

# IOS INTRODUCTION

# WHAT IS AN OPERATING SYSTEM?

- A set of programs that manage computer hardware resources and provide common services for application software
- Most important system software in computer system
- ex. Android, Mac OS X, Microsoft Windows

# WHAT IS IOS?

- The operating system that runs on iPad, iPhone, and iPod touch devices.
- The operating system manages the device hardware and provides the technologies required to implement native apps.
- The operating system also ships with various system apps, such as Phone, Mail, and Safari, that provide standard system services to the user.

# WEB VS. NATIVE VS. HYBRID



# WEB VS. NATIVE VS. HYBRID

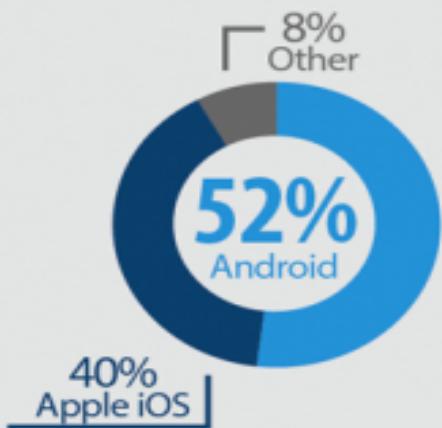
## Mobile App Development Timelines



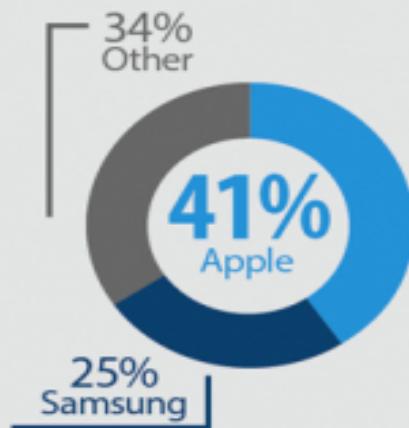
# WHY IOS ?

## MOBILE MARKET SHARE

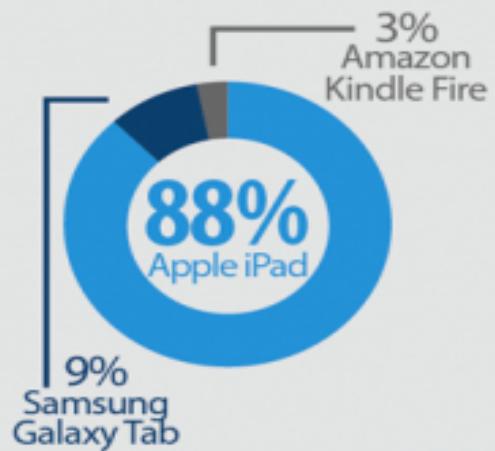
MOST POPULAR OS



MOST POPULAR SMARTPHONE



MOST POPULAR TABLET



# WHY IOS?

- 470 million iOS users
- 1 billion + Android users
- Last year, Apple paid \$10 billion to developers
- Last year, Google paid \$5 billion to developers

# iOS HISTORY

iOS 1 - iOS 8



June  
2007



# iOS 1

Realease date: June 29, 2007 Updates: iOS 1.1.1 | iOS 1.1.3

Samsung 32-bit RISC ARM 11 - 412 MHz



#### iTunes Music Store

Capability to download music directly to an iPhone.



#### Mobile Safari web browser

Ability to surf the net on a fully operational browser.



#### E-mail

Emails could be received and sent from an iPhone



#### Camera

2.0 Megapixel Camera



#### Multitouch Software Keyboard

The first genuinely finger usable keyboard



iPhone - Device Support and Backward Compatibility:

#### iPhone 2G

iPhone model that launched with specific version of iOS:

#### iPhone 2G



iPhone 2G

Noteworthy:

Was called the "iPhone OS"

iOS 1.0 had no support for native,  
3rd party apps

iOS 1.0 introduced Apple's  
"Springboard" homescreen

July  
2008



iPhone - Device Support and Backward Compatibility:

#### **iPhone 2G, 3G**

iPhone model that launched with specific version of iOS:

#### **iPhone 3G**

iPod Touch devices:

#### **iPod Touch 1st Gen**



# iOS 2

Realease date: July 11, 2008

Updates: iOS 2.0.1 | iOS 2.0.2 | iOS 2.1 | iOS 2.2



Samsung 32-bit RISC ARM 11 - 412 MHz



#### **GPS**

Navigation systems and location services available over network and WiFi



#### **App Store**

Huge range of Apps available to download straight to an iPhone from online store.



#### **Microsoft Exchange support**

Full support for Microsoft Exchange for push email, calendars, and contacts



#### **Scientific calculator**

Calculator feature with many more complex functions and capabilities.



#### **3rd-party native apps support**

iTunes customers were now able to buy 3rd-party apps from the App store.

June  
2009

# 3.0 iOS 3

Release date: June 17, 2009 | Updates: iOS 3.1 | iOS 3.2

Samsung SSPC100 - 600 MHz



## Spotlight Search

Ability to search for specific content on an iPhone or iPad



## MMS

Media Messaging, including video messages and picture messaging, now possible



## Camcorder

iPhone could now record video as well as take pictures

## Cut, copy and paste

Much requested text editing capabilities were introduced to iOS 3.



