House Course 59-20

Web and Mobile Applications
Week 3: UIKit Explored

Attendance: http://goo.gl/forms/OOqXPqega0

Davis Gossage
Jesse Hu

Week 3

- ToDo App Review
- Exploring UIKit
- Building the Yik-Yak front-end

Looking Ahead

- 2/17 - Backend services, database techniques
- 2/24 - Building the data model, connecting to the backend
- 3/02 - Additional iOS Topics (location services, local datastore)
- 3/09 - Catch-up / JS intro

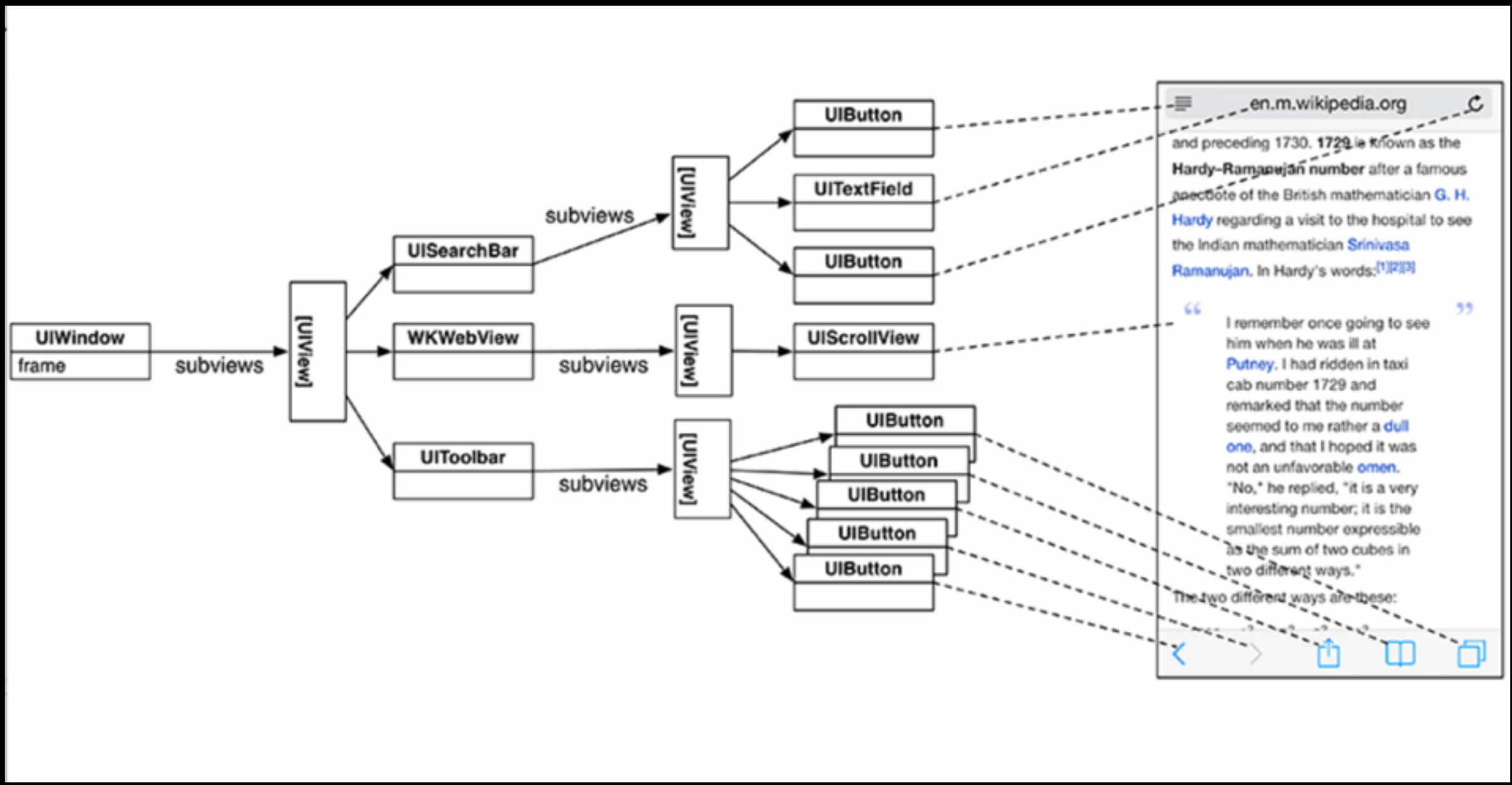
Attendance

Attendance: http://goo.gl/forms/OOqXPqega0

Views

- A view (UIView subclass) represents a rectangular area for drawing and handling touch events
- Scenes are built by creating a hierarchy of UIView objects
    - Rectangles stacked on rectangles…
- UIView is the parent class, its children include:
    - **UILabel** - single line of text
    - **UITextView** - multiple lines of text
    - **UITextField** - accepts user inputted text
    - **UITableView** - displays a list of graphical cells in a single column

View Hierarchy



View Controllers

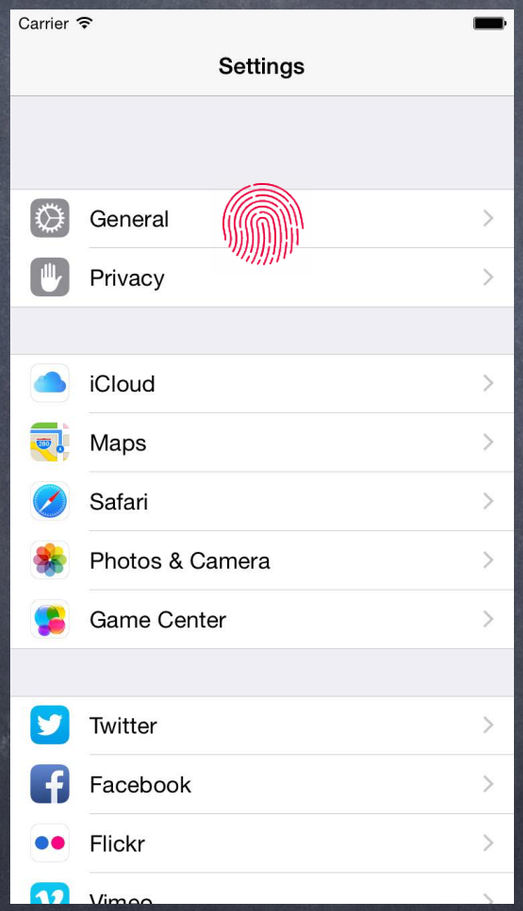
- Manipulates views
- View Controllers in the storyboard are backed by a .swift file
    - View Controllers make connections with the views using outlets (IBOutlet) and actions (IBAction)
- UIViewController is the parent class, it's children include:
    - UINavigationController - manages a navigation bar, allows for 'pushing' to new scenes
    - UITabBarController - lets the user tab between scenes via a bottom bar

