Designing iPhone Applications

448460
Fall 2016
10/11/2016
Kyoung Shin Park
Dankook University

Overview

- Designing iOS Applications
- Multiple Model-View-Controller (Why and How?)
- Segues
- View Controllers
- Navigation Controllers
- Tab Bar Controllers

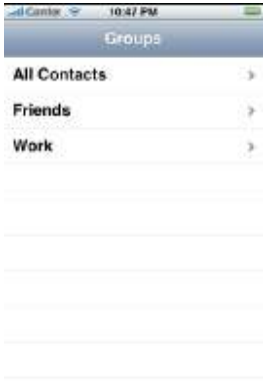
Designing iOS Applications

Organizing Content

- Focus on your user's data
- One thing at a time
- Screenfuls of content

Patterns for Organizing Content

- Navigation Bar
  - Hierarchy of content
  - Drill down into greater detail



- Tab Bar
  - Self-contained modes

Model-View-Controller (Why and How?)

Why Model-View-Controller?

- Clear responsibilities make things easier to maintain
- Avoid having one monster class that does everything
- Separating responsibilities also leads to **reusability**
- By minimizing dependencies, you can take a model or view class you've already written and use it elsewhere
- Think of ways to write less code

Model

- Not aware of views or controllers
- Typically **the most reusable**
- Communicate generically using
  - **Key-value observing**
  - **Notifications**

View

- Not aware of controllers, may be aware of relevant model objects
- Also **tends to be reusable**
- Communicate with controller using
  - **Target-action**
  - **Delegation**

Controller

- Knows about model and view objects
- The brains of the operation
- Manages relationships and data flow
- **Typically application-specific, so rarely resuable**

Application Data Flow

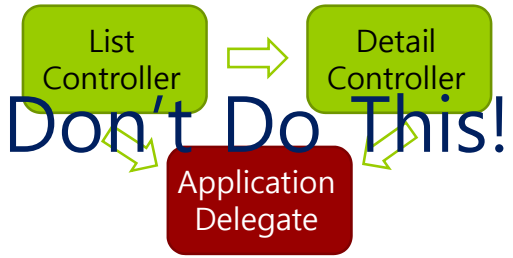
A Controller for Each Screen

Connecting View Controllers

- Multiple view controllers may need to share data
- One may need to know about what another is doing
  - Watch for added, removed or edited data
  - Other interesting events

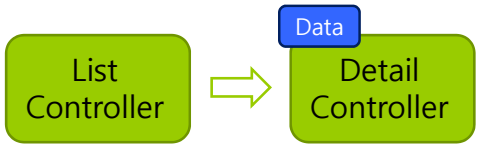
How Not To Share Data

- Global variables or singletons
  - **This includes your application delegate!**
- Direct dependencies make your code less reusable
  - And more difficult to debug & test



Best Practices for Data Flow

- Figure our exactly what needs to be communicated
- Define **input parameters** for your view controller
- For communicating back up the hierarchy, **use loose coupling**
  - **Define a generic interface for observers (like delegation)**



Segues

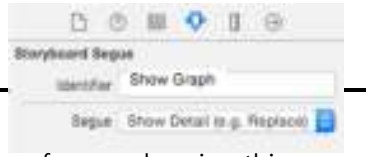
Segues

- Segue makes one MVC can cause another to appear
- Kinds of segues
  - **Show Segue** (will push in a Navigation Controller, else Modal)
  - **Show Detail Segue** (will show in Detail of a Split View or will push in a Navigation Controller)
  - **Modal Segue** (take over the entire screen while the MVC is up)
  - **Popover Segue** (make the MVC appear in a little popover window)
- Segues always create a new instance of an MVC
  - This is important to understand
  - The Detail of a Split View will get replaced with a new instance of that MVC
  - When you segue in a Navigation Controller it will not segue to some old instance, it will be new