## Voice Control

Users could use their voice to give specific commands to the iPhone such as "Call Joe Bloggs"

iPhone - Device Support and Backward Compatibility:  
**iPhone 2G, iPhone 3, iPhone 3GS**

iPhone model that launched with specific version of iOS:  
**iPhone 3GS**

iPod Touch Devices

**iPod Touch 1st Gen, iPod Touch 2nd Gen**

iPad - Device Support and Backward Compatibility  
**iPad 1**

iPad model that launched with specific version of iOS  
**iPad 1**



iPhone 3GS



iPad 1

Noteworthy:

**Push notifications for 3rd party apps**

**The iPad debuted on iOS 3.2 coined "iOS meets the Big Screen"**

**Skeuomorphism**

Glossy rich textured icons and the user interface on iOS 3.2 tried to mimic reality with a Skeuomorphic style

June  
2010



iPhone 4



iPad 2

**Noteworthy:**

**Antennagate scandal - users noticed a drop of signal bars when they gripped their iPhones**

**iOS 4.0.1 was released in July 2010 to address the Antennagate issue, by normalizing the number of "bars" displayed**



# iOS 4

Release date: June 21, 2010

Updates: iOS 4.1 | iOS 4.2.1 | iOS 4.2.5 | iOS 4.3



**Apple A4 - 1 GHz**



**Folders**

Apps can be grouped together in folders to create ordered home pages



**HD Recording + Autofocus**

Camera lens now focuses automatically for pictures and video, which now records in high definition



**FaceTime**

Users now able to call another iPhone device via video, enabling virtual face to face conversations over WiFi connections.



**Game Centre**

Lets you play and share games with your friends, track your progress using leaderboards, and more.

**Multitasking**

The multitasking menu was introduced, However iOS 4 did not technically support "true" multitasking



**iPhone - Device Support and Backward Compatibility:**

**iPhone 3, iPhone 3GS, iPhone 4**

iPhone model that launched with specific version of iOS:  
**iPhone 4**

**iPad - Device Support and Backward Compatibility:**

**iPad 1, iPad 2**

iPad model that launched with specific version of iOS  
**iPad 2**

Dec.  
2011

# 5 iOS 5

Realease date: December 10, 2011  
Updates: iOS 5.0.1 | 5.1 | 5.1.1



Apple A5 - 1 GHz



## Siri

Siri was dubbed the "intelligent personal assistant" that got things done just by asking. It allowed the use of voice to send messages, schedule meetings, place phone calls, and more.



## iMessage

Apple's answer to Blackberry's BBM service and unlike SMS was totally free. iMessage tied to either an Apple ID or a phone number.



## Newsstand

News & articles were made available to download and read on and iPhone



## iCloud

Was a new cloud service to replace MobileMe. Apps purchased on one device automatically appeared on other iOS devices



## iTunes Wi-Fi Sync

Automatic syncing of iTunes libraries were now possible over Wi-Fi but only when a device was charging

iPhone - Device Support and Backward Compatibility:

**iPhone 3GS, iPhone 4, iPhone 4S**

iPhone model that launched with specific version of iOS:  
**iOS 5 came out 3 months after launch of iPhone 4S**



iPhone 4S



iPad 3

Noteworthy:

**No PC Required - Apple removed the requirement to physically tether a phone with a computer via USB**

**Battery life was the biggest customer complaint about iOS 5.0. Apple released bugfix update, iOS 5.0.1 to correct this issue**

iPad - Device Support and Backward Compatibility:

**iPad 1, iPad 2, iPad 3**

iPad model that launched with specific version of iOS:  
**iPad 3**

Sept.  
2012



iPhone 5



iPad 4

Noteworthy:

Google Maps was controversially replaced with Apple Maps, which was powered by navigation platform: TomTom

## 6 iOS 6

Release date: September 19, 2012  
Updates: iOS 6.0.1 | iOS 6.0.2 | iOS 6.1



Apple A6 - 1.3 GHz



Panorama

Ability to take up to 360 degree photographs with new panoramic photo software.



Shared Photo Streams

Allowed users to share selections of photos with other iPhone users independently of social networks like Facebook.



FaceTime Video calling over 3G/LTE

Video calling now available over mobile networks - it used to be limited to just Wi-Fi networks.



Apple-sourced maps + Turn-by-turn navigation

Navigation system now with turn by turn instructions similar to other common car sat-nav systems.



Passbook

Keeps airline boarding passes, tickets, and gift cards all in one place, letting you scan your iPhone in their place.

iPhone - Device Support and Backward Compatibility:

**iPhone 3, iPhone 3GS, iPhone 4, iPhone 5**

iPhone model that launched with specific version of iOS:

**iPhone 5**

iPad - Device Support and Backward Compatibility

**iPad 2, 3, 4, Mini**

iPad model that launched with specific version of iOS

**iPad 4 & iPad Mini**

Oct.  
2013



# iOS 7

Release date: October 6, 2013  
Updates: iOS 7.0.1 | 7.0.6 | 7.1 | 7.1.2



Apple A7 - 1.3 GHz



## Touch ID

Fingerprint scanner allows access to an iPhone by placing a finger on the central button.



## Camera Software Update

The camera software had layout and speed improvements. With a "burst" mode option, photos can now be taken in rapid succession to get optimal shots.



## Improved Multitasking

Apps are triggered to download data by push notification; download data requests are grouped together and only triggered when you're in a good signal area.



