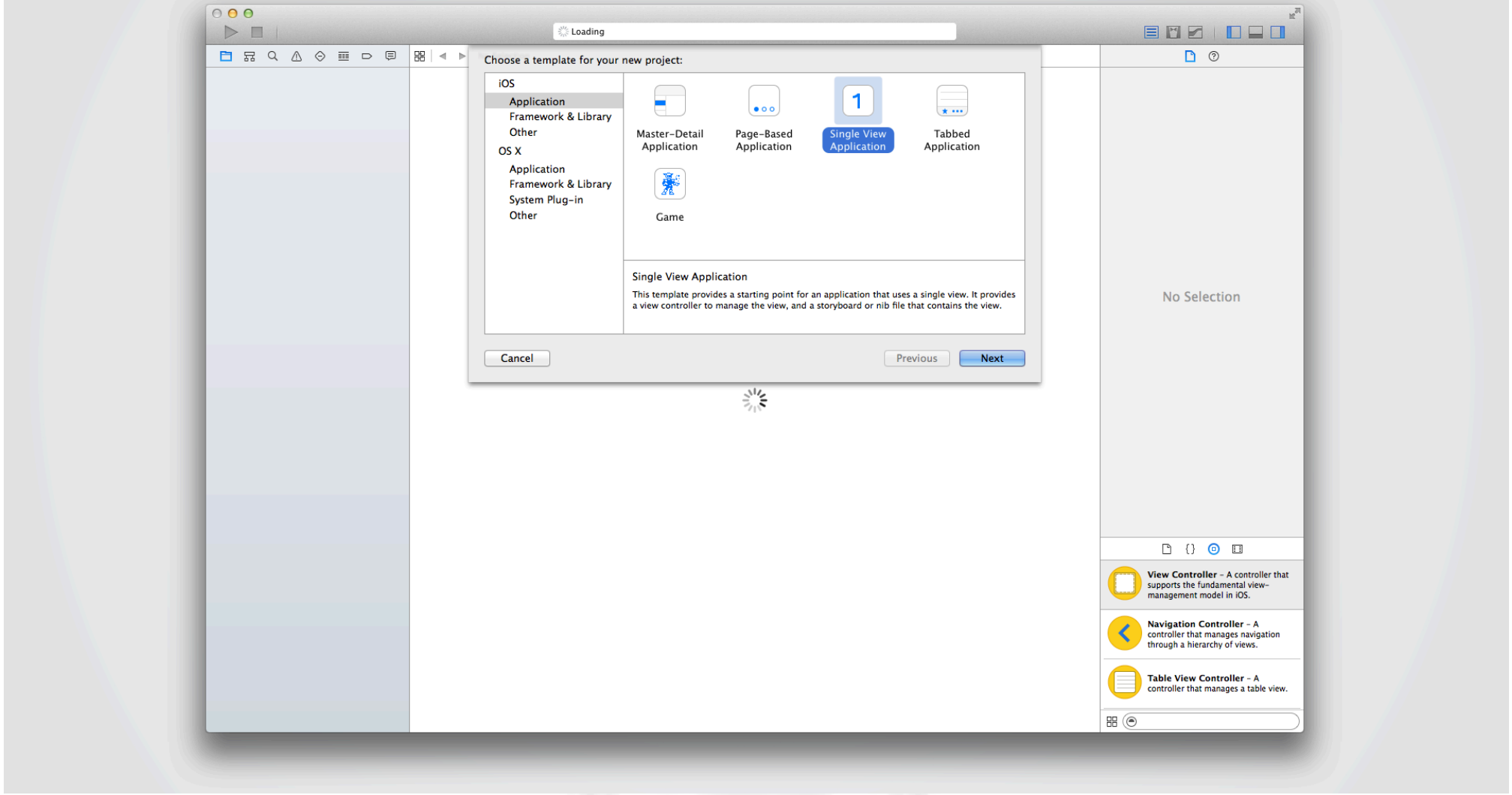
iPhone Development

Justin Toyama

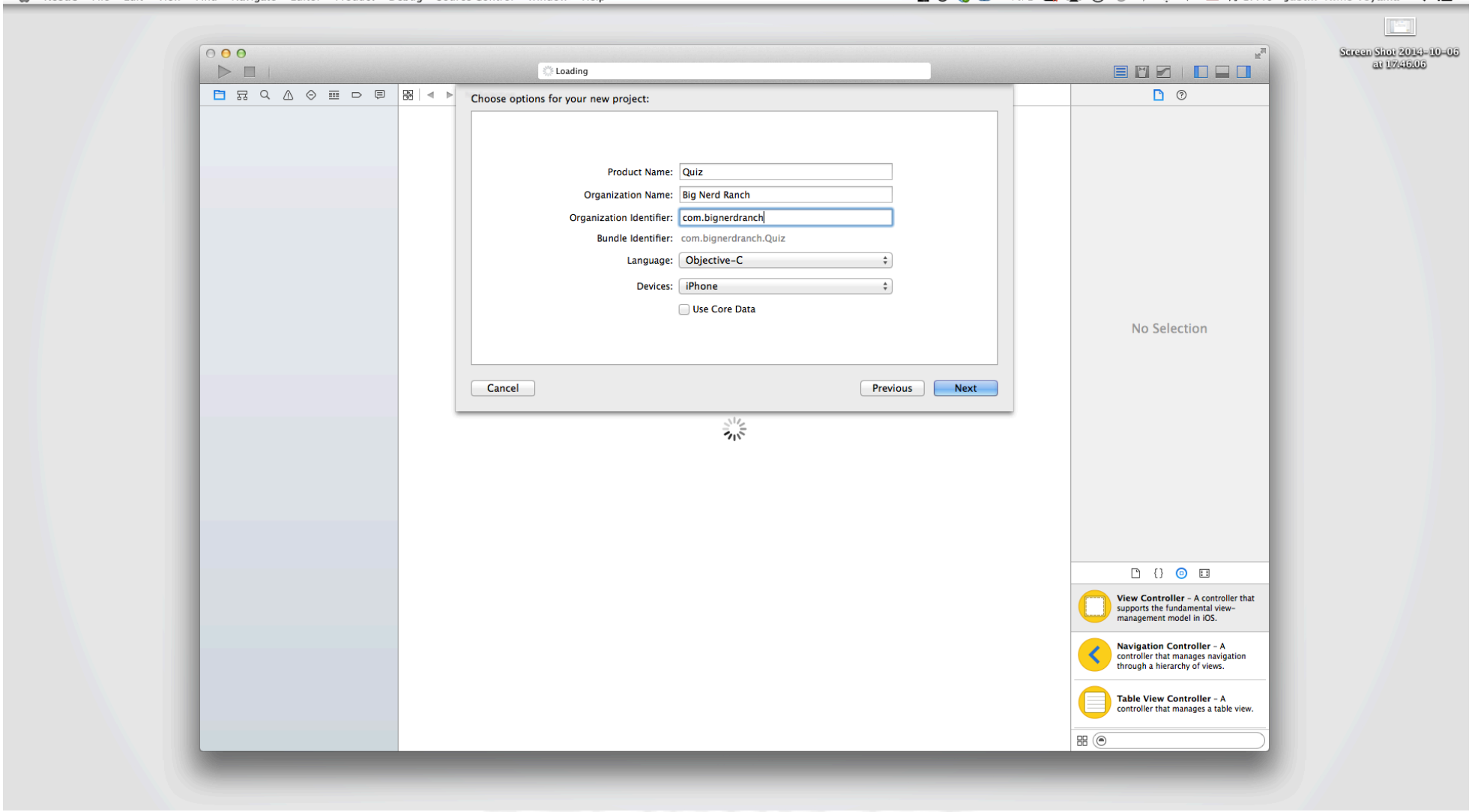
xCode 6 Updates
Intro to Swift with Example

