LECTURE #3

Welcome

So far you've been introduced to some UI components and the Swift programming language. This week our focus is on how we can navigate between different view controllers in our application, because most applications have more than one view.

Navigation

In this lesson, we'll see how we can go about navigating to different views in our application. Maybe we want a screen where the user can login to our app, another where the user interacts with their content, another to add new content. These are all common things in the world of iOS, navigating from one screen to another. In programming terms, this is known as navigating from one *view controller* to another. One of the most common forms of navigation in iOS is a navigation controller. An example of a navigation controller is the Settings app, when you select one of the items you transition into another view. We'll be focusing on this component in this lesson.

Basics

Start out by creating a new project in Xcode. It will be another view-based application like we did in the first lesson. Inside *Main.storyboard* click on the view controller, then head up to Xcode's menubar and select *Editor>Embed In>Navigation Controller*. Add a new button to the main view controller. Title the button as "Push View Controller", and connect it with an *IBAction* in *ViewController.swift*. Your projects should now look like Figure 3-1.

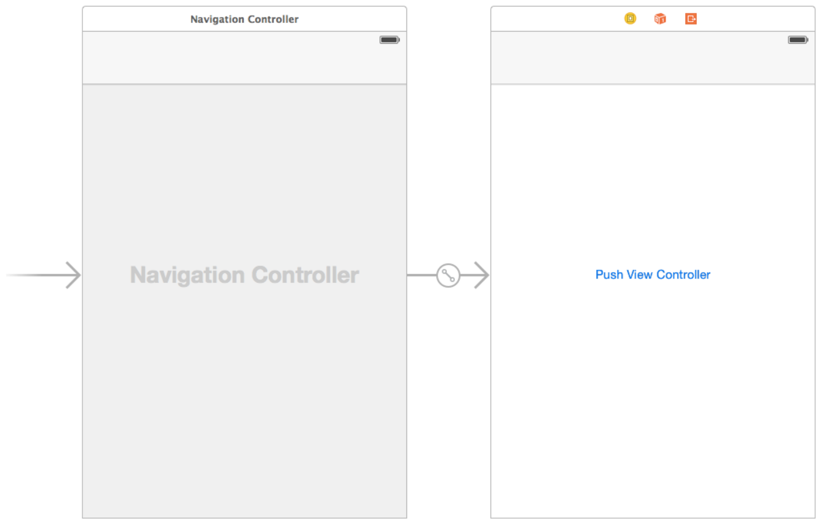
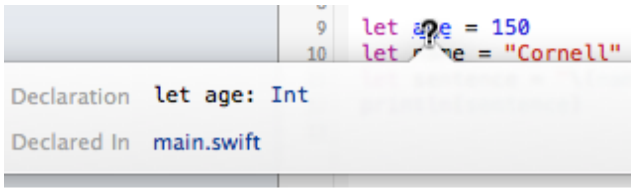


Figure 3-1

What we've setup is a navigation controller to load up our first ViewController's view, but the idea of this application is to transition from the first view controller to a second view controller. This is what the navigation controller specializes in. We can tell the navigation controller to "push" on another view controller and it will then transition from the current view controller to the new view controller. The reverse of this is to "pop" off the newly added view controller to return to the original.



```
let age = 150
let name = "Cornell"
```

Figure 3-2

Let's add the second view controller that we want to transition to. In the *Object Library*, drag out a new *View Controller* onto the storyboard. On the first view controller, control-drag from the "Push View Controller" button to the second view controller we just created. You'll be prompted to create a new *segue*, select the *show* segue type (see figure 3-3).

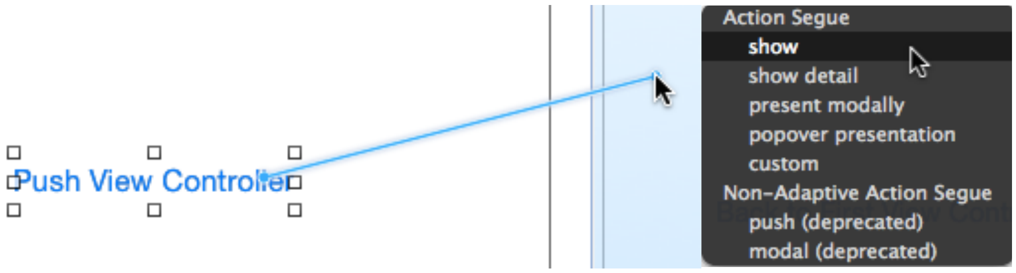


Figure 3-3

You've just created a fully functional navigation controller with 2 view controllers. Build and run the application and see that you can tap the button to go to the second view and select the "Back" button in the navigation bar to return to the first view. That's it!