## AirDrop

Allows for quick and easy beaming of files from one iPhone, iPod touch, or iPad to another over a secure, ad-hoc Bluetooth and Wi-Fi connection



## FaceTime Audio

A high quality codec, audio only version of FaceTime that allows Apple users make calls via 3G, LTE or Wi-Fi

iPhone - Device Support and Backward Compatibility:  
**iPhone 4, iPhone 4S, iPhone 5, iPhone 5C, iPhone 5S**

iPhone model that launched with specific version of iOS:  
**iPhone 5S, 5C**

iPad - Device Support and Backward Compatibility  
**iPad 2, 3, 4, Air, Mini, Mini 2**

iPad model that launched with specific version of iOS  
**iPad Air, iPad Mini 2**



iPad Air

## Noteworthy:

**A visual Overhaul of iOS - skeuomorphic glossy icons replaced by flattened graphics, colorful gradients, and transparencies**

**Tim Cook posts a personal apology on Apple's homepage about the issues with the Apple Map app**



4.7 inches  
iPhone 6

5.5 inches  
iPhone 6 Plus



iPad Air

Sept.  
2014



# iOS 8

Realease date: September 9, 2014 Updates: iOS 8.0.1



Apple A8, Dual-core 2 GHz



## Continuity

Continuity allows the iPhone, iPad, and Mac to remain true to their form-factors while still working seamlessly together. Continuity includes a new version of AirDrop, a new Instant Hotspot for effortless tethering, Handoff to pass app activities back and forth between devices, and the ability to take and make calls and SMS/MMS from your iPhone on your iPad or Mac.



## Apple Pay

Contactless payment finally arrived on iOS 8 but is unfortunately limited to the iPhone 6, 6 Plus and iWatch; which all have built in NFC chips that will store your payment information securely via encryption and require finger print authentication.



## Health App

Health app aggregates health and fitness data like Heart rate, calories burned, blood sugar, cholesterol and sleep patterns that have been collected by other iOS 8 apps. They can all be measured and recorded on an iPhone through apps such as Nike+. Apple is banking on the HealthKit developer software to be



### Third Party Keyboard Support

The ability to install third-party keyboards was one of the most-wanted features that has been added to iOS 8. That means apps such as Swype and Swiftkey will soon be available for iOS 8.



### Family Sharing

Allows parents to pay for and approve purchases made by children from their own device. Also allows up to six people share purchases from iTunes, iBooks and the App Store without having to share the same account. Create sharable family calendars and photo albums.

Noteworthy:

**Apple announced the launch of two versions of the iPhone 6: a 4.7-inch and 5.5-inch, "iPhone 6 Plus". Both of which come with NFC and the "Apple Pay" tap-to-pay system**

**The iWatch:** which according to Apple's CEO, Tim Cook, "is the most personal device we've ever created". It has health monitoring features: measures daily activity, steps taken and heart rate. But must be used with the iPhone.



### iCloud Drive

The new iCloud Drive finally brings comprehensive cloud storage features to iOS 8 and OSX Yosemite. You can store just about anything in iCloud Drive, providing you have sufficient space.



### Wi-Fi Calling

As an alternative to making phone calls over 3G or LTE, iOS 8 has a Wi-Fi Calling feature, that supports sending voice over a Wi-Fi network. It's designed to improve call quality due to the generally faster and less-laggy connection.



### Message App Update

The Messages app now includes options to quickly send voice snippets, videos, group conversations, location sharing and a Do Not Disturb button that mutes a conversation so that it can be read later.

iPhone - Device Support and Backward Compatibility:

**iPhone 4S, iPhone 5, iPhone 5C, iPhone 5S**

iPhone model that launched with specific version of iOS:

**iPhone 6, iPhone 6 Plus**

iPad - Device Support and Backward Compatibility:

**iPad 2, 3, 4, Air, Mini, Mini 2**

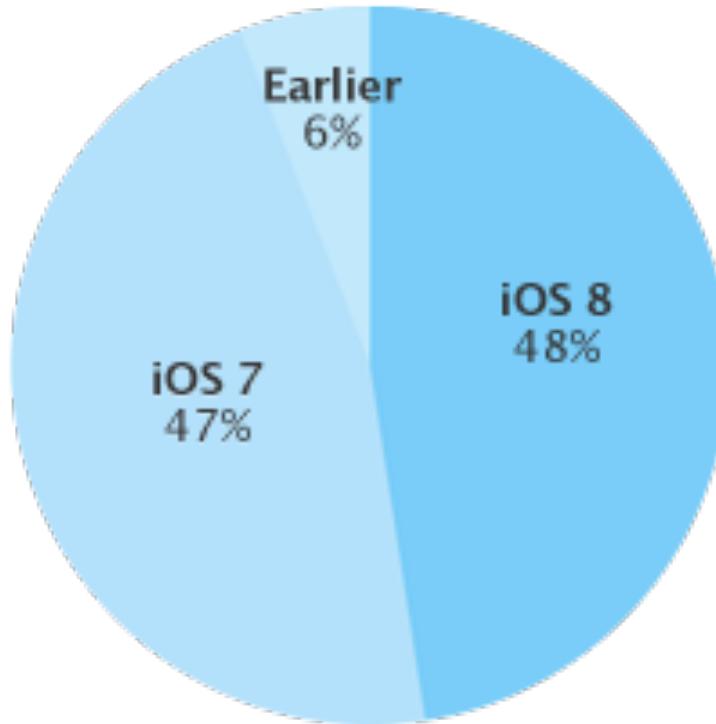
iPad model that launched with specific version of iOS

