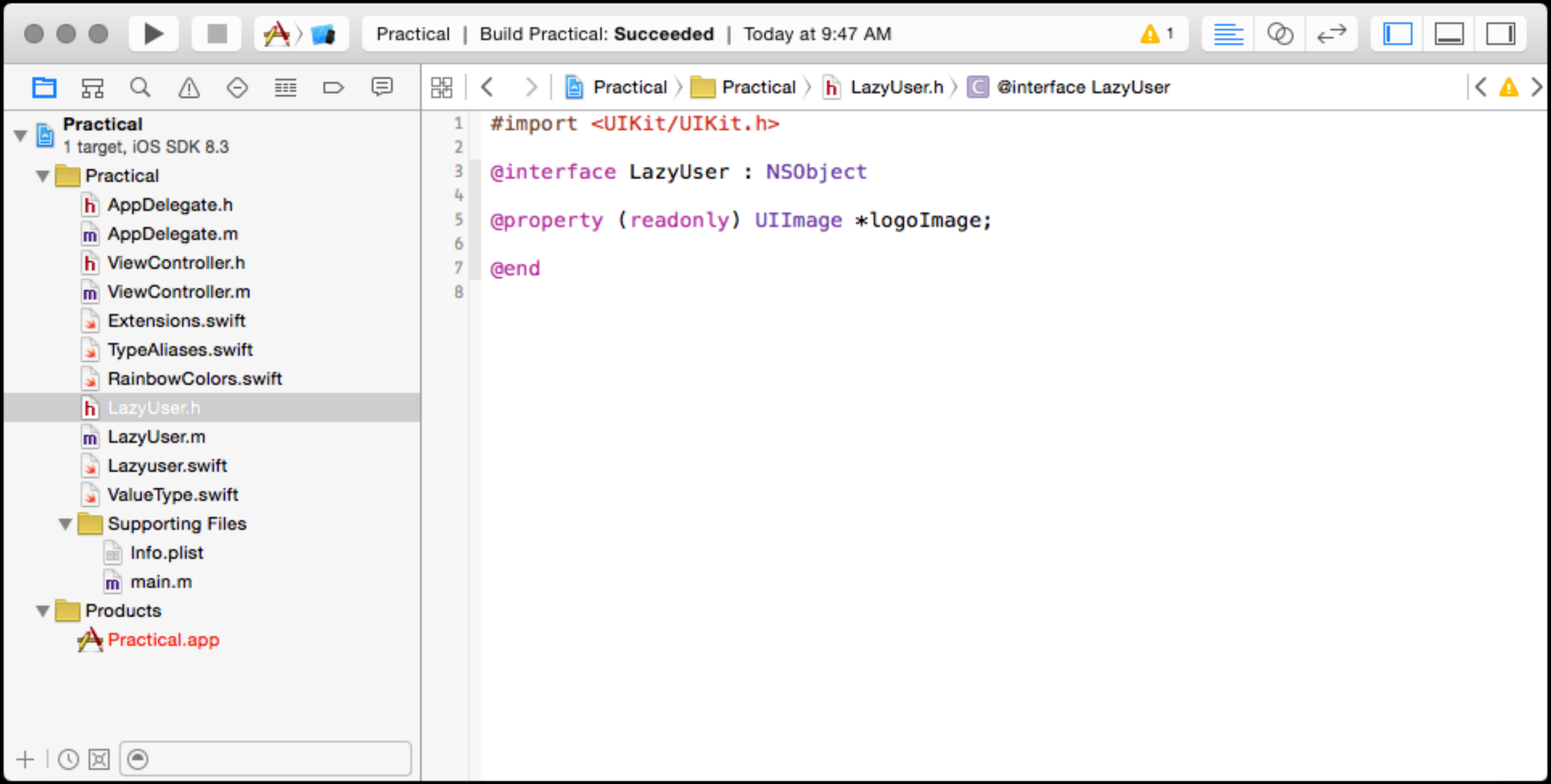
- Ternary operator exists in Swift, but what about optionals?
    - `var bar: AnyObject? = nil`
    - …
    - `foo = (bar != nil ? bar! : someDefault)`
- Instead:
    - `foo = bar ?? someDefault`
- Works like JavaScript's `||` operator, except testing `nil` instead of truth
    - `foo = bar || someDefault;`