Presenting Modally

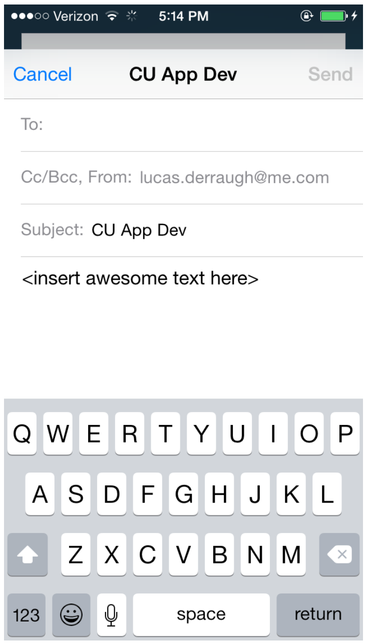


Figure 3-4

Sometimes we want to present the user with a modal view. A modal view is one that has to be "completed" before continuing to use the rest of the app. A perfect example of this is in the Mail app. When you create a new message, you're presented with a view that you enter the recipient, subject and message content. This view is animated in from the bottom of the screen (characteristic of presenting modal views). If you wanted to go back to look at your inbox, you have to either send or cancel your message. This compose view is a modal view because it is presented and must be completed (saved or canceled) to return to the rest of your application.

Let's go back to the application we were making with the navigation controller and add in a new button on the first view controller called "Present Modal View Controller". Also add in a new view controller to the storyboard. Similar to how we created the show segue in the navigation controller example, we want to control-drag from the "Present Modal View Controller" button to the new view controller (see Figure 3-5).

Build and run this app and test out presenting the new modal view controller. By tapping on the "Present Modal View Controller" button you'll see the new view controller animate in from the bottom and fill the screen. However, there is no way to dismiss this view controller, we're now stuck in this modal view. Let's fix this.

Back in the storyboard, let's add a new button to our modal view controller and call it "Dismiss View Controller".

Go into the *ViewController.swift* and add the following method:

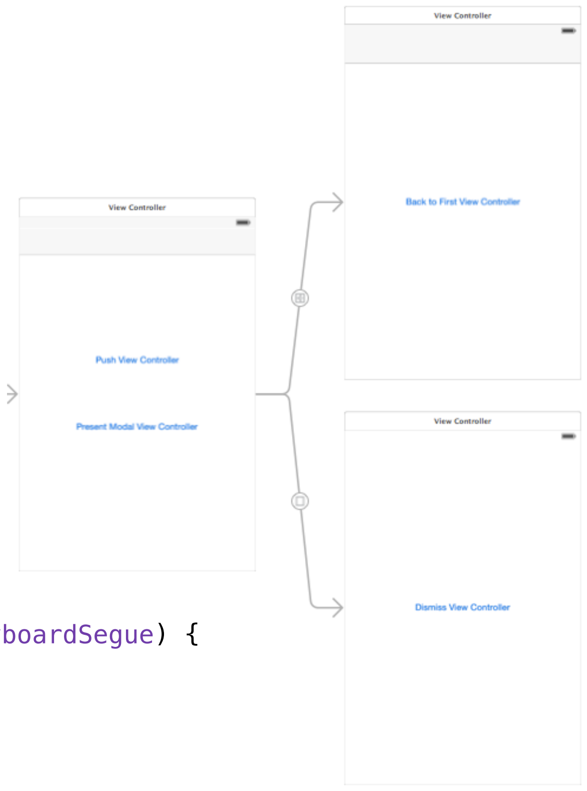
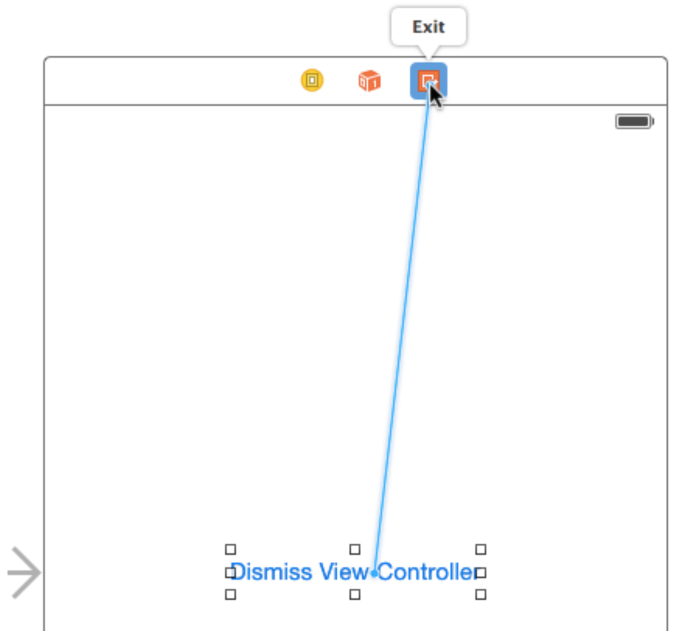