**N/A**

Noteworthy:

**The iPhone 6's camera sensor has been upgraded to "focus pixels" and now offers DSLR-like phase-detection autofocus that is twice as fast as the 5S. HDR photos can now be taken with single shutter click and videos can be shot at 1080p at 30 or 60 frames per second.**

# IOS VERSION CHART



# HOME SCREEN DESIGN

2007



2008



2009



# HOME SCREEN DESIGN



iPhone OS  
**4**

iOS 4

5.

iOS 5

6

iOS 6

# HOME SCREEN DESIGN



# IPADS



iPad mini



iPad mini 2



iPad mini 3



iPad Air



iPad Air 2

---

Starting at

\$249

Wi-Fi

---

Starting at

\$299

Wi-Fi

---

Starting at

\$399

Wi-Fi

---

Starting at

\$399

Wi-Fi

---

Starting at

\$499

Wi-Fi

# IOS DEVELOPMENT

# IOS DEVELOPER PROGRAM

- The Standard program costs \$99/year. It provides a host of development tools and resources, technical support, distribution of your application via Apple's App Store, and, most importantly, the ability to test and debug your code on an iOS device, rather than just in the simulator.
- Free account is also enough for accessing technical resources, guides, videos, etc.
- To create your developer account, just navigate to <http://developer.apple.com/ios/>

# WHAT'S DIFFERENT ABOUT CODING FOR IOS?

- Only One Active Application
- Only One Window
- Limited Access
- Limited response time
- Limited screen size
- Limited system resources
- Automatic Reference Counting

# iOS SDK

- SDK contains the tools and interfaces needed to develop, install, run, and test native apps that appear on an iOS device's Home screen.
- Native apps are built using the iOS system frameworks and Objective-C language and run directly on iOS.

# IOS SDK ARCHITECTURE

Cocoa Touch

Media

Core Services

Core OS

# IOS SDK ARCHITECTURE

- Cocoa Touch Layer
- Key frameworks for building iOS apps
  - UIKit
  - Multitasking
  - Storyboards and auto layout
  - Event Kit Notifications
  - Address book
  - Game Kit, Map Kit, iAds, Messaging, and Social frameworks

# IOS SDK ARCHITECTURE

- Media Layer
- Graphics, audio, video
  - Core graphics (Quartz), animation, images, OpenGL
  - Assets Library
  - Media Player
  - AV Foundation
  - AirPlay – stream to Apple TV

# IOS SDK ARCHITECTURE

- Core Services Layer
- Foundation Framework – basic types
- Features
  - Location services
  - iCloud
  - In-app purchasing
  - Motion framework
  - Core Data

# IOS SDK ARCHITECTURE

- Core OS Layer
- Low level features
  - Accelerate framework – image and signal processing
  - Core Bluetooth
  - External Accessory Framework
  - Security Framework
  - Low level system features

# DEVELOPING TOOLS

To develop iOS apps, you need:

- A Mac computer running OS X 10.8 (Mountain Lion) or later
- XCode
- iOS SDK (The iOS SDK is included with Xcode)