Starting a new project



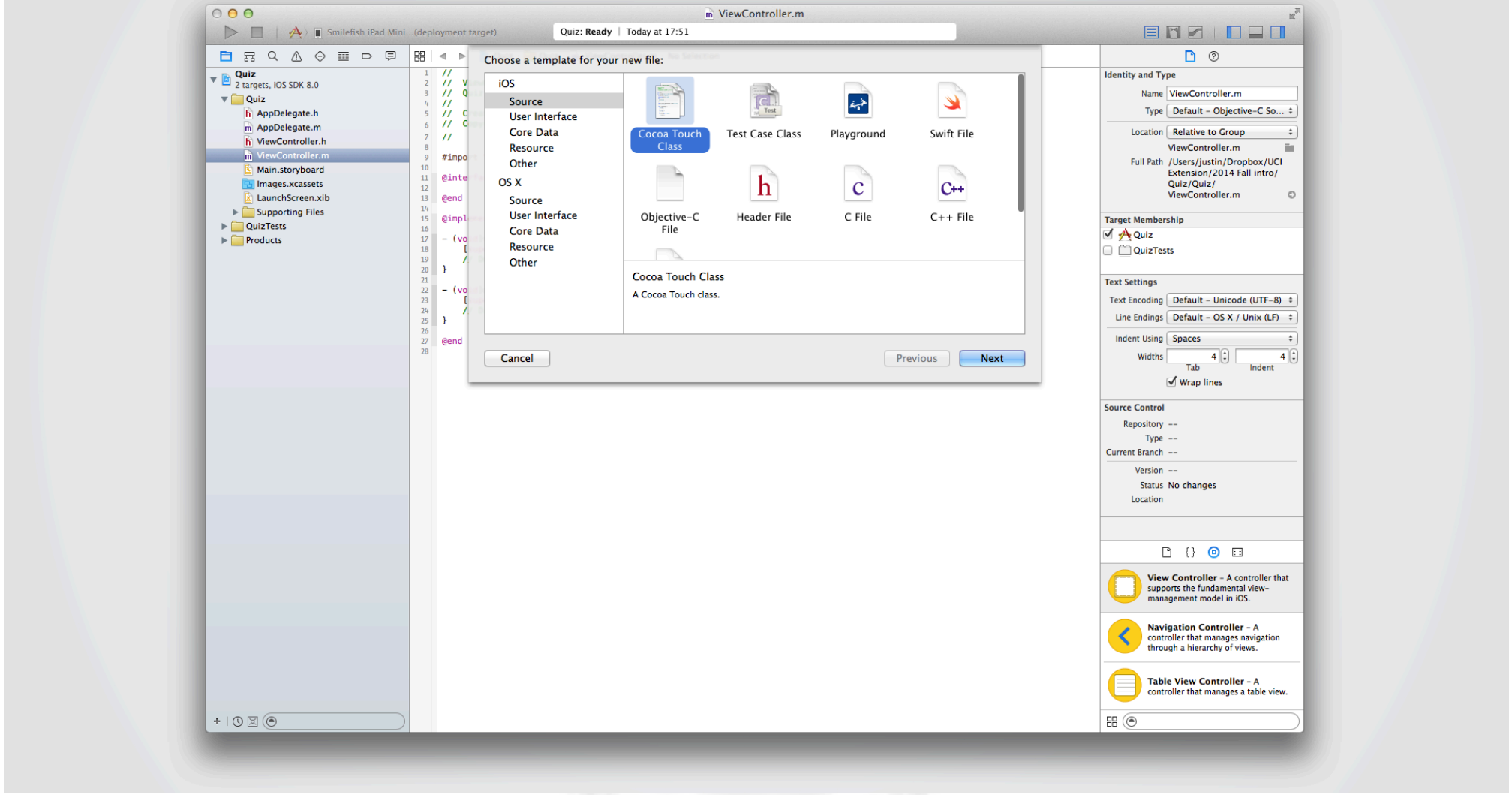
Select the Single View Application Option (Empty application now in the ‘Other’ tab.)