Figure 3-5

```
@IBAction func unwindToMainView(sender: UIStoryboardSegue) {

}
```

We've seen this type of code before, but let's talk about what's going on here. **`@IBAction`** before a method indicates that we can connect this in the storyboard. We are defining a method called **`unwindToMainView(sender:)`** and the parameter type is a **`UIStoryboardSegue`**. The method name isn't required to be written exactly as is, but it's informative. We want to use this call for an *unwind segue* and the sender must be a **`UIStoryboardSegue`** because this tells the storyboard file that we can hook in with this method from a segue. By adding this method to your first view controller (*ViewController.swift)*, you can now unwind from anywhere in your application, back to this main view controller.



Back in *Main.storyboard*, we can control-drag from the "Dismiss View Controller" button we just created to the view controller's *Exit* on our storyboard. This can be done as shown in Figure 3-6.

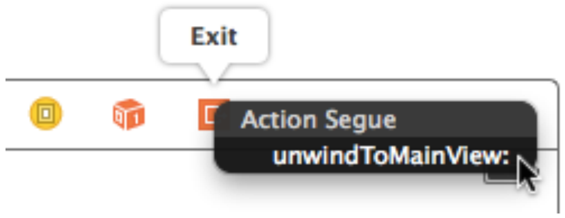


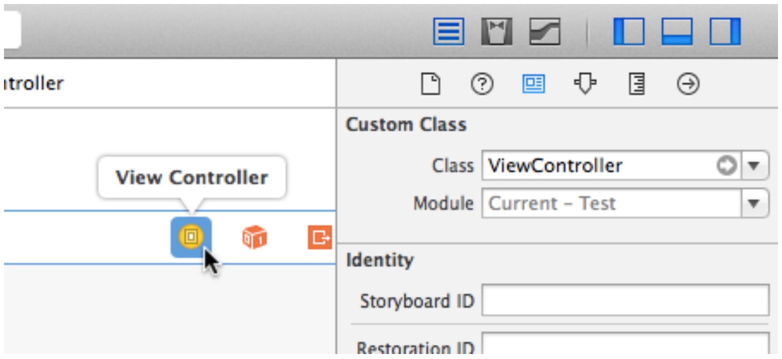
Figure 3-6

That's all you have to do! Build and run, and watch as your modal view controller is presented and dismissed with animation ease.

Subviews

It may seem that there is very little code to write in iOS, and for the most part, Apple does try to keep coding to a minimum. However, there are times where you will need to code things and that time starts now. Let's create a new class so that we can begin coding. Head up to Xcode's menubar and select *File>New>File…*, when the panel appears select under the *iOS>Source* heading, *Cocoa Touch Class*. Call this class "ModalViewController" and make it a subclass of *UIViewController* (make sure the language selected is Swift). Hit *Next* and then *Create*, and now you've created your own Swift file that contains a subclass of *UIViewController*.

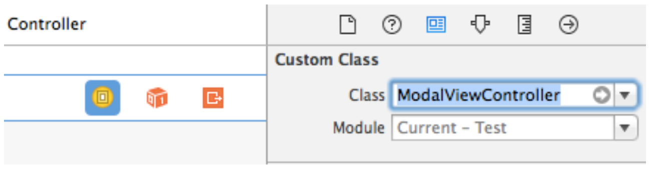
The first view controller that you have when you create a new Xcode project is a subclass of *UIViewController* called *ViewController*. This is controlled inside the *ViewController.swift* file. To verify this, select our main view controller in our storyboard and click on the identity inspector in the Utilities panel, you will find that it is the *ViewController* class.



Class: ViewController
Module: Current – Test

Figure 3-7

When we created the new *ModalViewController.swift* file, we also created our own class called *ModalViewController* that can act as a view controller (because it is a subclass of *UIViewController*). Let's select the view controller that appears modally (the last view controller we added) in our storyboard and change its class from *UIViewController* to be *ModalViewController* (Figure 3-8). Now this view controller will have all the properties of a *ModalViewController* class.



