iOS App Components
iOS Architecture

Cocoa Touch

Media

Core Services

Core OS
Building Blocks

- **UIApplication**
  - The main entry point for your app
  - Each app has exactly one instance of this class
  - Provides the main interface back to the OS
  - Handles all incoming info from the OS (such as touch event, memory warnings, incoming phone call, etc.)
  - Passes these messages off to...
Building Blocks

- **UIApplicationDelegate**
  - Manages the running of your app
  - Handles “major” events, like app switching, app initialization, etc.
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    return true
}

Other Components

- **UIDocument**: allows for internal app documents and data stores
- **ViewController**: manages all views (scenes) for the app
- **UIWindow**: the one window of the iOS device (only have more if external display)
- **View Objects**: all the widgets in a scene
The Main App Loop

Operating system

Event source

Port

Event queue

Main run loop

Application object

Core objects
The Main App Loop

• The OS receives input and passes it to the UIApplication
• Which passes it to the UIApplicationDelegate
• Which passes it to the UIWindow
• Which passes it to the currently seen ViewController
The Main App Loop

- Other events are passed to the “First Responder” object that is available
- Everything that can accept and respond to an event is a responder object
- The first responder object is the top-level designated object in a view to handle events
App States

- Not running
  - Foreground
    - Inactive
    - Active
  - Background
    - Background
    - Suspended
AppStates

- `applicationWillEnterForeground` – starting to move to active state
- `applicationDidBecomeActive` – called right before view displayed
- `applicationWillResignActive` – first call before going into background
- `applicationDidEnterBackground` – now is in background
- `applicationWillTerminate` – will end
On Launch

• `application` comes in with `didFinishLaunchingWithOptions`

• Check dictionary `launchOptions` for info on why was launched (somewhat like looking at the `Intent`)

• Any app not responding in 5 seconds is killed

• Start initialization

• UIKit grabs first storyboard and `ViewController`
Inside the View

- **Appearing**
  - `viewWillAppear:`
  - `viewDidAppear:`

- **Appeared**
  - `viewWillDisappear:`
  - `viewDidDisappear:`

- **Disappearing**
  - `viewWillDisappear:`
  - `viewDidDisappear:`

- **Disappeared**
  - `viewWillAppear:`
  - `viewDidAppear:`
Using Segues

• Instead of Intents like Android, we’ll use Segues to pass data between Scenes / ViewControllers
MVC in iOS

• We again see Model-View-Controller as part of the foundation for a mobile system

• Because of the nature of Objective-C and NeXTSTEP, MVC is one of the primary design patterns for both iOS and OS X
MVC in iOS
MVC in iOS

Diagram showing the Model-View-Controller (MVC) architecture in iOS, with components such as Model, Controller, View, UIApplication, Application Delegate, Root View Controller, and UIWindow.
MVC in iOS

• Model: The base classes you write to hold data
  – Could subclass `NSObject/Object`
  – Could connect to a database or other data source
  – Could be a simple class you write
MVC in iOS

• View: Any rectangular drawable object on the screen
  – Various Stack, Table, and Collection Views
  – Image, Text, Picker Views
  – Map and WebKit View
  – Scene Kit View (for 3D scenes)
  – Manages drawing its area on the screen
  – Can contain other views (or be contained)
  – Responds to touch and other events
MVC in iOS

• **Controller: The ViewController class**
  – Each ViewController manages a hierarchy of Views
  – The `view` property of the class contains the root
  – Views are access lazily; that is, they are only loaded when needed
  – Updates the contents of the views, usually in response to changes to the underlying data
  – Responds to user interactions with views
  – Resizes views and manages the layout of the overall interface
Building up in MVC

• Start by considering your data
  – Where does it come from?
  – How will you store it?
  – What sort of access do you need from it?
Building up in MVC

• Storyboard your idea
  – Just as with Android, layout each screen
  – What views make up each screen?
    • Text?
    • A Table?
    • An Image?
  – What happens when you touch or swipe on each view? Or on the screen?
Building up in MVC

- Create the appropriate Controller type
- Add Views to the Controller to get the layout the way you want it
- Start with dummy data
- Run in the simulator often!
- Check and adjust your constraints to get everything on screen
- Test rotation!
Building up in MVC

- In your Storyboard, link the various Views back to the code using ctrl-click/drag
- Do the same for buttons and other controls
- Load data as needed to refresh the view (often happens automatically)
- Add in code to handle user events, like touches and swipes
- Add Navigation Controllers to allow for switching between scenes