Lazy Loading

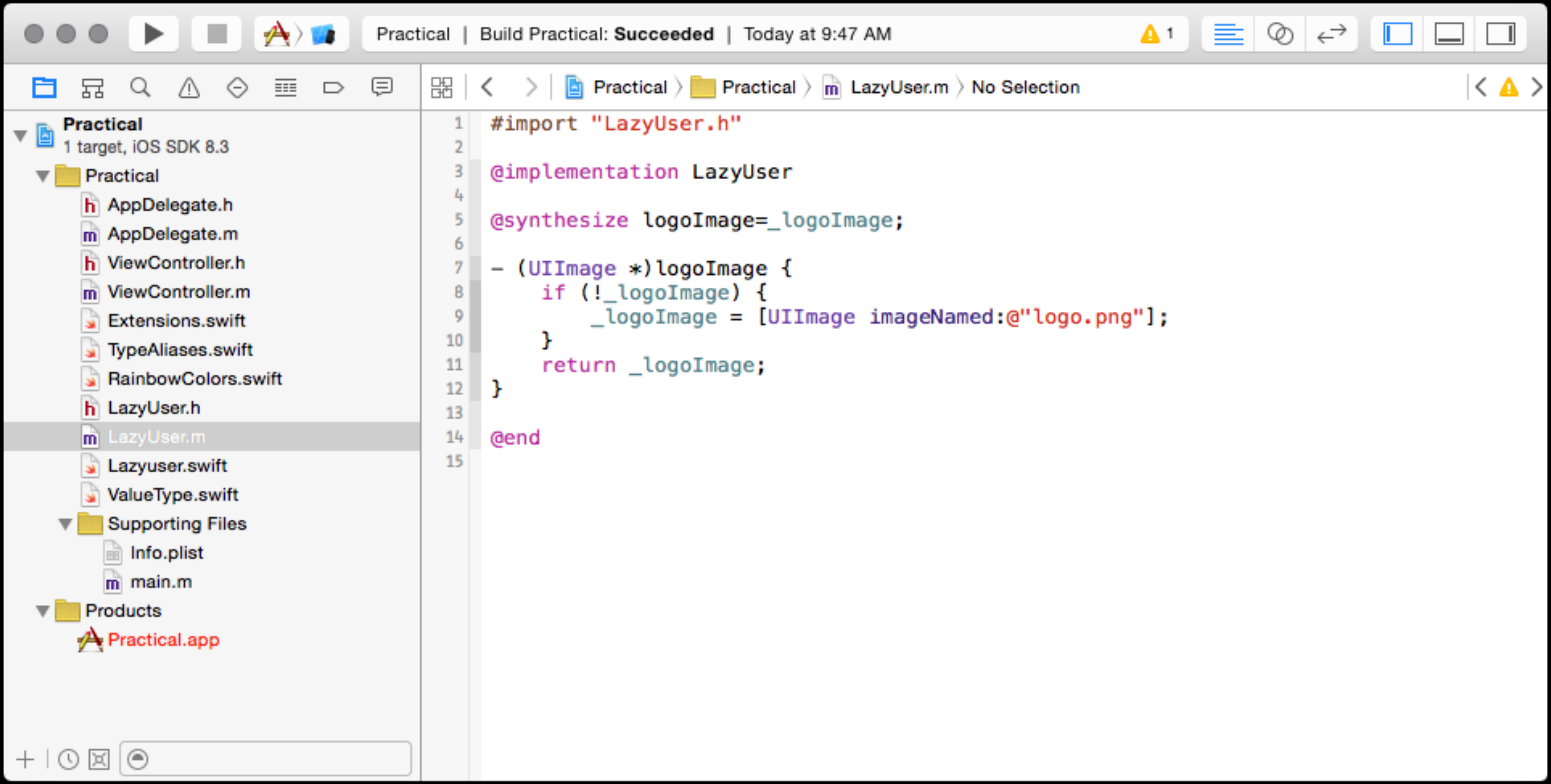
- Not creating the overhead of a variable until the first time it’s used
- In Objective-C, we do this with properties backed by instance variables (ivars)
  - **`@property id foo;`**
  - **`@synthesize foo=_foo;`**
  - Later, setup **`_foo`** if necessary, but always return it