Starting a new project



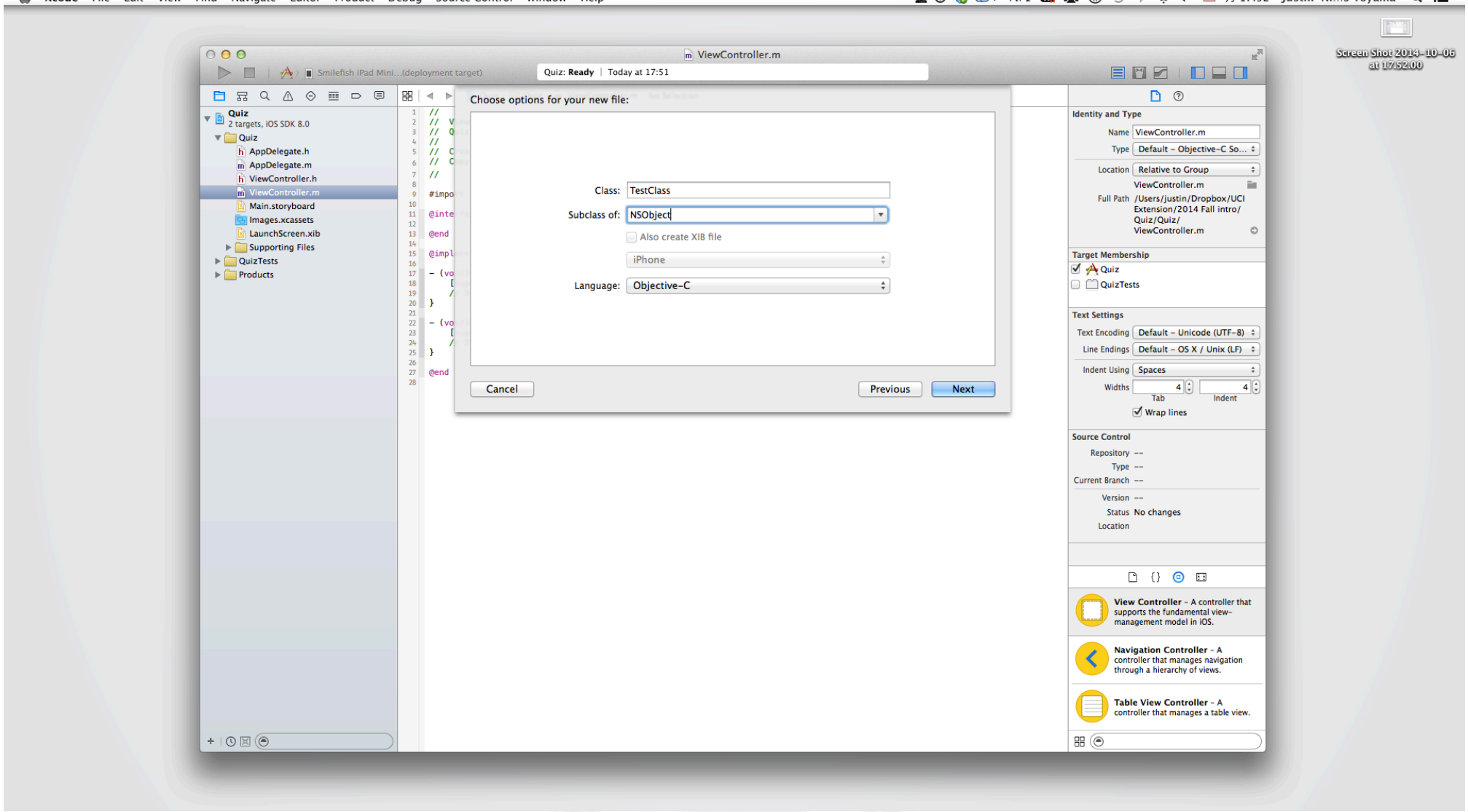
- Don't worry about the Class prefix (but when you create files, prefix manually)
- Select Swift