.storyboard

- Storyboards are composed of Scenes. A scene contains a hierarchy of **views**, often with a single **view controller**.
    - Scenes can be connected to other Scenes with **segues**.
- Going to the **Identity Inspector** of the Scene shows the **view controller** the scene belongs to

UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …



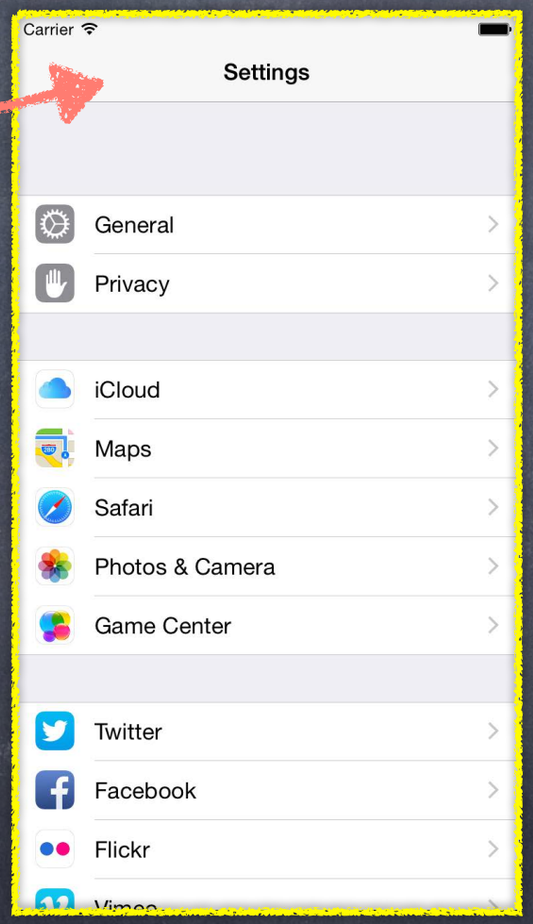
UINavigationController

- Pushes and pops MVCs off of a stack (like a stack of cards) …

This top area is drawn by the `UINavigationController`

But the contents of the top area (like the title or any buttons on the right) are determined by the MVC currently showing (in this case, the "All Settings" MVC)

Each MVC communicates these contents via its `UIViewController`'s `navigationItem` property