Segues

- How do we make these segues happen?
  - **Ctrl-drag** in a storyboard from an instigator (like a button) to the MVC to segue to, then select the **kinds of segue** you want (Usually Show or Show Detail). Now click on the segue and open the Attribute Inspector, and give **the segue a unique identifier** here.
  - Can be done in code as well

Segues



- What's that segue identifier?
  - You would need it to invoke this segue from code using this UIViewController method

  func performSegueWithIdentifier(identifier: String, sender: AnyObject?)
  - The sender can be whatever you want (you'll see where it shows up in a moment)
  - You can ctrl-drag from the Controller itself to another Controller if you're segueing via code (because in that case, you'll be specifying the sender above)
- More important use of the identifier: **preparing for a segue**
  - When a segue happens, the View Controller containing the instigator gets a chance to prepare the destination View Controller to be segued to

Segues

- The method that is called in the instigator's Controller

```
func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if let identifier = segue.identifier {
        switch identifier {
            case "ShowGraph":
                if  let vc = segue.destinationViewController as? MyController{
                    vc.property1 = ...
                    vc.callMethodToSetUp(...)
                }
            default: break
        }
    }
}
```

The segue passed in contains important information about this segue:
1. The identifier from the storyboard
2. The controller of the MVC you are segueing to (which was just created for you)

The sender is either the instigating object from a storyboard (e.g. UIButton) or the sender you provided if you invoked the segue manually in code

Segues

- You can prevent a segue from happening too
  - Just implement this in your UIViewController

func shouldPerformSegueWithIdentifier(identifier: String?, sender: AnyObject?) -> Bool

  - The **identifier** is the one in the storyboard
  - The **sender** is the instigating object (e.g. the button that is causing the segue)

View Controllers

Problem: Managing a Screenful

- Controller manages views, data and application logic
- Applications are made of many of these controllers
- Would be nice to have a well-define starting point
  - UIView for views
  - Common language for talking about controllers

Problem: Building Typical Applications

- Some application flows are very common
  - Navigation-based
  - Tab bar-based
  - Combine the two
- Don't reinvent the wheel
- Plug individual screens together to build an application

UIViewController

- Basic building block
- Manages a screenful of content
- Subclass to add your application logic
  - Create **"your" own UIViewController subclass** for each screenful
  - Plug them together using existing **composite view controllers**

Your View Controller Subclass

```
import UIKit
class MyViewController: UIViewController {
    // a view controller will usually manage views and data
    var  myArray : [MyData] = [MyData] ()
    @IBOutlet weak var myLabel: UILabel!
    @IBOutlet weak var myTableView: UITableView!
}
```

The "View" in "View Controller"

- UIViewController superclass has a view property
  - **var view : UIView**
- Loads lazily
  - On demand when requested
  - Can be purged on demand as well (low memory)
- Sizing and positioning the view?
  - Depends on where it's being used
  - Don't make assumptions, be flexible

Accessing the sub-MVCs

- You can get the sub-MVCs via the viewController's property

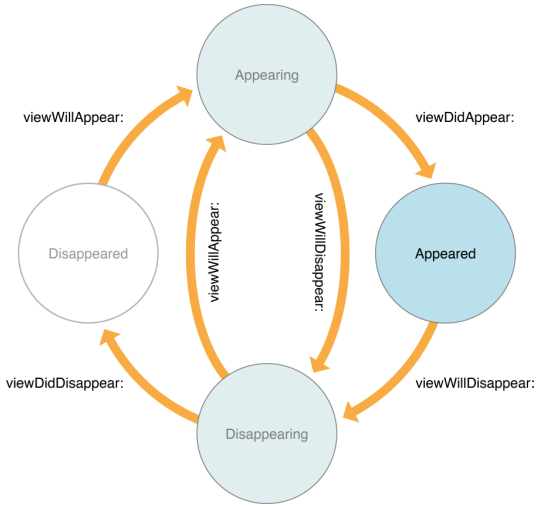
```
var viewControllers: [UIViewController] { get set } // optional
// for a tab bar, they are in order, left to right, in the array
// for a split view, [0] is the master and [1] is the detail
// for a navigation controller, [0] is the root and the rest are in order on the stack
```