Class: ModalViewController
Module: Current – Test

Figure 3-8

Head on over to our newly created *ModalViewController.swift* file. The job of a view controller is to manage the views displayed on the particular view it manages. All of the view controllers we see in our storyboard manage the view that fills the screen of the iOS device. To get a better understanding of this, let's try out some code. Alter the **`viewDidLoad`** method to look like the following:

```
class ModalViewController: UIViewController {
    var greenView: UIView!

    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = UIColor.redColor()
        greenView = UIView(frame: CGRect(x: 50, y: 50, width: 100, height: 200))
        greenView.backgroundColor = UIColor.greenColor()
        view.addSubview(greenView)
    }
}
```

`viewDidLoad` is one of many methods defined in *UIViewController* and it is called when our view is first loaded from the storyboard file. This makes it an appropriate place to initialize any items that need to be set before the view is ever visible. Here we are accessing our view (the main view that this view controller manages) and setting its background color to be red. We can access the **`backgroundColor`** property because our view is a **`UIView!`** (check out the class reference to see what else you can change on UIViews). Then we are creating a new **`UIView`** called **`greenView`** and we give it initial dimensions, set its background color to green, then add it to our view controller's view as a subview. Adding the greenView as a subview to our view means that it will appear on top of that view.

Before you read anymore, be sure to test out the results of this code. When the modal view controller is presented, we see that our background color is red and that we've added a green rectangle to our view. This is of course what we expected.

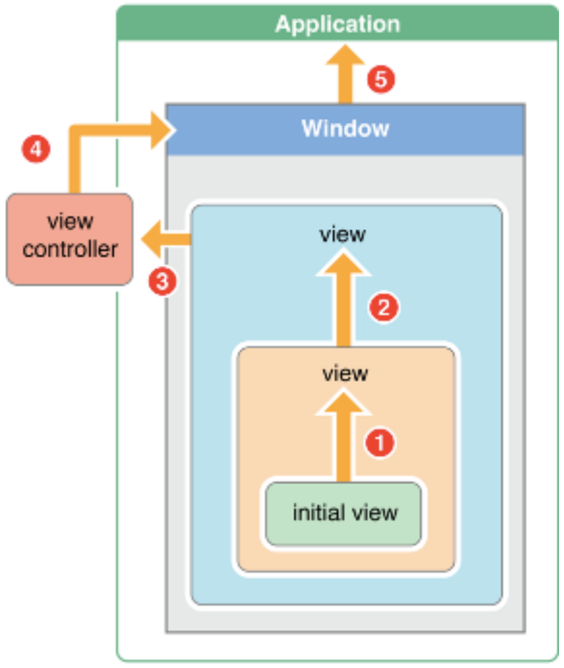


Figure 3-8

What if we wanted to detect when the user taps somewhere on our view? A nice feature of *UIViewController* and *UIView* is that they are subclasses of *UIResponder*, which allows you to respond to touch events. When the user taps the screen, the first item that gets notified is the view that is directly underneath your finger. So in our example, we have our view controller, the view managed by the view controller (currently the red view) and the green view which are all part of the *responder chain*. When we tap on the green view, the UIResponder method **`touchesBegan`** will be called on our green view, if the greenView doesn't override this method (which it does not) then the **`touchesBegan`** event will be forwarded to the superview (in this case, our view controller's view). It also does not implement this method and so it will forward the touches event to our view controller, because it is also in the responder chain. This is useful for us because we can now implement **`touchesBegan`** in our **`ModalViewController`**. To implement this, add the following block of code right after the **`viewDidLoad`** method, then build and run and watch the magic!

```
override func touchesBegan(touches: NSSet!, withEvent event: UIEvent!) {
    greenView.center = touches.anyObject().locationInView(view)
}
```

Cool! Now when we tap anywhere on our view, it moves the green view's center to the point we tapped. By overriding **`touchesBegan`**, we get the set of touches that occurred when the user tapped the screen. In this example we take **`anyObject`** from the set of all possible touches (there is only one in most tapping cases), we get its location in the view, then we set our **`greenView`**'s **`center`** to that location.

Conclusion

We learned a lot in this lesson. We discovered how we can present view controllers using segues and navigation controllers. The difference between a pushing a view controller on by navigation controller or by presenting one modally. Lastly, we addressed programmatically adding our own subviews, changing some view properties, and dealing with the responder chain. With all that said, the only way to truly learn this information is to put it into practice. Doing the project's challenge problems will make all the difference between learning the material vs knowing the material. Aim for knowing!