An "All Settings" MVC

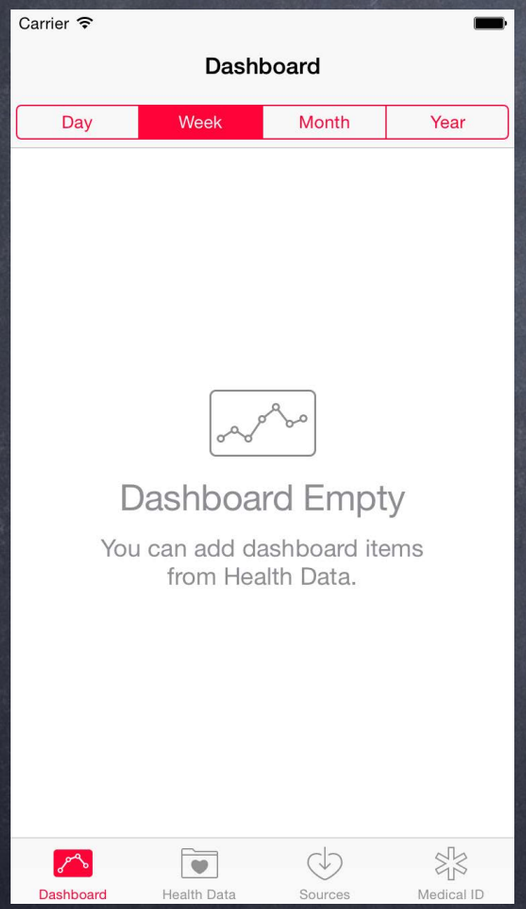
Navigation Controller Example

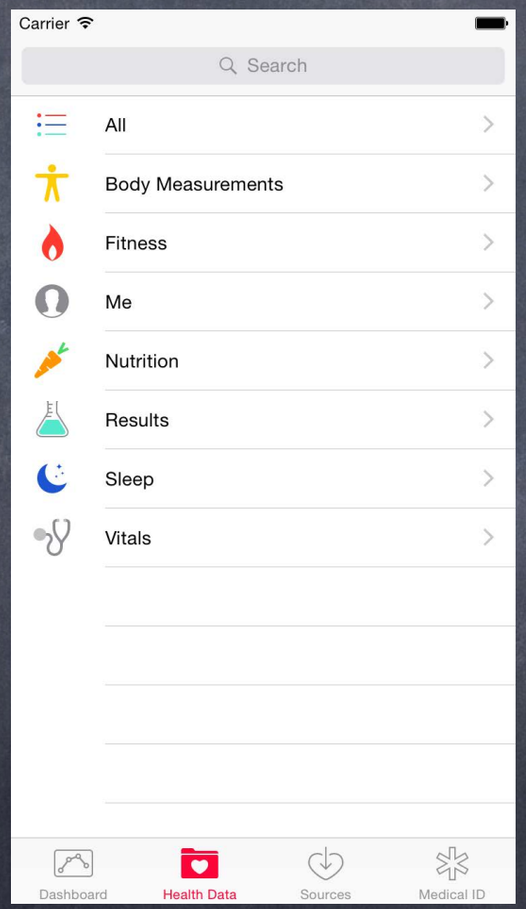
ToDoList

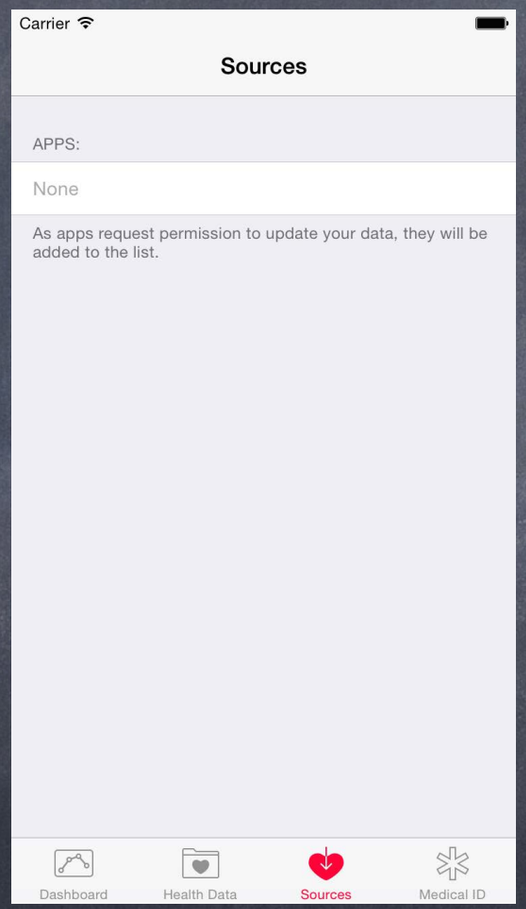
- Let’s explore the .storyboard of the ToDoList app
- How many Scenes?
- What View Controller owns each scene?
- What views make up each scene?