Lazy Loading

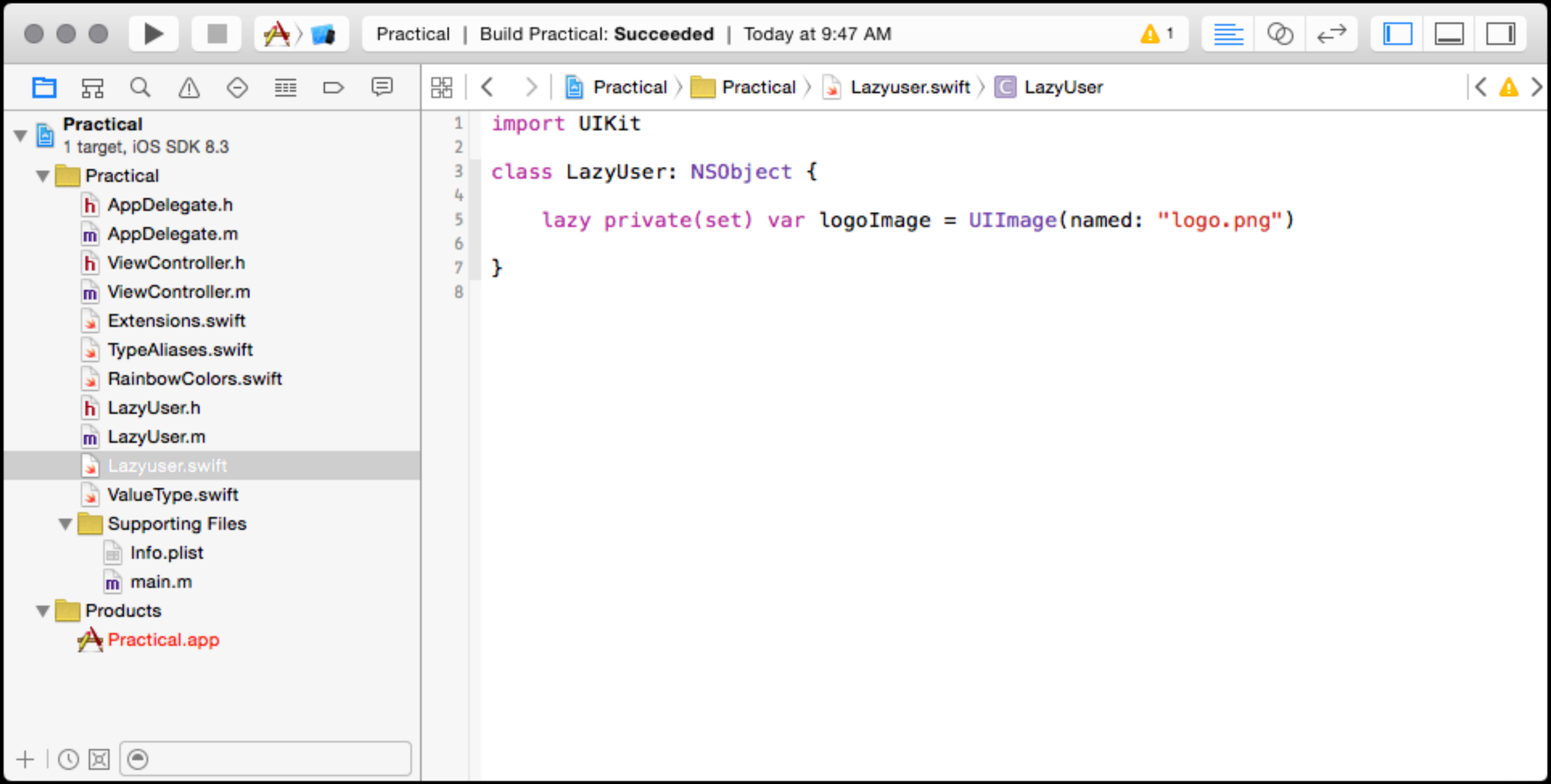
- In Swift, we can get *lazy*
- Technically, “lazy stored properties”
- **`lazy var foo: AnyObject =`** …



```
#import <UIKit/UIKit.h>

@interface LazyUser : NSObject

@property (readonly) UIImage *logoImage;

@end
```



```
#import "LazyUser.h"

@implementation LazyUser

@synthesize logoImage=_logoImage;

- (UIImage *)logoImage {
    if (!_logoImage) {
        _logoImage = [UIImage imageNamed:@"logo.png"];
    }
    return _logoImage;
}

@end
```



```swift
import UIKit

class LazyUser: NSObject {

    lazy private(set) var logoImage = UIImage(named: "logo.png")

}
```