- But how do you get ahold of the Split View Controller, Tab Bar Controller or Navigation Controller itself
  - Every UIViewController knows the Splie View, Tab Bar or Navigation Controller it is currently in
  - These are UIViewController properties

```
var tabBarController: UITabBarController? { get }
var splitViewController: UISplitViewController? { get }
var navigationController: UINavigationController? { get }
```

View Controller Lifecycle

View Controller Lifecycle



View Controller Lifecycle

- A sequence of messages is sent to a View Controller as it progresses through its "lifetime"
- Why does this matter?
  - You very commonly override these methods to do certain work
- The Start of the lifecycle
  - Creation
  - MVCs are most often instantiated out of a storyboard
  - There are ways to do it in code as well (rare)
- What then?
  - Preparation if being segued to
  - Outlet setting
  - Appearing and disappearing
  - Geometry changes
  - Low-memory situations

View Controller Lifecycle

- After instantiation and outlet-setting, viewDidLoad() is called
  - This is an exceptionally good place to put a lot of **setup code**
  - It's better than an init() because your outlets are all set up by the time this is called.
- One thing you may well want to do here is update your UI from your Model
  - Because now you know all of your outlets are set
- But be careful because the geometry of your view (its bounds) is not set yet!
  - At this point, you can't be sure you're on the iPhone 2-sized screen or an iPad or..
  - So **do not initialize things that are geometry-dependent here**.

viewDidLoad

- **viewDidLoad()** – called when the view controller's content view (the top of its view hierarchy) is created and loaded from a storyboard. This method is intended for **initial set up**.

```
override func viewDidLoad() {
    super.viewDidLoad()
    // your view has been loaded, customize it here if needed
    myLabel.text = "Test";
}
```

viewWillAppear: & viewWillDisappear:

- viewWillAppear() – intended for any operations that you want always to occur before the view becomes visible.
- viewDidAppear() – intended for any operations that you want to occur as soon as the view becomes visible, such as getching data or showing an animation.

```
override func viewWillAppear(animated: Bool) {
    super.viewWillAppear(animated)
    // your view is about to show on the screen
    self.beginLoadingDataFromTheWeb()
    self.startShowingLoadingProgress()
}
```

viewWillAppear: & viewWillDisappear:

- viewWillDisappear() – intended for any operations that you want to occur before the view disappear off screen.
- viewDidDisappear() – intended for any operations that you want to occur as soon as the view disappeared.

```
override func viewWillDisappear(animated: Bool) {
    super.viewWillDisappear(animated)
    // your view is about to leave the screen
    // do some clean up now that we've been removed from the
   screen
    // but be carefule not to do anything time-consuming here
    // maybe even kick off a thread to do stuff here
    self.rememberScrollPosition()
    self.saveDataToDisk()
}
```

Geometry changed?

- Geometry changed in View Controller?
  - Most of the time this will be automatically handled with Autolayout.
  - But you can get involved in geometry changes directly with these methods

  func viewWillLayoutSubviews()

  func viewDidLayoutSubviews()
  - They are called any time a view's frame changed and its subviews were thus re-layed out, e.g. autorotation.
  - You can reset the frames of your subviews here or set other geometry-related properties.
  - Between "will" and "did" autolayout will happen
  - These methods might be called more often than you'd imagine
  - So don't do anything in here that can't properly be done repeatedly

Geometry changed?