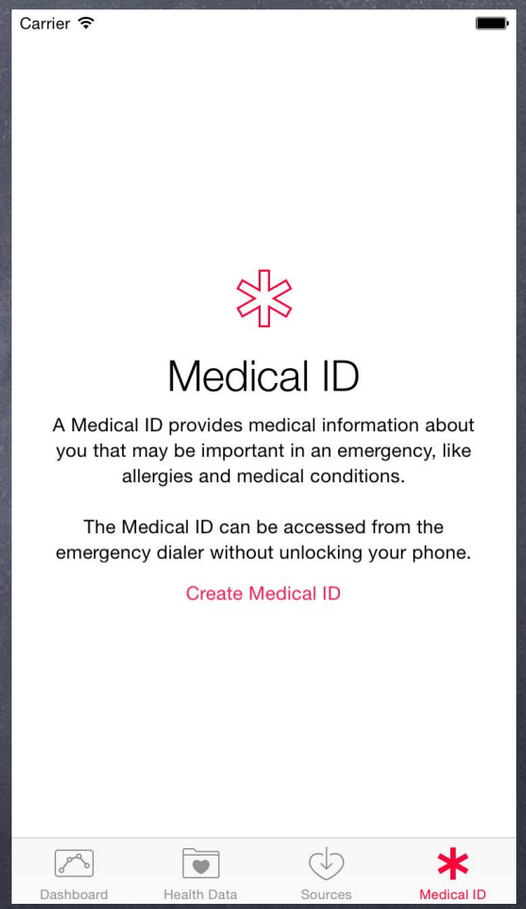
UITabBarController

- It lets the user choose between different MVCs …







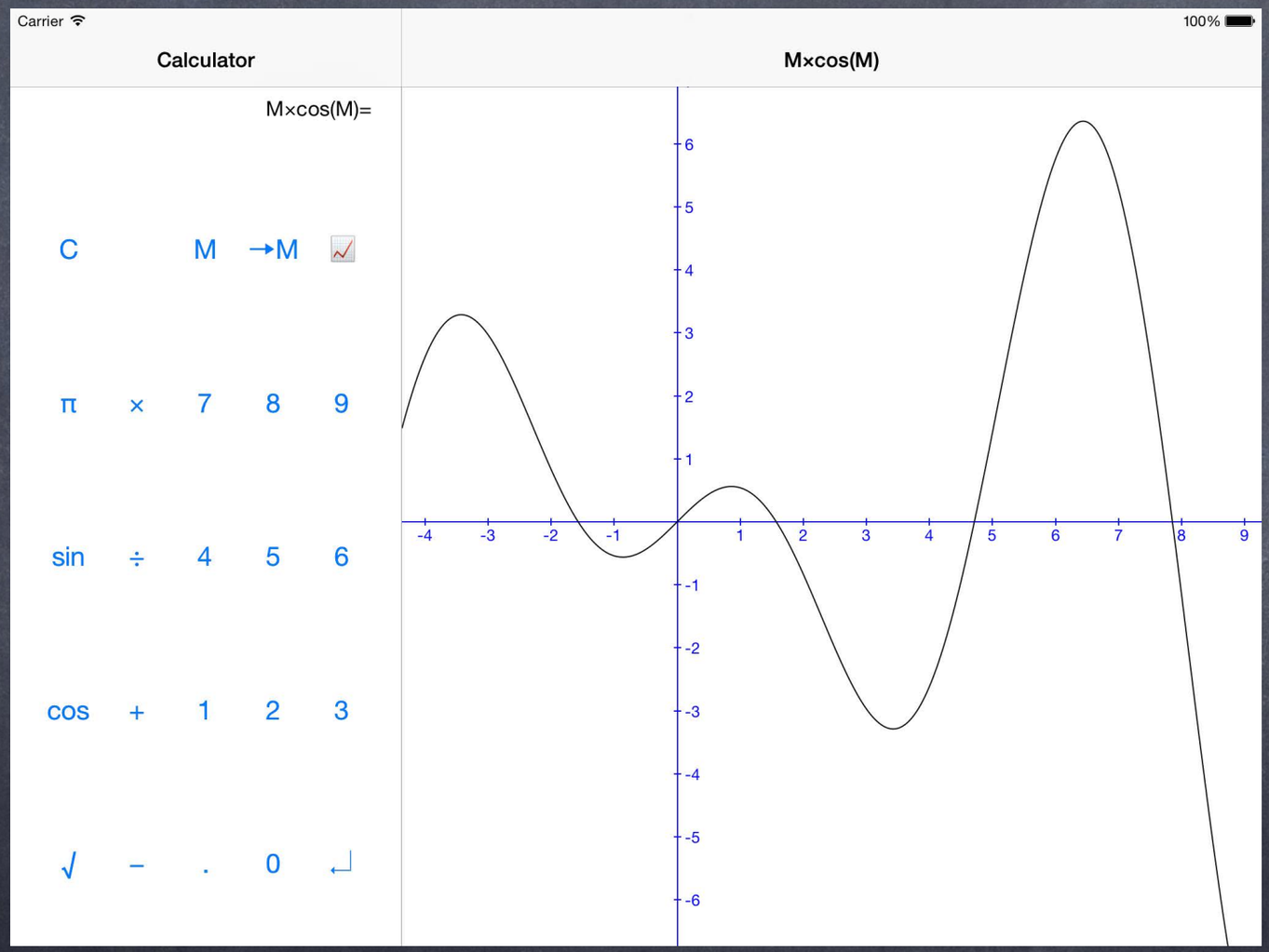


CS193p
Winter 2015

Tab Bar Example

UISplitViewController

- Puts two MVCs side-by-side …



Master

Detail

CS193p
Winter 2015

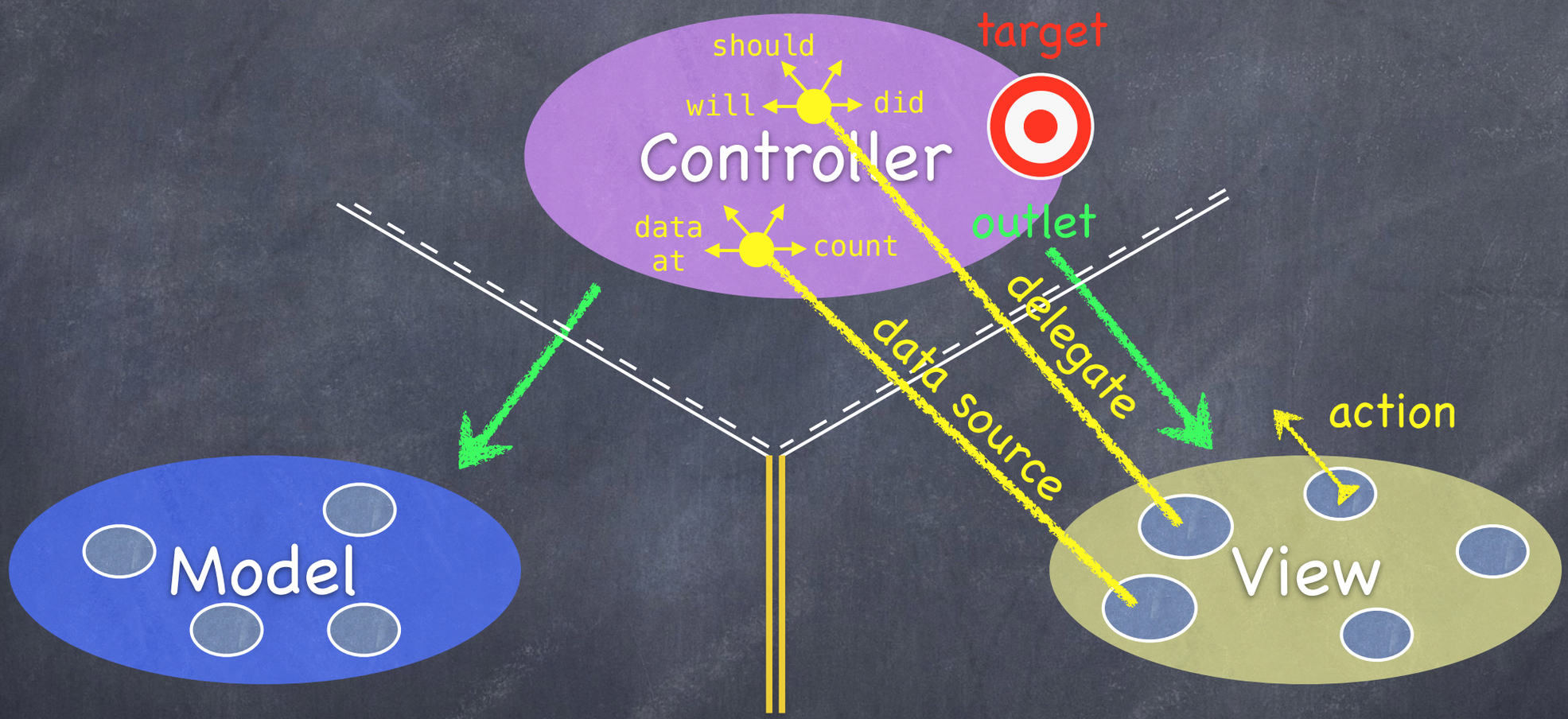
Segue

- Segues cause new scenes to appear
- Kinds of segues
  - Show segue (pushes a new scene in a navigation controller)
  - Show detail segue (shows detail of a split view)
  - Modal segue (takes over the entire screen with the new scene)
  - Popover segue (makes the new scene appear in a little popover window)

ToDoList

- Where is the segue?
- We wired this segue directly to a button, no code required
- Alternatively, we could have created a segue manually triggered by its identifier
  - self.performSegueWithIdentifier(id: String)

MVC



Controllers are almost always that data source (not Model!).