Starting a new project



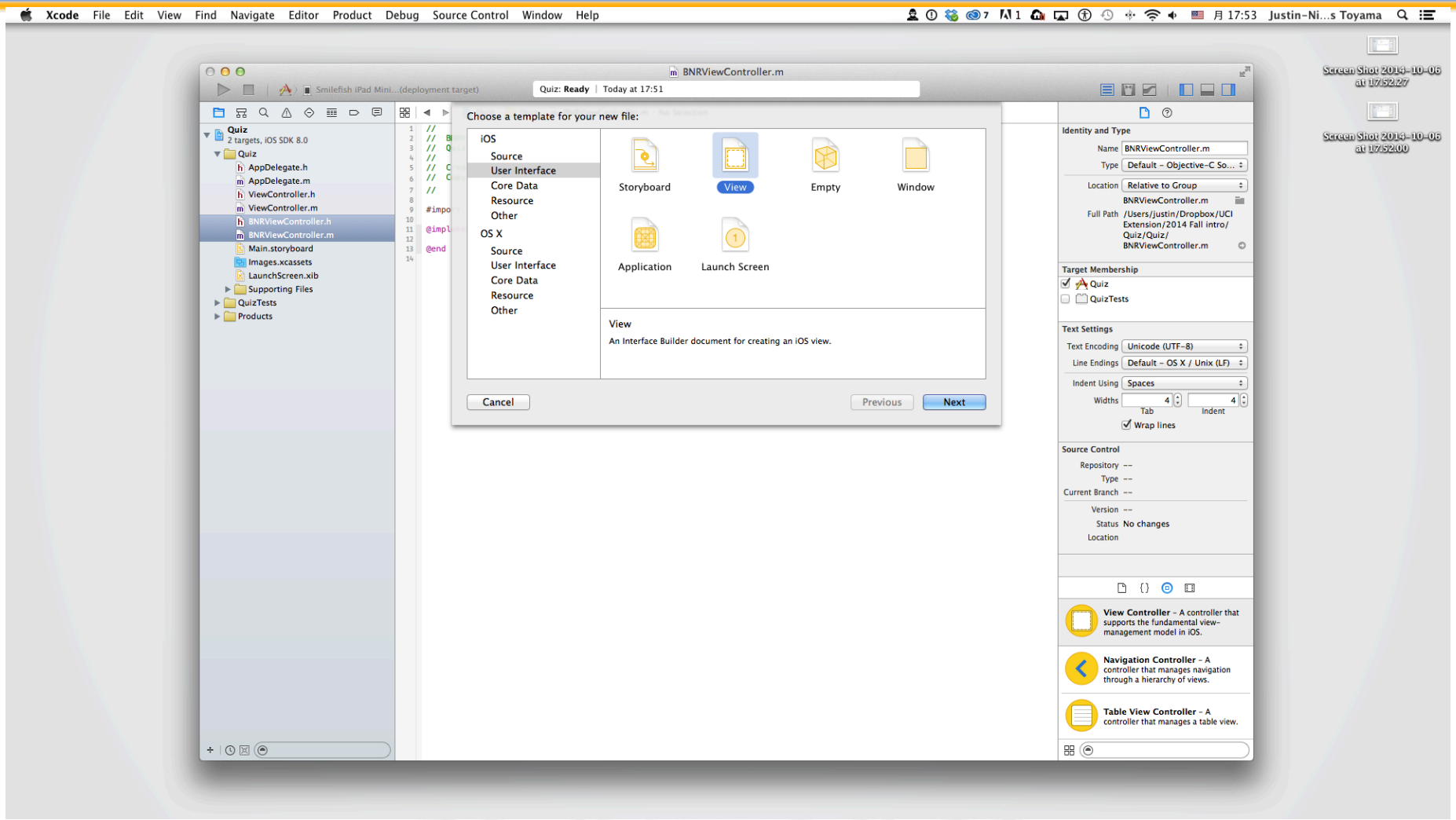
File > New… > Cocoa Touch Class

Starting a new project