Mac App Store



Xcode FREE



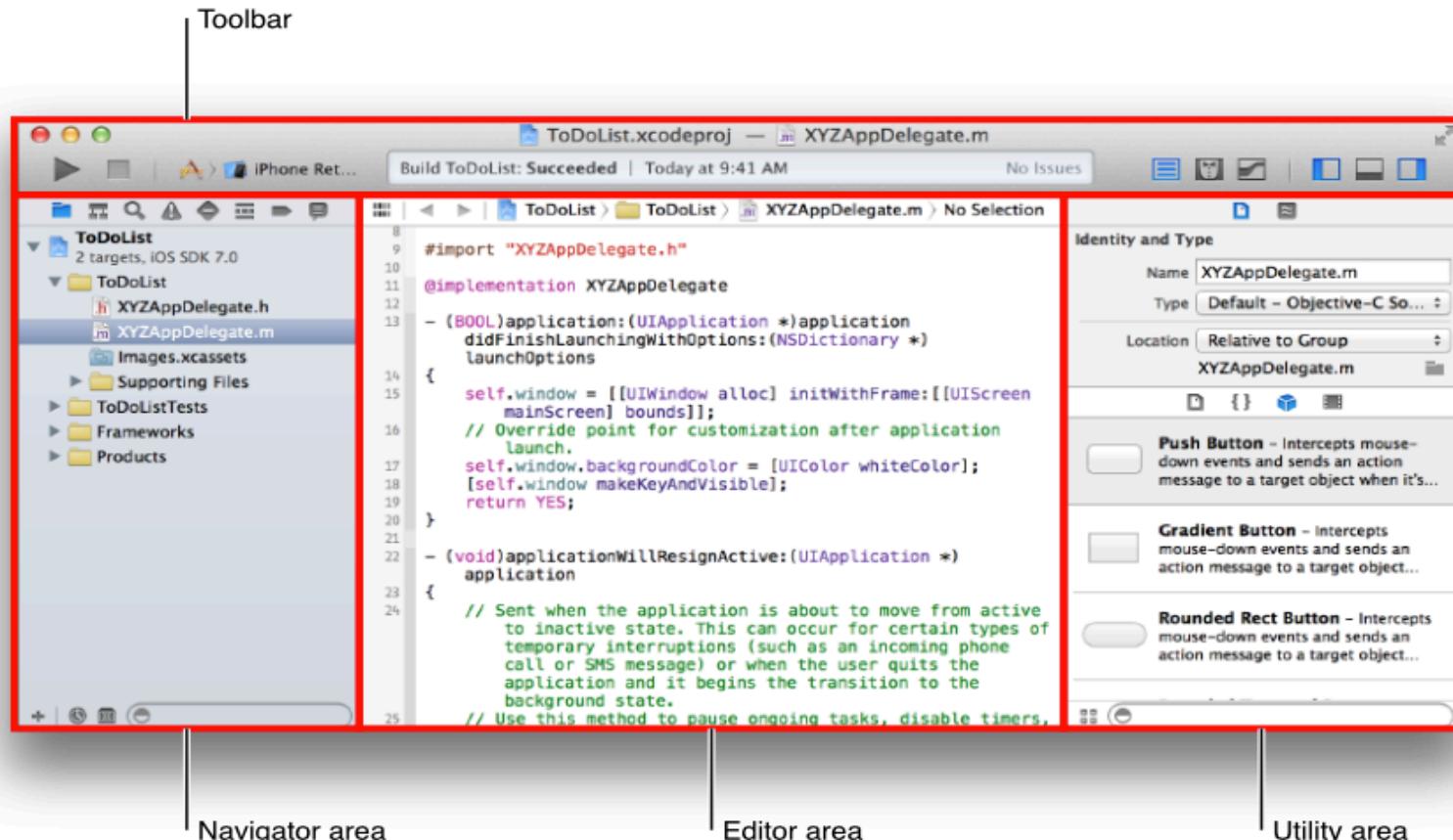
Download



# XCODE

- Xcode is Apple's integrated development environment (IDE).
- The Xcode interface integrates code editing, user interface design, asset management, testing, and debugging within a single workspace window.
-

# XCODE OVERVIEW



# XCODE OVERVIEW

## Toolbar

## Editing Area

## Debugging Area

## Utility Area

## Navigation Area

```
glBindFramebuffer(GL_FRAMEBUFFER, m_defaultFBOName);

[super render];

glBindRenderbuffer(GL_RENDERBUFFER, m_colorRenderbuffer);
[m_context presentRenderbuffer:GL_RENDERBUFFER];

- (BOOL) resizeFromLayer:(CAEAGLLayer *)layer
{
    // The pixel dimensions of the CAEAGLLayer
    GLint backingWidth;
    GLint backingHeight;

    // Allocate color buffer backing based on the current layer size
    glBindRenderbuffer(GL_RENDERBUFFER, m_colorRenderbuffer);
    [m_context renderbufferStorage:GL_RENDERBUFFER fromDrawable:layer];
    glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_WIDTH, &backingWidth);
    glGetRenderbufferParameteriv(GL_RENDERBUFFER, GL_RENDERBUFFER_HEIGHT, &backingHeight);

    glGenRenderbuffers(1, &m_depthRenderbuffer);
    glBindRenderbuffer(GL_RENDERBUFFER, m_depthRenderbuffer);
    glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT16, backingWidth, backingHeight);
    glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER,
        m_depthRenderbuffer);

    [super resizeWithWidth:backingWidth AndHeight:backingHeight];

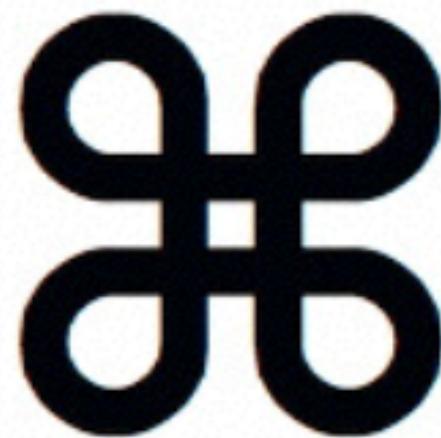
    if (!glCheckFramebufferStatus(GL_FRAMEBUFFER) == GL_FRAMEBUFFER_COMPLETE)
    {
        NSLog(@"Failed to make complete framebuffer object %x", glCheckFramebufferStatus(GL_FRAMEBUFFER));
    }
}
```

- { } C Block typedef – Used for defining a block as a type.
- { } C Inline Block as Variable – Used for saving a block to a variable so we can pass it as an argument multiple times.
- { } C typedef – Used for defining a type.