Communication from View <-> View Controller

- Communication from V to VC, or VC to VC happens 4 different ways:
  - Outlet: allows the VC to change the view
    - Ex: make a label show some text
  - Action: allows the view to notify the VC of some user action
    - Ex: user taps a button
  - Delegate : allows a view or VC to blindly publish an event. This is received by any VC that has claimed responsibility for that event
    - Ex: user taps on a row in a tableview
  - Data source : allows a view to blindly request information. This is fulfilled by any VC that claims responsibility for this information
    - Ex: tableview requests how many rows it needs to display

ToDoList

- Example of Outlet?
- Example of Action?

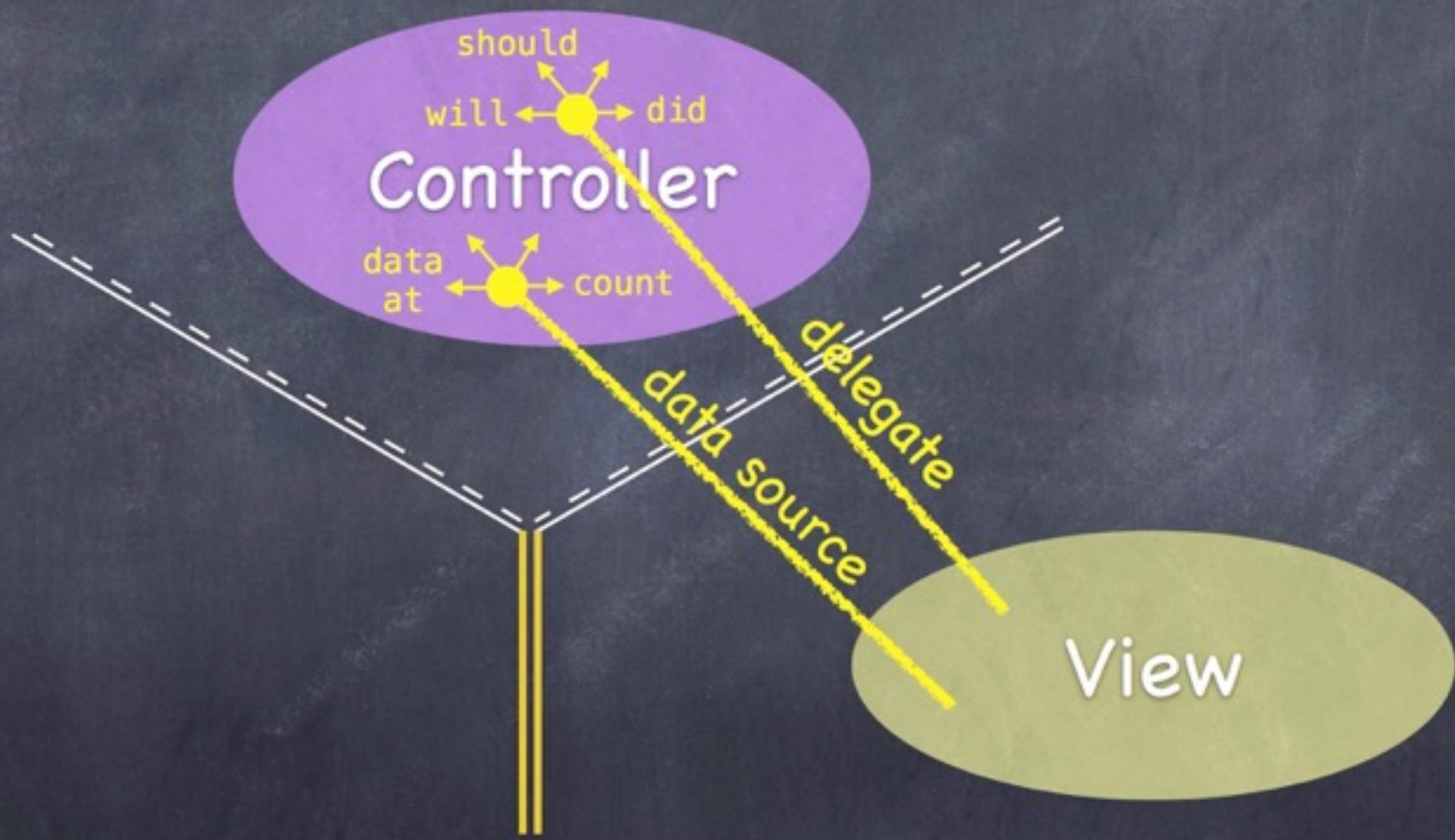
Protocols

- A way to express an API minimally
  - Instead of forcing the caller to pass a `class`/`struct`, we can ask for specifically what we want
  - We just specify the properties and methods needed
- A protocol is a TYPE just like any other type, except ...
  - It has no storage or implementation associated with it
  - Any storage or implementation required to implement the protocol is in an implementing type
  - An implementing type can be any `class`, `struct` or `enum`
  - Otherwise, a `protocol` can be used as a type to declare variables, as a function parameter, etc.
- There are three aspects to a protocol
  1. the `protocol` declaration (what properties and methods are in the `protocol`)
  2. the declaration where a `class`, `struct` or `enum` says that it implements a `protocol`
  3. the actual implementation of the `protocol` in said `class`, `struct` or `enum`

Delegation

- A very important use of `protocols`

  It's how we can implement "blind communication" between a View and its Controller



Delegation

- A very important use of `protocols`

  It's how we can implement "blind communication" between a View and its Controller