- **Autorotation**
  - Usually, the UI changes shape when the use rotates the device between portrait/landscape
  - You can control which orientations your app supports in the Settings of your project
  - Almost always, your UI just responds naturally to rotation with autolayout.
  - But if you want to participate in the rotation animation, you can use this method

func viewWillTransitionToSize(size: CGSize, withTransitionCoordinator: UIViewControllerTransitionCoordinator)

// The coordinator provides a method to animate alongside the rotation animation

didReceivedMemoryWarning

- **didReceovedMemoryWarning**
  - This method gets called when the device is in low-memory situations
  - This is rarely happens, but well-designed code with big-ticket memory uses might anticipate it.
  - Anything "big" that is not currently in use and can be recreated relatively easily should probably be released (by setting any pointers to it to nil)

awakeFromNib

- **awakeFromNib**
  - This method is sent to all objects that come out of a storyboard (including your Controller)
  - Happens before outlets are set (i.e., before the MVC is loaded)
  - Put code somewhere else if at all possible (e.g. viewdidLoad or viewWillAppear)

View Controller Lifecycle

- **Summary**
  - **Instantiated** (from storyboard usually)
  - **awakeFromNib**
  - **Segue** preparation happens
  - **Outlets** get set
  - **viewDidLoad**
  - These pairs will called each time your Controller view goes on/off screen

viewWillAppear and **viewDidAppear**

viewWillDisappear and **viewDidDisappear**

  - These geometry changed methods might be called at any time after viewDidLoad

viewWillLayoutSubviews (... then **autolayout** happends then...)
viewDidLayoutSubviews

  - If memory gets low, you might get **didReceiveMemoryWarning**

Controller of Controllers

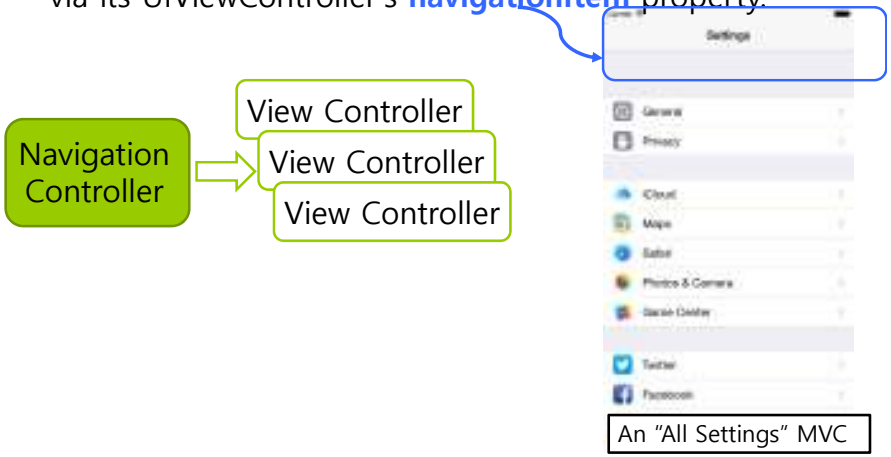
Controller of Controllers

- Special View Controllers that manage a collection of other MVCs
- **UINavigationController**
  - Manages a hierarchical flow of MVCs and presents them like a "stack of cards"
  - Commonly used on the iPhone
- **UITabBarController**
  - Manages a group of independent MVCs selected using tabs on the bottom of the screen
- **UISplitViewControllers**
  - Side-by-side, master-detail arrangement of two MVCs

Navigation Controller

UINavigationController

- **Pushes** and **pops** MVCs off of a stack (of view controllers)
- Each MVC communicates these contents via its UIViewController's **navigationItem** property.



An "All Settings" MVC

UINavigationController

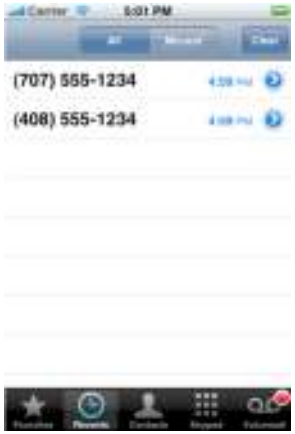
- Top view controller's **title**
  - This top area is drawn by the UINavigationController
- Previous view controller's **title**
  - This "back" button has appeared automatically.
- Top view controller's **view**
  - UIView
- Top view controller's **toolbar items** (iPhone OS 3.0)
  - It is possible to add MVC specific buttons via UIViewController's **toolbarItems** property (Array of UIBarButtonItems)

