Announcements

- Lab 2 is due tonight by 11:59 PM
  - Late policy is 10% of lab total per day late
    - So -7.5 points per day late for lab 2

- Lab 3 is due on Wednesday Oct 4th

Today’s Topics

- Designing iPhone Applications
- Model-View-Controller (Why and How?)
- View Controllers
Designing iPhone Applications

Different Flavors of Mail
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

A Screenful of Content

- Slice of your application
- Views, data, logic
Parts of a Screenful

Model

Controller

View

Parts of a Screenful

Model

Controller

View
Model-View-Controller
(Why and How?)

Why Model-View-Controller?

• Ever used the word “spaghetti” to describe code?
• Clear responsibilities make things easier to maintain
• Avoid having one monster class that does everything
Why Model-View-Controller?

- 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 fewer lines of code

Communication and MVC

Model

KVO, notifications

View

KVO, notifications

Controller

KVO, notifications

target-action, delegation
View Controllers

Problem: Managing a Screenful

- Controller manages views, data and application logic
- Apps are made up of many of these
- Would be nice to have a well-defined starting point
  - A la UIView for views
  - Common language for talking about controllers
Problem: Building Typical Apps

- 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 app

UIViewController

- Basic building block
- Manages a screenful of content
- Subclass to add your application logic
“Your” and Apple View Controllers

• Create your own UIViewController subclass for each screenful

• Plug them together using existing composite view controllers

“Your” and “Our” View Controllers

• Create your own UIViewController subclass for each screenful

• Plug them together using existing composite view controllers
Your View Controller Subclass

```swift
import UIKit

class MyViewController : UIViewController {
    // A view controller will usually
    // manage views and data
    var myData = [String]()

    // And respond to actions
    @IBAction func doSomeAction(_ sender: UIButton) {
        // Do something here when button pressed
    }
}
```

The “View” in “View Controller”

• UIViewController superclass has a view property
  – view: UIView

• Loads lazily
  – On demand when requested
    • OS figures this out
  – 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
When to call -loadView?

• Don’t do it!

• Cocoa tends to embrace a lazy philosophy
  – Call setNeedsDisplay() instead of draw(_:)

• Allows work to be deferred
  – Performance!
    • Consider time to launching an application

View Controller Lifecycle

```swift
override func viewDidLoad() {
    // Your view has been loaded
    // Customize it here if needed

    view.someWeirdProperty = true
}
```
override func viewWillAppear(_ animated: Bool) {
    super.viewDidLoad(animated)
    // Your view is about to show on the screen
    beginLoadingDataFromTheWeb()
    startShowingLoadingProgress()
}

override func viewWillAppear(_ animated: Bool) {
    super.viewDidLoad(animated)
    // Your view is about to show on the screen
    beginLoadingDataFromTheWeb()
    startShowingLoadingProgress()
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    // Your view is about to leave the screen
    rememberScrollPosition()
    saveDataToDisk()
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewDidLoad(animated)
    // Your view is about to leave the screen
    rememberScrollPosition()
    saveDataToDisk()
}
Navigation View Controller Demo

Loading and Saving Data

• Lots of options out there, depends on what you need
  – UserDefaults
  – Property lists
  – SQLite
  – Web services
• Covering in greater depth in a few lectures
Saving State Across App Launches

- UserDefaults (renamed from NSUserDefaults) to read and write prefs & state
- Singleton object: UserDefaults.standard
- Methods for storing & fetching common types:
  - integer(forKey: String)
  - set(value: Int, forKey: String)
- Find an appropriate time to store and restore your state

More View Controller Hooks

- Automatically rotating your user interface
- Low memory warnings
UserDefaults Demo