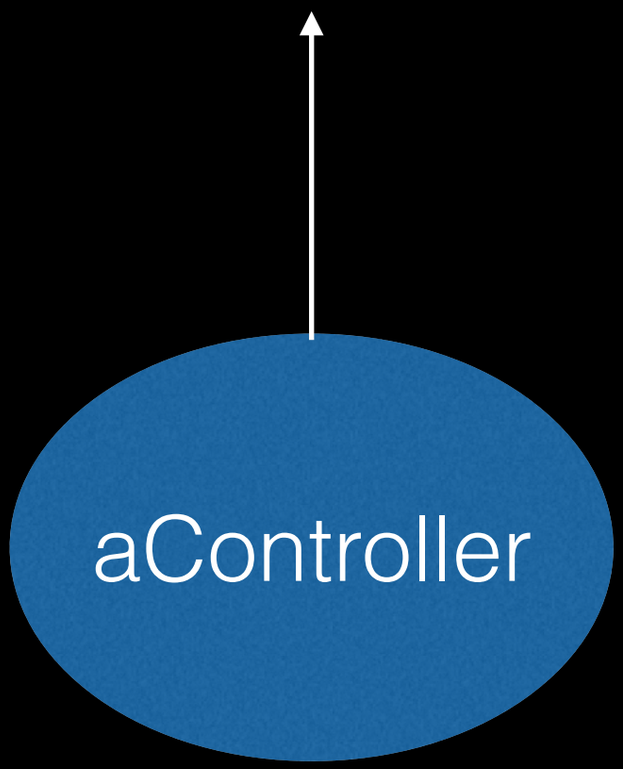
- How it plays out ...

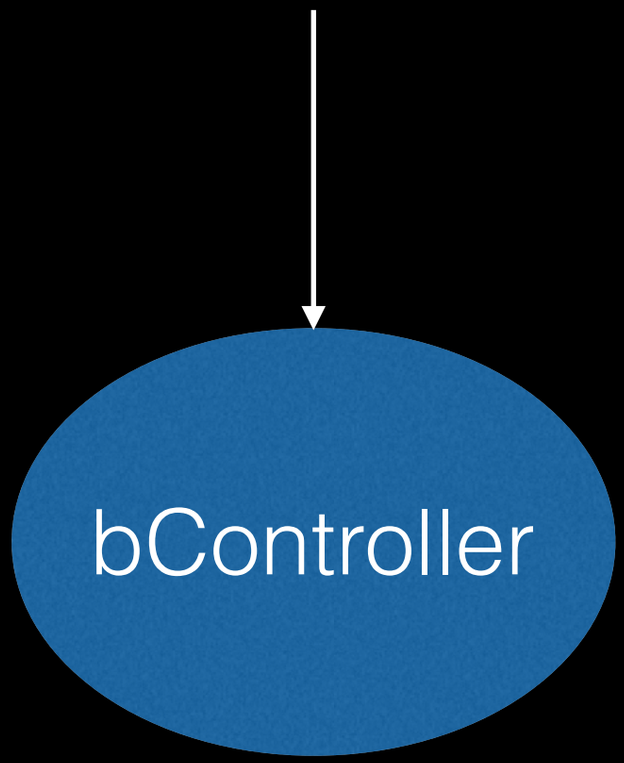
  1. Create a delegation `protocol` (defines what the View wants the Controller to take care of)
  2. Create a `delegate` property in the View whose type is that delegation `protocol`
  3. Use the `delegate` property in the View to get/do things it can't own or control
  4. Controller declares that it implements the `protocol`
  5. Controller sets `self` as the `delegate` of the View by setting the property in #2 above
  6. Implement the `protocol` in the Controller

- Now the View is hooked up to the Controller

  But the View still has no idea what the Controller is, so the View remains generic/reusable

Delegates






```
protocol ComposeDelegate {
    func userSavedItem(item: ToDoItem)
}

class ComposeViewController: UIViewController {

    var delegate: ComposeDelegate?
```

```
    delegate?.userSavedItem(newItem)
```

```
class TableViewController: UITableViewController, ComposeDelegate {

    // MARK: - Navigation

    // In a storyboard-based application, you will often want to do a little preparation before
        navigation
    override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
        let destinationVC = segue.destinationViewController as! ComposeViewController
        destinationVC.delegate = self
    }

    // MARK: compose delegate

    func userSavedItem(item: ToDoItem) {
        print("user wants to save an item!")
    }

}
```

ToDoList

- Look at delegate implementation
- Look at UITableView's delegate and data source implementation

Views

- We add views to the hierarchy most often via the storyboard
- Can also, like almost everything in the storyboard, do this programmatically
    - addSubview(aView: UIView)
    - removeFromSuperview()
- What is at the top of the view hierarchy… where do things start?
    - Every VC includes a base var view: UIView property, goes from edge to edge of the scene
    - Any views that are added are subviews of the base view

View Coordinate System

- CGFloat
    - Used instead of a Double when dealing with coordinates. Initialize from Double with CGFloat(myDouble)
- CGPoint
    - Contains an x: CGFloat and a y: CGFloat
- CGSize
    - Contains a width: CGFloat and a height: CGFloat
- Every UIView has a property called frame: CGRect
    - CGRect has the following properties:
        - origin: CGPoint
        - size: CGSize

View Coordinate System

(0,0)

increasing x

(500, 35)

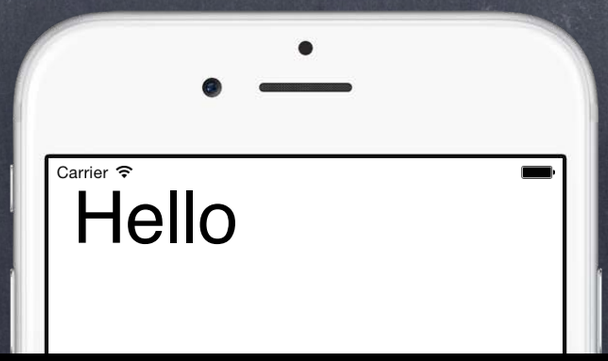
increasing y

- Origin is upper left
- Units are points, not pixels
  - Pixels are the minimum-sized unit of drawing your device is capable of
  - Points are the units in the coordinate system
  - Most of the time there are 2 pixels per point, but it could be only 1 or something else
  - How many pixels per point are there? `UIView`'s `var contentScaleFactor: CGFloat`
- The boundaries of where drawing happens
  - `var bounds: CGRect` // a view's internal drawing space's origin and size
  - This is the rectangle containing the drawing space in its own coordinate system
  - It is up to your view's implementation to interpret what `bounds.origin` means (often nothing)
- Where is the `UIView`?
  - `var center: CGPoint` // the center of a `UIView` in its superview's coordinate system
  - `var frame: CGRect` // the rect containing a `UIView` in its superview's coordinate system