A "General" MVC

Customizing Navigation

Customizing Navigation

- Buttons or custom controls
- Interact with the entire screen



UINavigationItem

- Describes appearance of the navigation bar
  - Title string or custom title view
  - Left & right bar buttons
  - More properties defined in UINavigationBar.h
- **Every view controller has a navigation item** for customizing
  - Displayed when view controller is **on top of the stack**

Navigation Item Ownership

Displaying a Title

- UIViewController already has a **title** property
- Navigation item inherits automatically
  - Previous view controller's title is displayed in back button


```
viewController.title = "Detail"
```

Left & Right Buttons

- **UIBarButtonItem**
  - Special object, defines appearance & behavior for items in navigation bars and toolbars
- Display a string, image or predefined system item
- **Target + action** (like a regular button)

Text/System Bar Button Item


```
override func viewDidLoad() {
    let settingImage = UIImage(named: "fooButton")
    self.navigationItem.leftBarButtonItem = UIBarButtonItem(
            image: settingImage, style: UIBarButtonItemStyle.Plain,
            target: self, action: "foo")
    let addImage = UIImage(named: "addButton")
    self.navigationItem.rightBarButtonItem = UIBarButtonItem(
             image: addImage, style: UIBardButtonStyled.Bordered,
             target: self, action: "add:")
}
```

Edit/Done Button

- Very common pattern
- Every view controller has one available
  - Target/action already set up

```
self.navigationItem.leftBarButtonItem = self.editButtonItem;

//called when the user toggles the edit/done button
func setEditing(editing: Bool, animated: Bool)
{
    // update appearance of views
}
```

Custom Title View

- Arbitrary view in place of the title

```
var segmentedControl : UISegmentedControl = ……
self.navigationItem.titleView = segmentedControl
```

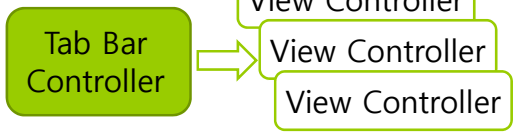
Back Button

```
self.title = "Hello there"
let heyButton = UIBarButtonItem("Hey!" …..)
self.navigationItem.backButtonItem = heyButton
```

Tab Bar Controllers

UITabBarController

- Array of view controllers



UITabBarController

- Selected view controller's view
- All view controller's titles

Tab Bar Appearance

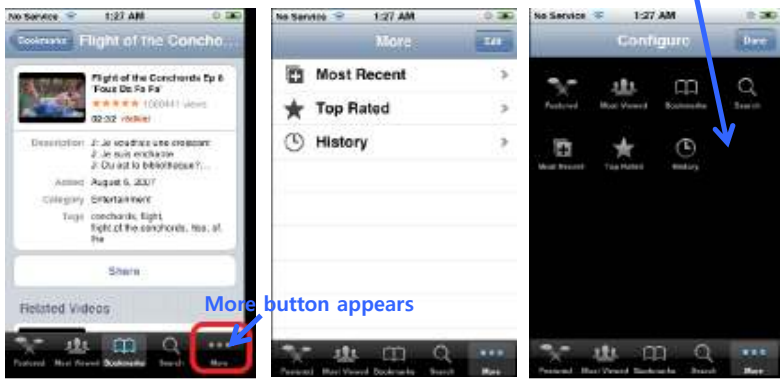
- View controllers can define their appearance in the tab bar
- Each view controller comes with a tab bar item for customizing
- **UITabBarItem**
  - **Title + image** or **system item**

Tab Bar Item


```
override func viewDidLoad() {
    let tabItems = self.tabBar.items as [UITabBarItem]
    let tabItem1 = tabItems[0] as UITabBarItem
    let tabItem2 = tabItems[1] as UITabBarItem
    tabItem1.title= "Playlists"
    tabItem2.title = "Bookmarks"
    tabItem1.image = UIImage(named: "music")
    tabItem2.image = UIImage(named: "bookmark")
}
```