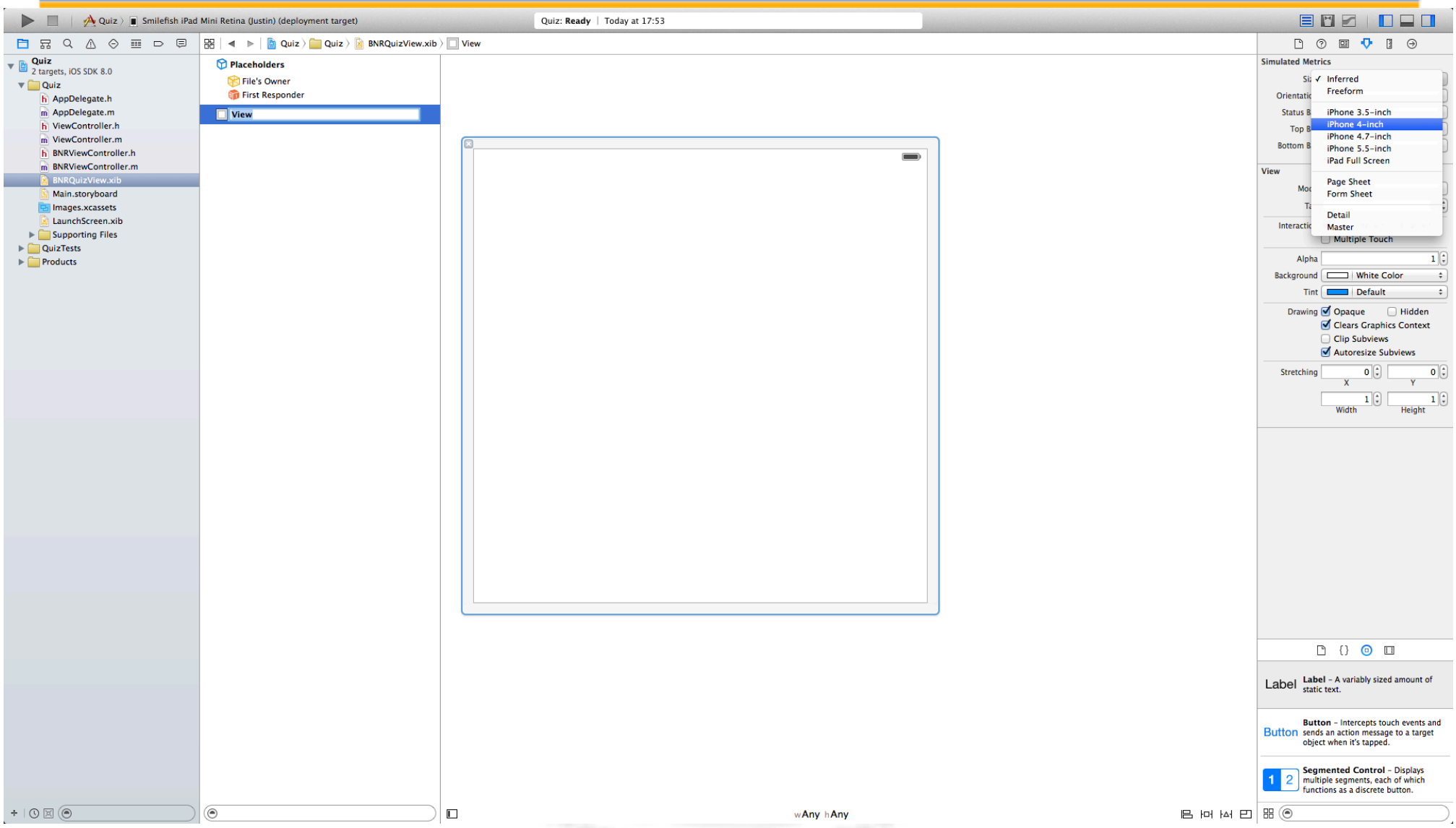
Set the base class (aka “the class from which to inherit” aka “subclass of:”)