Creating Views

- Most often your views are created via your storyboard
  - Xcode's Object Palette has a generic UIView you can drag out
  - After you do that, you must use the Identity Inspector to changes its class to your subclass
- On rare occasion, you will create a UIView via code
  - You can use the frame initializer … `let newView = UIView(frame: myViewFrame)`
  - Or you can just use `let newView = UIView()` (frame will be CGRectZero)
- Example

```
// assuming this code is in a UIViewController
let labelRect = CGRect(x: 20, y: 20, width: 100, height: 50)
let label = UILabel(frame: labelRect) // UILabel is a subclass of UIView
label.text = “Hello”
view.addSubview(label)
```



CS193p
Winter 2015

UIColor

- Colors are set using `UIColor`
  - There are type methods for standard colors, e.g. `let green = UIColor.greenColor()`
  - You can also create them from RGB, HSB or even a pattern (using `UIImage`)
- Background color of a `UIView`
  - `var backgroundColor: UIColor`
- Colors can have alpha (transparency)
  - `let transparentYellow = UIColor.yellowColor().colorWithAlphaComponent(0.5)`
  - Alpha is between 0.0 (fully transparent) and 1.0 (fully opaque)
- If you want to draw in your view with transparency …
  - You must let the system know by setting the `UIView` `var opaque = false`
- You can make your entire `UIView` transparent …
  - `var alpha: CGFloat`

CS193p
Winter 2015

Try it

- New single view project, add a black square 100 x 100 to the center of the screen

View Transparency

- What happens when views overlap and have transparency?
  - As mentioned before, `subviews` list order determines who is in front
  - Lower ones (earlier in the array) can "show through" transparent views on top of them
  - Transparency is not cheap, by the way, so use it wisely
- Completely hiding a view without removing it from hierarchy
  - `var hidden: Bool`
  - A `hidden` view will draw nothing on screen and get no events either
  - Not as uncommon as you might think to temporarily hide a view

Try it

- Change the black square to be blue, make it 50% transparent

UIView Layer

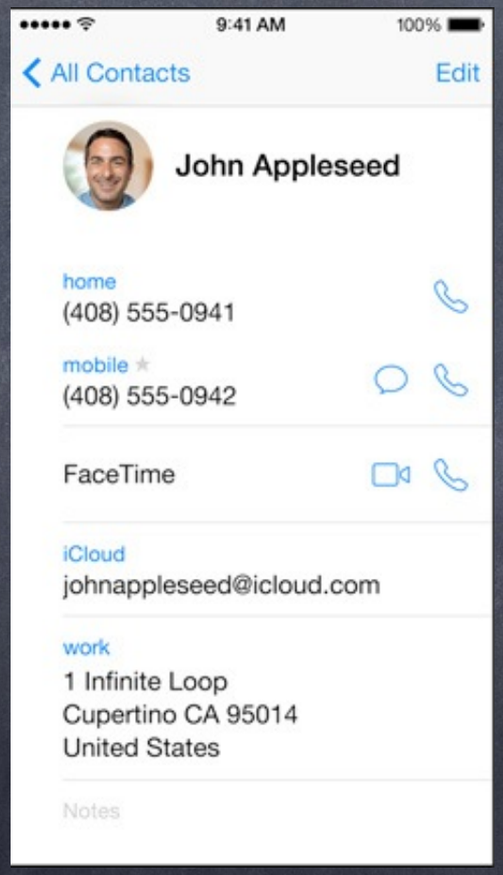
- Every UIView has a Core Animation layer (layer: CALayer)
- The layer has some interesting properties:
  - cornerRadius
  - borderWidth, borderColor
  - shadowRadius, shadowColor

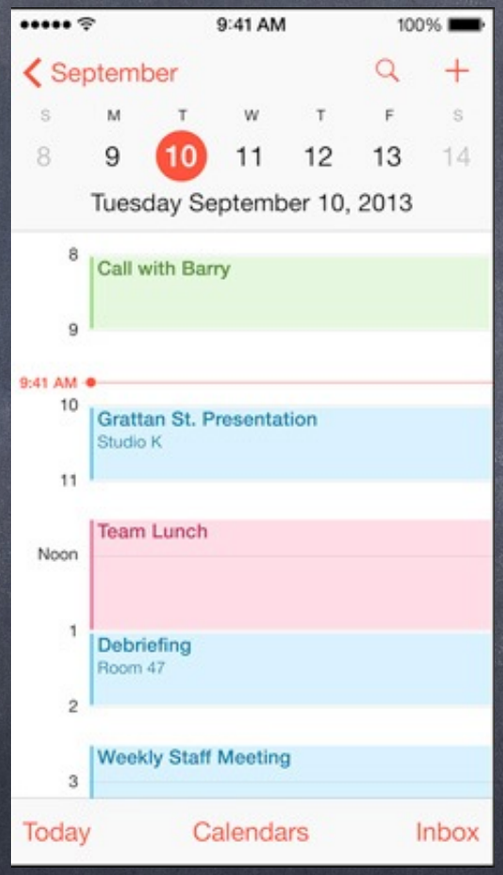
Try it

- Make a blue circle with a 5pt orange border

Fonts

- Fonts in iOS 7 and later are very important to get right
  - They are fundamental to the look and feel of the UI





Fonts

- The absolutely best way to get a font in code
  - Get preferred font for a given text style (e.g. body, etc.) using this `UIFont` type method …
  - `class func preferredFontForTextStyle(UIFontTextStyle) -> UIFont`
  - Some of the styles (see `UIFontDescriptor` documentation for more) …
  - `UIFontTextStyle.Headline`
  - `UIFontTextStyle.Body`
  - `UIFontTextStyle.Footnote`
- There are also "system fonts"
  - These appear usually on things like buttons
  - `class func systemFontOfSize(pointSize: CGFloat) -> UIFont`
  - `class func boldSystemFontOfSize(pointSize: CGFloat) -> UIFont`
  - Don't use these for your user's content. Use preferred fonts for that.
- Other ways to get fonts
  - Check out `UIFont` and `UIFontDescriptor` for more, but you should not need that very often