# MAGIC KEYS



**Shift**

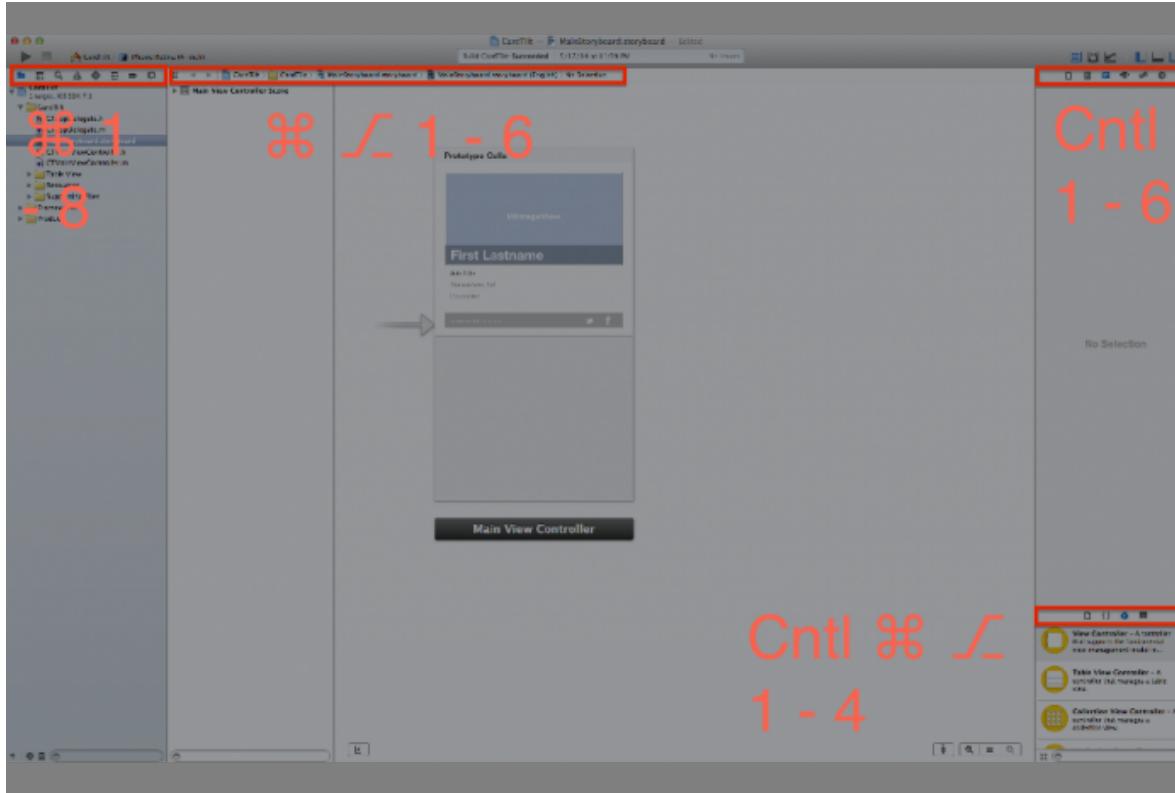


**Command**



**Option** (Alt)

# HOT KEYS

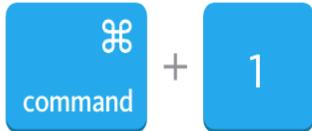


# XCODE SHORTCUTS



## Project Navigator

Quickly view all your code, images, and user interface files.



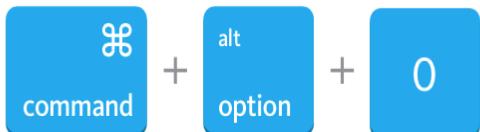
## Show/Hide Navigator Panel

Toggle left panel open and closed.



## Show/Hide Utilities Panel

Toggle right panel open and closed.



# XCODE SHORTCUTS



## Open File in Assistant Editor

Show multiple code or interface files with the assistant editor.

alt  
option

+

Left-Click  
File



## Find Navigator (Search)

Quickly find variables or methods.

⌘  
command

+

shift

+

F

## File Jump Bar

Jump to a line of code within your current file.

control

+

6

## Open Quickly

Open a file quickly or jump to a specific method.

⌘  
command

+

shift

+

O

*letter*

# XCODE SHORTCUTS

## Jump Bar / Open Quickly Search Tip

Only type the first letter of each word in a method.

## Type Method Abbreviations

### (Obj-c) Switch Between .h and .m

Switch between two related code files.



### Run Xcode App

Use this often while writing code to catch bugs.



### Clean Xcode App

Use when Xcode freezes or fails to work.



# XCODE SHORTCUTS

## Build Xcode App

Build instead of Run when making minor changes.



## Documentation & Reference

Learn about code.



## Quick Help

Inline help while writing code.



# APP BUNDLE

Xcode packages every app  
as a bundle

# WHAT IS APP BUNDLE ?

- A **bundle** is a directory in the file system that groups related resources together in one place.
- An iOS app bundle contains the app executable file and supporting resource files such as app icons, image files, and localized content.

# A TYPICAL APP BUNDLE

- App Executable
- The information property file
- App Icons
- Launch Images
- Storyboard files
- Ad Hoc Distribution Icon
- Settings Bundle
- Non Localized Resource Files
- Sub Directories For Localized Resources

# APP EXECUTABLE

- Example
  - MyApp
- The executable file contains your app's compiled code. The name of your app's executable file is the same as your app name minus the .app extension.
- This file is required.

# THE INFORMATION PROPERTY LIST FILE

- Example
  - Info.plist
- The Info.plist file contains configuration data for the app. The system uses this data to determine how to interact with the app.
- Xcode uses information from the General, Capabilities, and Info tabs of your project to generate an information property list (Info.plist) file for your app at compile time.
- Whenever possible, use the General and Capabilities tabs to specify the configuration information for your app. Those tabs contain the most common configuration options available for apps. If you do not see a specific option on either of those tabs, use the Info tab.
- This file is required and must be called Info.plist.

# APP ICONS

- Example
  - Icon.png
  - Icon@2x.png
  - Icon-Small.png
  - Icon-Small@2x.png
- Your app icon is used to represent your app on the device's Home screen. Other icons are used by the system in appropriate places. Icons with @2x in their filename are intended for devices with Retina displays.
- An app icon is required.

# LAUNCH IMAGES

- Example
  - Default.png
  - Default-Portrait.png
  - Default-Landscape.png
- The system uses this file as a temporary background while your app is launching. It is removed as soon as your app is ready to display its user interface.
- At least one launch image is required.