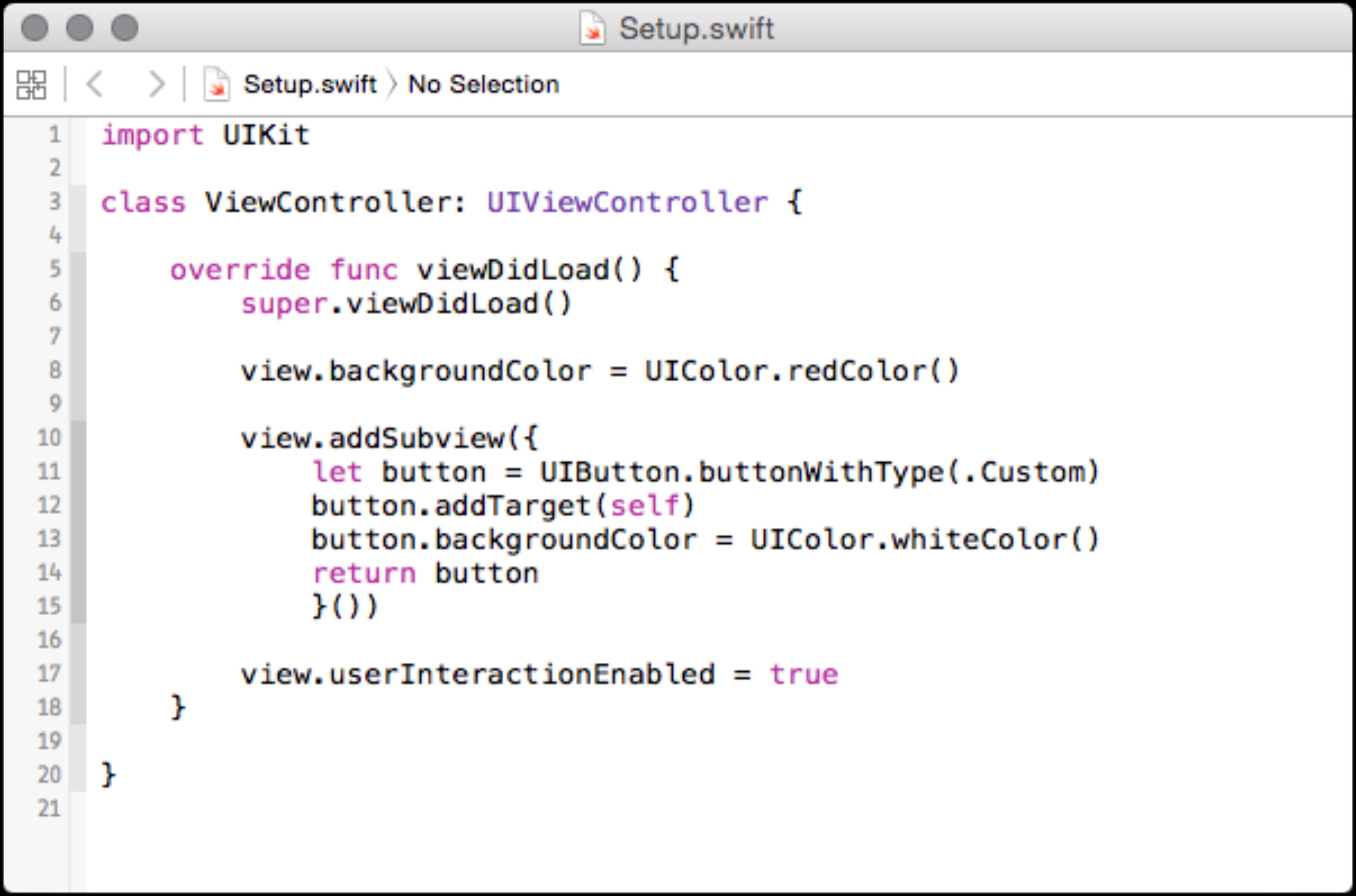
Closure Paradise

- Closures, a.k.a. blocks, lambdas, callbacks, anonymous functions (sorta)
- “Unified with function pointers”
    - Unlike Objective-C, functions are first-class objects, meaning they can be passed
- Essentially, a way to pass around code *in* code

Closure Uses

- Great as trailing arguments to functions
  - e.g., Do some heavy lifting work, then call this code, *kthxbai!*
- I like them for setup of more-than-trivial variables

Closures During Init



```swift
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = UIColor.redColor()

        view.addSubview({
            let button = UIButton.buttonWithType(.Custom)
            button.addTarget(self)
            button.backgroundColor = UIColor.whiteColor()
            return button
            }())

        view.userInteractionEnabled = true
    }

}
```