Drawing Images

- There is a `UILabel`-equivalent for images

  `UIImageView`

  But, again, you might want to draw the image inside your `drawRect` …

- Creating a `UIImage` object

  `let image: UIImage? = UIImage(named: “foo”)` // note that its optional

  You add `foo.jpg` to your project in the `Images.xcassets` file (we've ignored this so far)

  Images will have different resolutions for different devices (all managed in `Images.xcassets`)

- You can also create one from files in the file system

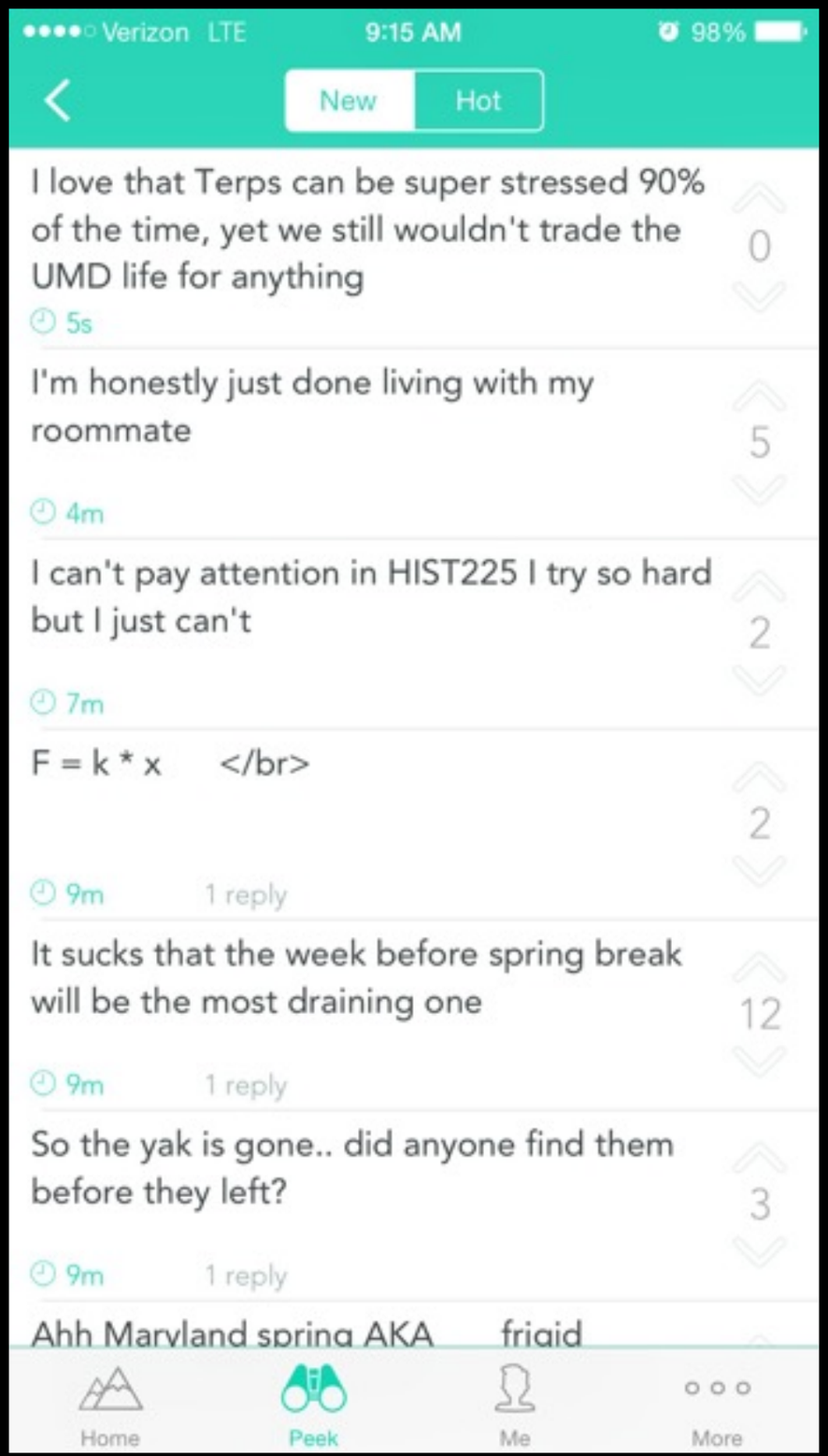
  (But we haven't talked about getting at files in the file system … anyway …)

  `let image: UIImage? = UIImage(contentsOfFile: aString)`

  `let image: UIImage? = UIImage(data: anNSData)` // raw jpg, png, tiff, etc. image data

- You can even create one by drawing with Core Graphics

  See documentation for `UIGraphicsBeginImageContext(CGSize)`


Verizon LTE
9:15 AM
98%
New
Hot
I love that Terps can be super stressed 90% of the time, yet we still wouldn't trade the UMD life for anything
0
5s
I'm honestly just done living with my roommate
5
4m
I can't pay attention in HIST225 I try so hard but I just can't
2
7m
F = k * x </br>
2
9m
1 reply
It sucks that the week before spring break will be the most draining one
12
9m
1 reply
So the yak is gone.. did anyone find them before they left?
3
9m
1 reply
Ahh Maryland spring AKA frigid
Home
Peek
Me
More

Fun with animations

- UIView.animateWithDuration
  - Animates any changes to views at 60 frames per second
  - Includes options like Repeat and Autoreverse

Reading

- Make sure you have **5th** Edition
- Last week: Chapters 1 & 3
- Ch 4: Text Input & Delegation
- Ch 5: View Controllers
- Ch 9: UITableView and UITableViewController
- Ch 11: Subclassing UITableViewCell (first 3 sections)