Adding a view



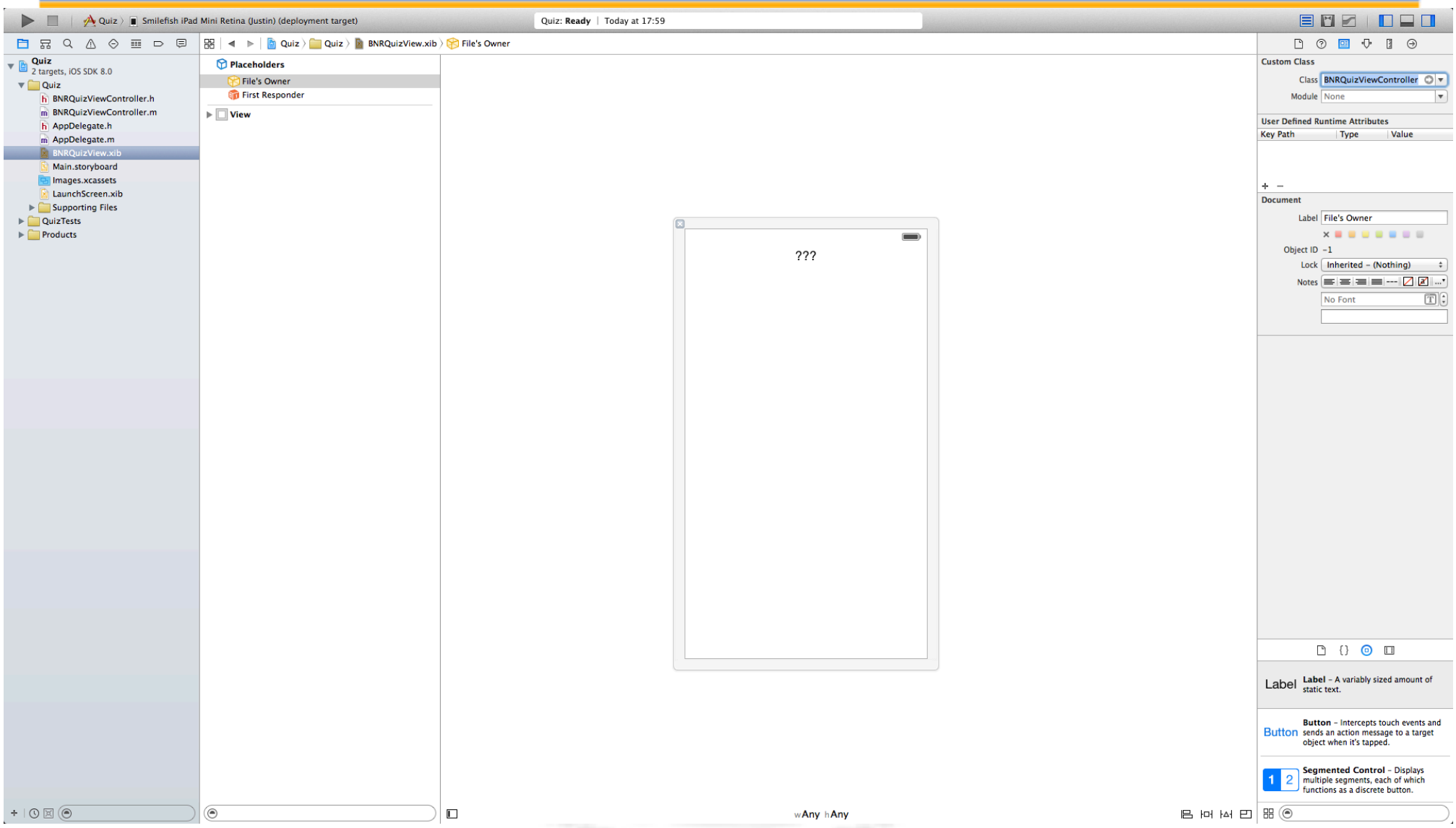
- Select File > New > View.
- Link to custom subclass