More View Controllers

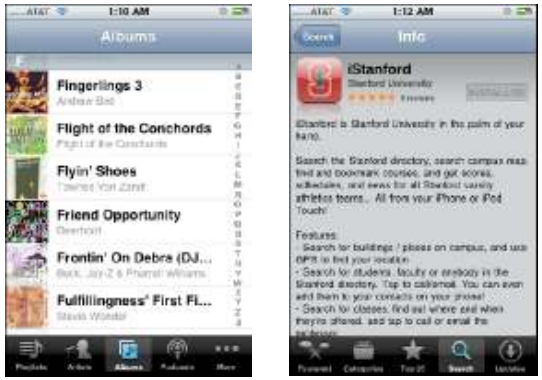
- What happens when a tab bar controller has too many view controllers to display at once?
  - **More tab bar item displayed automatically**
  - Use can navigate to remaining view controllers
  - Customize order



Combining Approaches

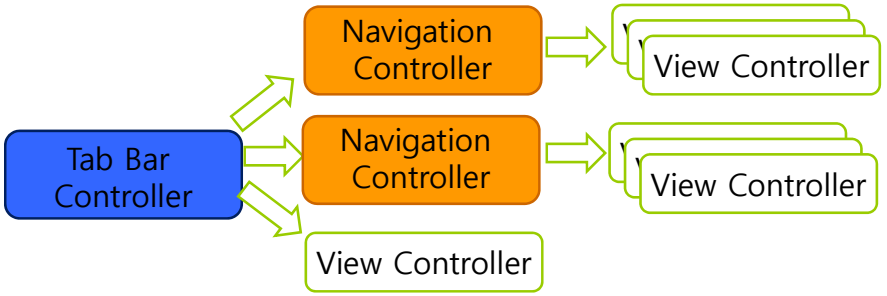
Tab Bar + Navigation Controllers

- Combine UINavigationController & UITabBarController?
  - **Quite common**
  - Multiple parallel hierarchies



Tab Bar + Navigation Controllers

- UINavigationController goes "inside" the UITabBarController
  - Never the other way around



Nesting Navigation Controllers

- Create a tab bar controller
  ```
  tabBarController = UITabBarController()
  ```
- Create each navigation controller
  ```
  navController = UINavigationController()
  navigationController?.pushViewController(firstViewController
                  animated:NO)
  ```
- Add them to the tab bar controller
  ```
  tabBarController.viewControllers = [ MyViewController1(),
  MyViewController2() ]
  ```

Setting Up TabBar+Navigation Controller

```
func application(application: UIApplication,
      didFinishLaunchingWithOptions launchOptions: [NSObject:
      AnyObject]?) -> Bool {
  let tarBarController = UITabBarController()
  let vc1 = MyVC(nibName: "MyVC", bundle: nil)
  let vc2 = OtherVC(nibName: "OtherVC", bundle: nil)
  let controllers = [ vc1, vc2 ] // set the array of view controllers
  tarBarController.viewControllers = controllers
  // add the tab bar controller's view to the window
  windows?.rootViewController = tabBarController
  vc1.tabBarItem = UITabBarItem(title: "MyVC", image:
                   UIImage(named: "my vc icon"), tag: 1)
  vc2.tabBarItem = UITabBarItem(title: "OtherVC", image:
                   UIImage(named: "other vc icon"), tag: 2)
  return true
}
```

References

- Lecture 7(MultipleMVC Controller) & 8(ViewControllerLifeCycle) Slide from Developing iOS8 Apps with Swift (Winter 2015) @Stanford University
- http://www.codingexplorer.com/segue-swift-view-controllers/
- https://developer.apple.com/library/prerelease/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/Lesson4.html#//apple_ref/doc/uid/TP40015214-CH6-SW1