# STORYBOARD FILES (OR NIB FILES)

- MainBoard.storyboard
- Storyboards contain the views and view controllers that the app presents on screen. Views in a storyboard are organized according to the view controller that presents them. Storyboards also identify the transitions (called segues) that take the user from one set of views to another.
- The name of the main storyboard file is set by Xcode when you create your project. You can change the name by assigning a different value to the `UIStoryboardFile` key in the `Info.plist` file.) Apps that use nib files instead of storyboards can replace the `UIStoryboardFile` key with the `NSMainNibFile` key and use that key to specify their main nib file.
- The use of storyboards (or nib files) is optional

# AD HOC DISTRIBUTION ICON

- Example
  - iTunesArtwork
- If you are distributing your app ad hoc, include a 512 x 512 pixel version of your app icon. This icon is normally provided by the App Store from the materials you submit to iTunes Connect. However, because apps distributed ad hoc do not go through the App Store, your icon must be present in your app bundle instead. iTunes uses this icon to represent your app.

# SETTINGS BUNDLE

- Example
  - Settings.bundle
- If you want to expose custom app preferences through the Settings app, you must include a settings bundle. This bundle contains the property list data and other resource files that define your app preferences. The Settings app uses the information in this bundle to assemble the interface elements required by your app.
- This bundle is optional.

# NONLOCALIZED RESOURCE FILES

- Example
  - sun.png, mydata.plist
- Nonlocalized resources include things like images, sound files, movies, and custom data files that your app uses.
- All of these files should be placed at the top level of your app bundle.

# SUBDIRECTORIES FOR LOCALIZED RESOURCES

- Example
  - en.lproj , fr.lproj ,es.lproj
- Localized resources must be placed in language-specific project directories, the names for which consist of an ISO 639-1 language abbreviation plus the .lproj suffix. (For example, the en.lproj, fr.lproj, and es.lproj directories contain resources localized for English, French, and Spanish.)
- An iOS app should be internationalized and have a *language* .lproj directory for each language it supports. In addition to providing localized versions of your app's custom resources, you can also localize your app icon, launch images, and Settings icon by placing files with the same name in your language-specific project directories.

# IOS APPLICATION LIFECYCLE

# THE MAIN FUNCTION

- The entry point for every C-based app is the main function and iOS apps are no different. What is different is that for iOS apps you do not write the main function yourself. Instead, Xcode creates this function as part of your basic project.

```
#import <UIKit/UIKit.h>
#import "AppDelegate.h"
int main(int argc, char * argv[])
{
    @autoreleasepool
    {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}
```

- The UIApplicationMain function handles this process by creating the core objects of your app, loading your app's user interface from the available storyboard files, calling your custom code so that you have a chance to do some initial setup, and putting the app's run loop in motion.
- The only pieces that we have to provide are the storyboard files and the custom initialization code.

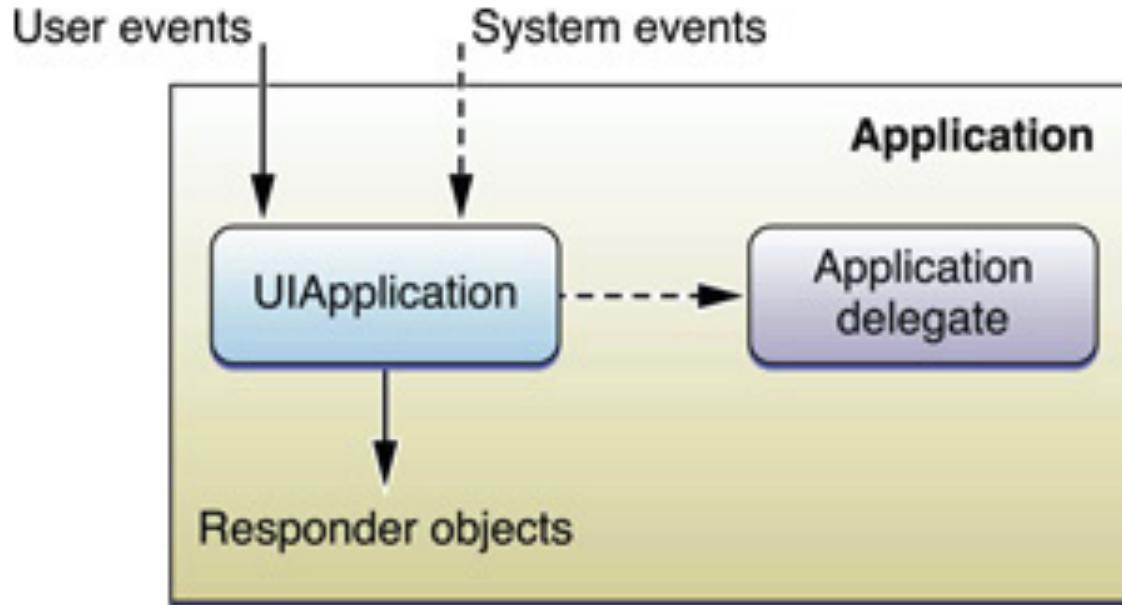
# KEY OBJECTS IN IOS APP

- UIApplication object
- App delegate object
- UIWindow object
- View objects, control objects, and layer objects
- View controller objects
- Documents and data model objects

# UIAPPLICATION OBJECT

- The UIApplication object manages the event loop and other high-level app behaviors. It also reports key app transitions and some special events (such as incoming push notifications) to its delegate, which is a custom object you define.
- Use the UIApplication object as is—that is, without subclassing.
- An Application has a single application object

# UIAPPLICATION OBJECT



# APP DELEGATE OBJECT

- The app delegate is the heart of your custom code. This object works in tandem with the UIApplication object to handle app initialization, state transitions, and many high-level app events.
- This object is also the only one guaranteed to be present in every app, so it is often used to set up the app's initial data structures.