Starting a new project



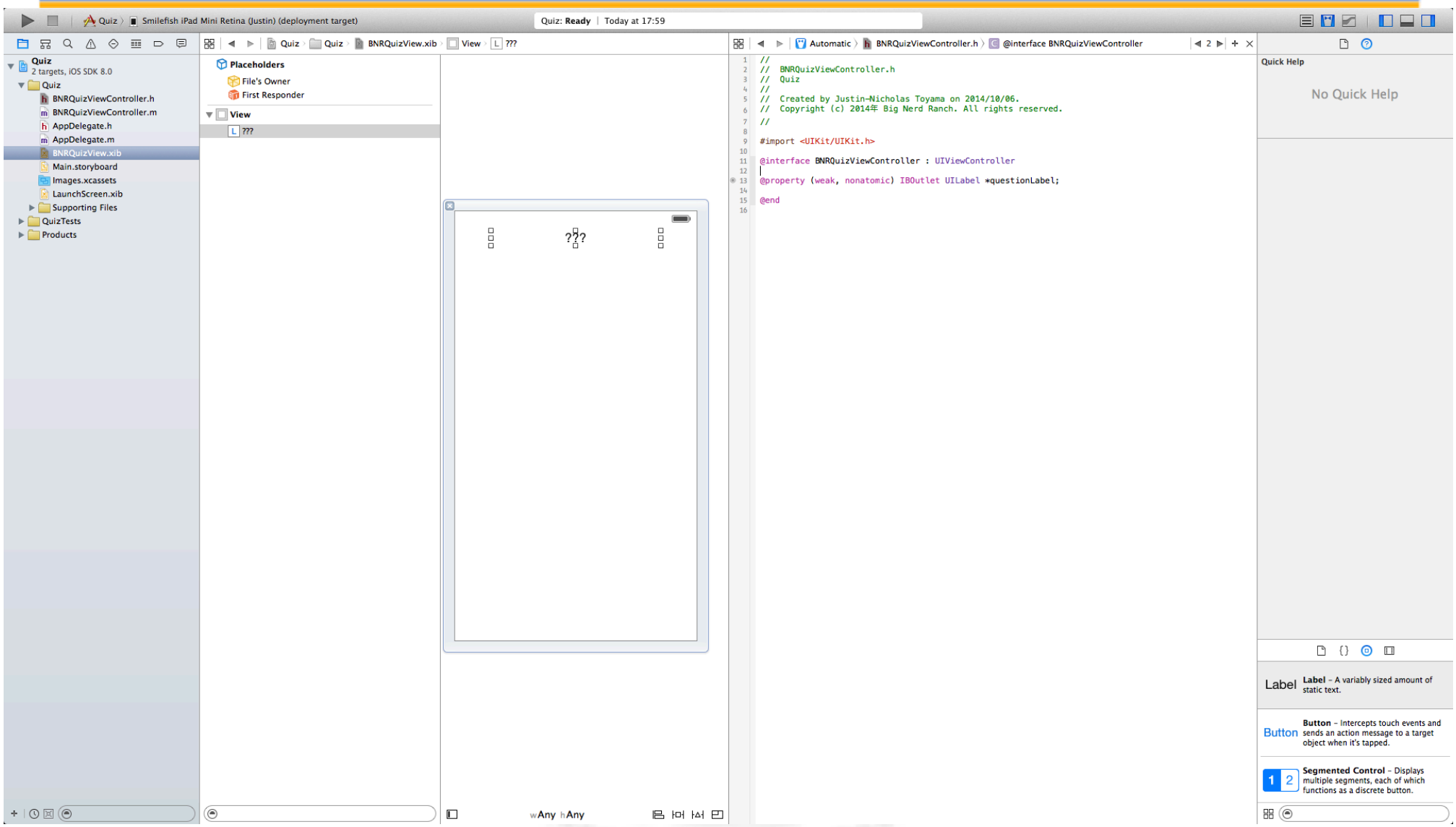
- Be sure to set the Simulated Size (but we will eventually use auto layout…)