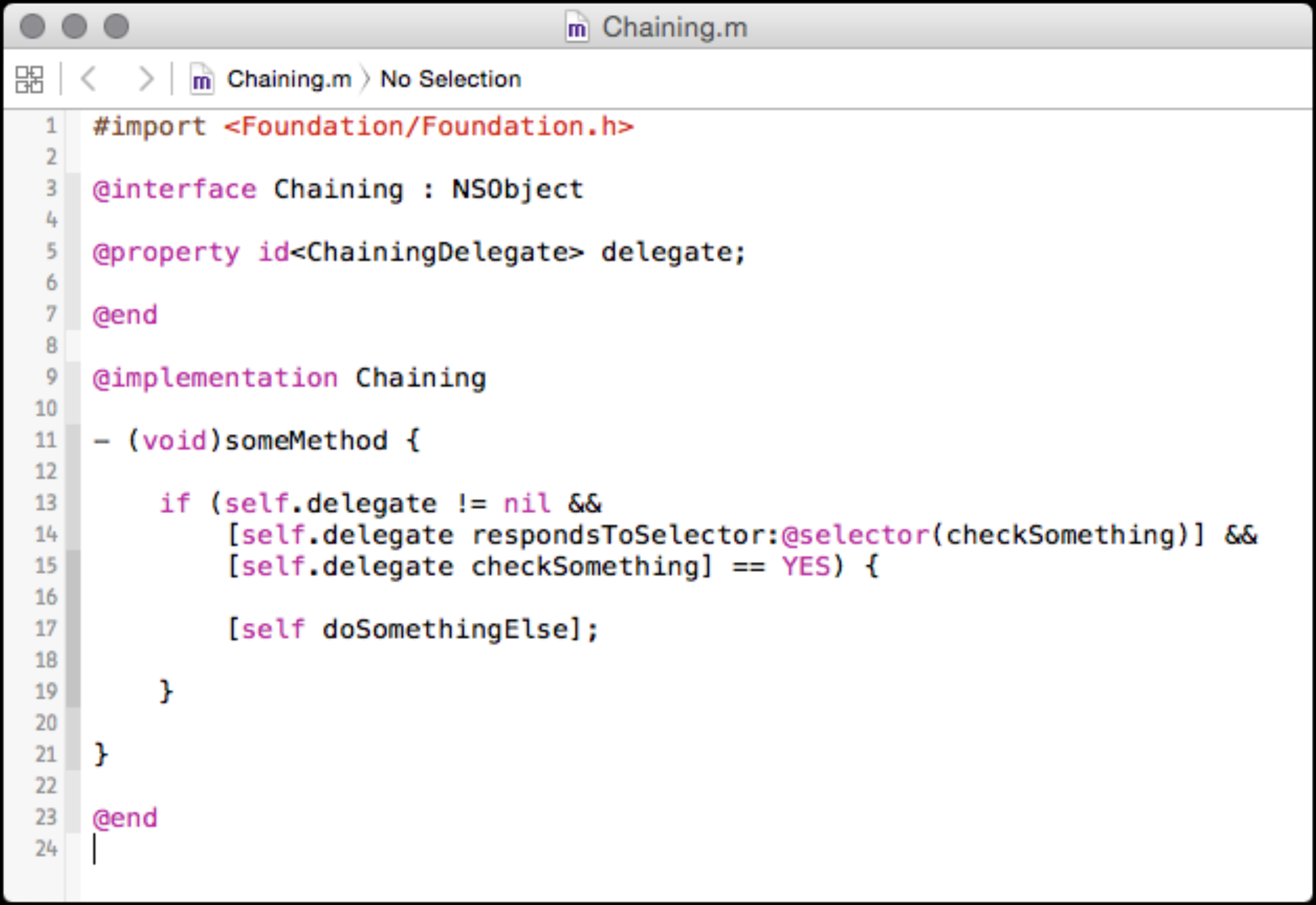
“Immediate Closures”

- `{ … }()`
- You’ll see this in C++ (modern versions) as well
- You can think of it as calling **`foo()`**
  - No arguments
  - **`foo`** contains code; so does **`{ … }`**
  - Do this right now!
- BTW: does this concept have a name?