Starting a new project



Select the File's Owner object and declare the custom subclass

Creating XIBs



Now you can link up your UI to your code

Xcode 6 and Swift

- Get Xcode 6 from the Mac Appstore
  - Allows you to code and compile Swift code
  - Backwards compatibly with Obj-C projects
  - Introduces Playgrounds – see your code output “live” without building or running

- Swift Resources
  - The Swift Programming Language (iBooks) (read the tour)
    - Also online (free): https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/Swift_Programming_Language/index.html
  - Using Swift with Cocoa and Objective-C (iBooks)
    - Also online (free):https://developer.apple.com/library/prerelease/ios/documentation/Swift/Conceptual/BuildingCocoaApps/index.html#//apple_ref/doc/uid/TP40014216

Introduction to Swift

- Swift closely parallels Obj-C – we will explore the Swift Programming Language over the next 5 weeks while we **focus on learning the APIs and functionality provided by the iOS SDK.**
  - Provides its own versions of all fundamental C and Obj-C types (`Int, Double, Float, Bool, String, Array, Dictionary`)
  - Introduces advanced types (tuples – multiple return values)
  - Introduces optional types (aka "optionals") to protect from runtime exceptions caused by `nil`s
  - Type-safety (strictly and statically typed) is stressed by the language and the compiler but you don't have to specify type if the type is expected/can be inferred
  - Importing the Foundation framework or UIKit maps Obj-C classes to Swift classes "automagically". For example NSString works nicely with String.

```
import Foundation
Class MyViewController : UIViewController {
func describeMyself(name:String) -> String {
 return "My name is \(name)."
}
let myLabel = UILabel(frame: CGRectZero)
myLabel.text = describeMyself("Justin")
println(myLabel.text)
}
```

Let's Compare

Obj-C

```
int myVariable = 42;

//---------------------------------------

const int myConstant = 50;

//---------------------------------------

UITableView *myTableView = [[UITableView
alloc] initWithFrame:CGRectZero
style:UITableViewStyleGrouped];

myTableView.backgroundColor = [UIColor
darkGrayColor];

NSLog(@"%@",
myTableView.backgroundColor];

[myTableView reloadData];

////
```

Swift

```
var myVariable = 42

//---------------------------------------

let myConstant = 50

//---------------------------------------

let myTableView =
UITableView(frame:CGRectZero, style:
.Grouped)

myTableView.backgroundColor =
UIColor.darkGrayColor()

println(myTableView.backgroundColor)

myTableView.reloadData()


////
```

cheat sheet

Swift

```
var myVariable = 42

//----------------------

let myConstant = 50

//----------------------

let myTableView = UITableView(frame:CGRectZero,
style: .Grouped)

myTableView.backgroundColor =
UIColor.darkGrayColor()

println("The color is" +
myTableView.backgroundColor)

myTableView.reloadData()
```

type is inferred; no *

Table view will not be modified, so we use let

no semicolons! =)

no need to call alloc init

method arguments in parentheses, separated by comma

reference constants by dot syntax (type inferred)

Still use parentheses even if no arguments (this makes it a method call)

- With just this much, we can create a view hierarchy, access properties, get and set the values of variables we define…in other words, 90% of your life as a developer
- As we go through the advanced course, I will pepper the slides with the Swift way of doing things

SwiftQuiz Sample

- In this application, we will use Swift to build a very simple app that displays a question and the corresponding answer.
- This will let us study a little Swift code and also get familiar with Interface Builder so that we can start making more sophisticated view hierarchies next week.
- Implementation:
  - We will create two arrays – one to hold the questions and another to hold the answers
  - We will generate a simple user interface with a label to display the question and a label to display the corresponding answer when the appropriate button is pressed

Homework

- Read the 'Welcome to Swift' section from The Swift Programming Language
- Read 'Getting Started' from Using Swift with Cocoa and Objective-C
- This is enough for you to start a simple project in Swift or perhaps convert an existing project to Swift
- Some things may go over your head but we will cover the material over the next 5 weeks, so don't worry. Some topics—like blocks or closures—will be covered in the advanced class.
- Challenge: write the Quiz sample application from BNR Ch.1 as a Swift project following the video and lecture notes.
- Look at the Tutorials in the Homework Guide