Optional Chaining

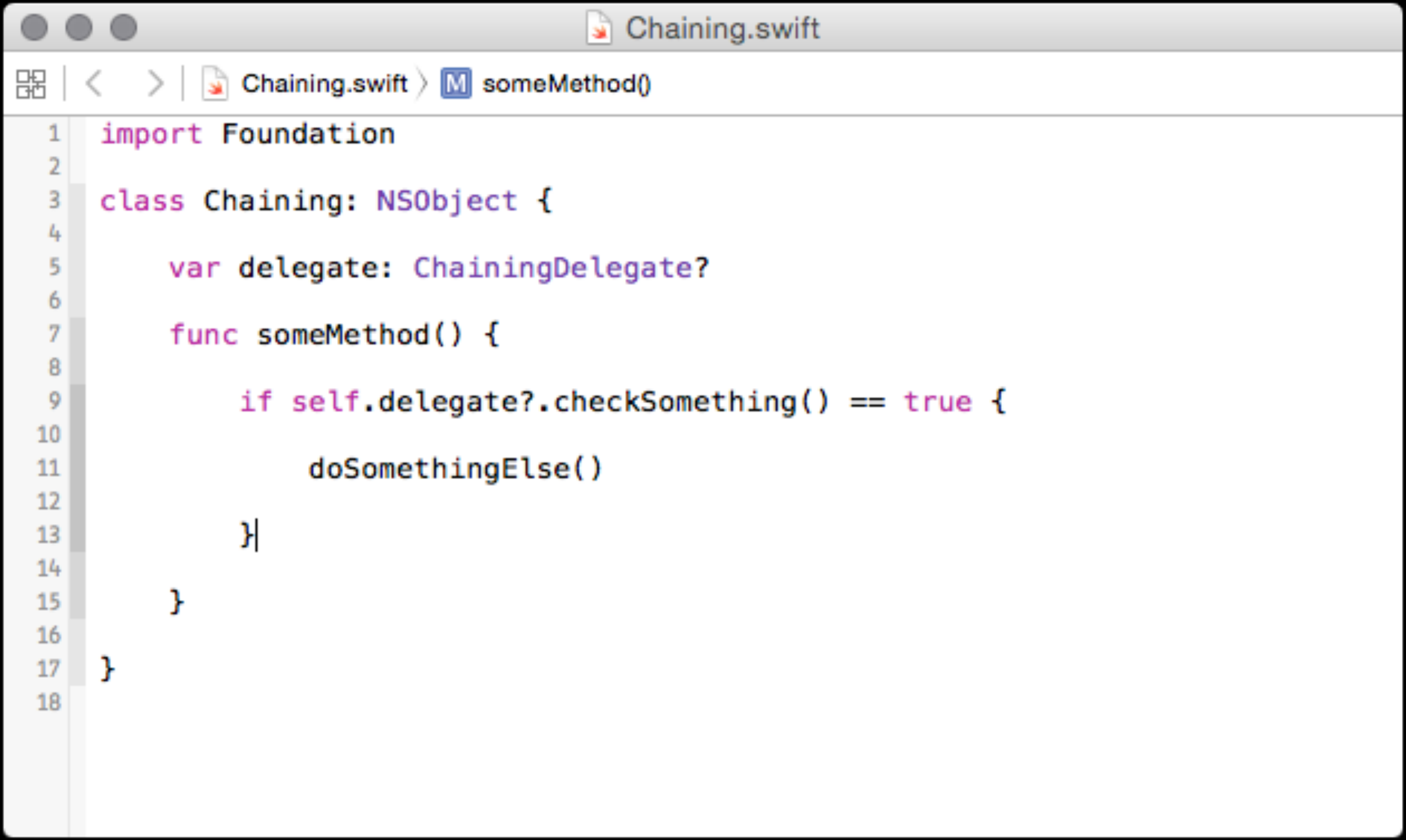
- Solves the (pretty common!) problem of having:
  - Optional properties (like a delegate)
  - With optionally-implemented methods
  - That return a variety of types

Optional Chaining



```objc
#import <Foundation/Foundation.h>

@interface Chaining : NSObject

@property id<ChainingDelegate> delegate;

@end

@implementation Chaining

- (void)someMethod {

    if (self.delegate != nil &&
        [self.delegate respondsToSelector:@selector(checkSomething)] &&
        [self.delegate checkSomething] == YES) {

        [self doSomethingElse];

    }

}

@end
```

Optional Chaining



```swift
import Foundation

class Chaining: NSObject {

    var delegate: ChainingDelegate?

    func someMethod() {

        if self.delegate?.checkSomething() == true {

            doSomethingElse()

        }

    }

}
```

Recap

- Swift is easy to start dabbling with piecemeal
  - Easy integration into Objective-C apps
  - REST services
  - Command-line
- Swift has got some language features that'll do you good
  - Type aliases & nested functions, including locally
  - Nil coalescing & optional chaining to wrangle **`nil`**
  - Lazy loading & closures for brevity & efficiency

Discussion

Thank You!

- @incanus77
- justin@mapbox.com
- https://github.com/mapbox
- mapbox.com/blog