# UIWINDOW OBJECT

- A UIWindow object coordinates the presentation of one or more views on a screen. Most apps have only one window, which presents content on the main screen, but apps may have an additional window for content displayed on an external display.
- To change the content of your app, you use a view controller to change the views displayed in the corresponding window. You never replace the window itself.
- In addition to hosting views, windows work with the UIApplication object to deliver events to your views and view controllers.

# VIEW OBJECTS, CONTROL OBJECTS, AND LAYER OBJECTS

- Views and controls provide the visual representation of your app's content. A **view** is an object that draws content in a designated rectangular area and responds to events within that area. **Controls** are a specialized type of view responsible for implementing familiar interface objects such as buttons, text fields, and toggle switches.
- The **UIKit** framework provides standard views for presenting many different types of content. You can also define your own custom views by subclassing **UIView** (or its descendants) directly.
- In addition to incorporating views and controls, apps can also incorporate **Core Animation** layers into their view and control hierarchies. **Layer objects** are actually data objects that represent visual content. Views use layer objects intensively behind the scenes to render their content. You can also add custom layer objects to your interface to implement complex animations and other types of sophisticated visual effects.

# VIEW CONTROLLER OBJECTS

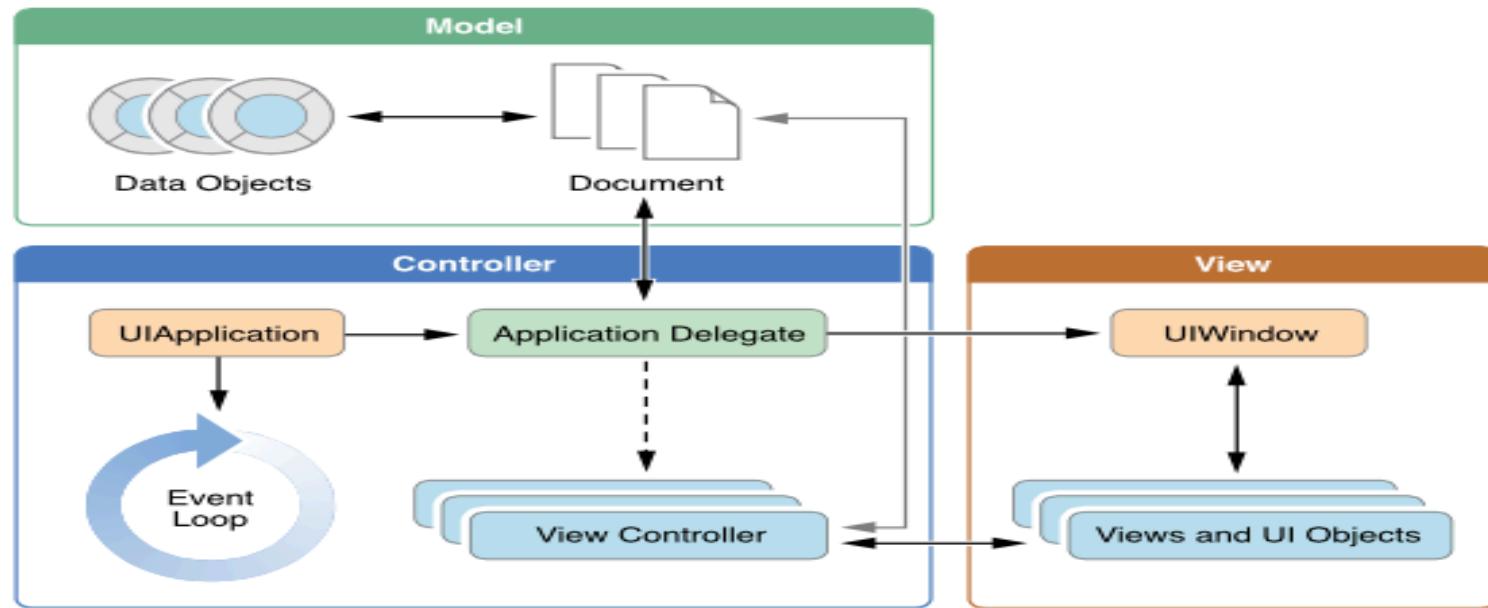
- **View controller objects** manage the presentation of your app's content on screen. A view controller manages a single view and its collection of subviews. When presented, the view controller makes its views visible by installing them in the app's window.
- The `UIViewController` class is the base class for all view controller objects. It provides default functionality for loading views, presenting them, rotating them in response to device rotations, and several other standard system behaviors. UIKit and other frameworks define additional view controller classes to implement standard system interfaces such as the image picker, tab bar interface, and navigation interface.

# DATA MODEL OBJECTS

- **Data model objects** store your app's content and are specific to your app. For example, a banking app might store a database containing financial transactions, whereas a painting app might store an image object or even the sequence of drawing commands that led to the creation of that image. (In the latter case, an image object is still a data object because it is just a container for the image data.)

# KEY OBJECTS IN IOS APP

Key objects in an iOS app

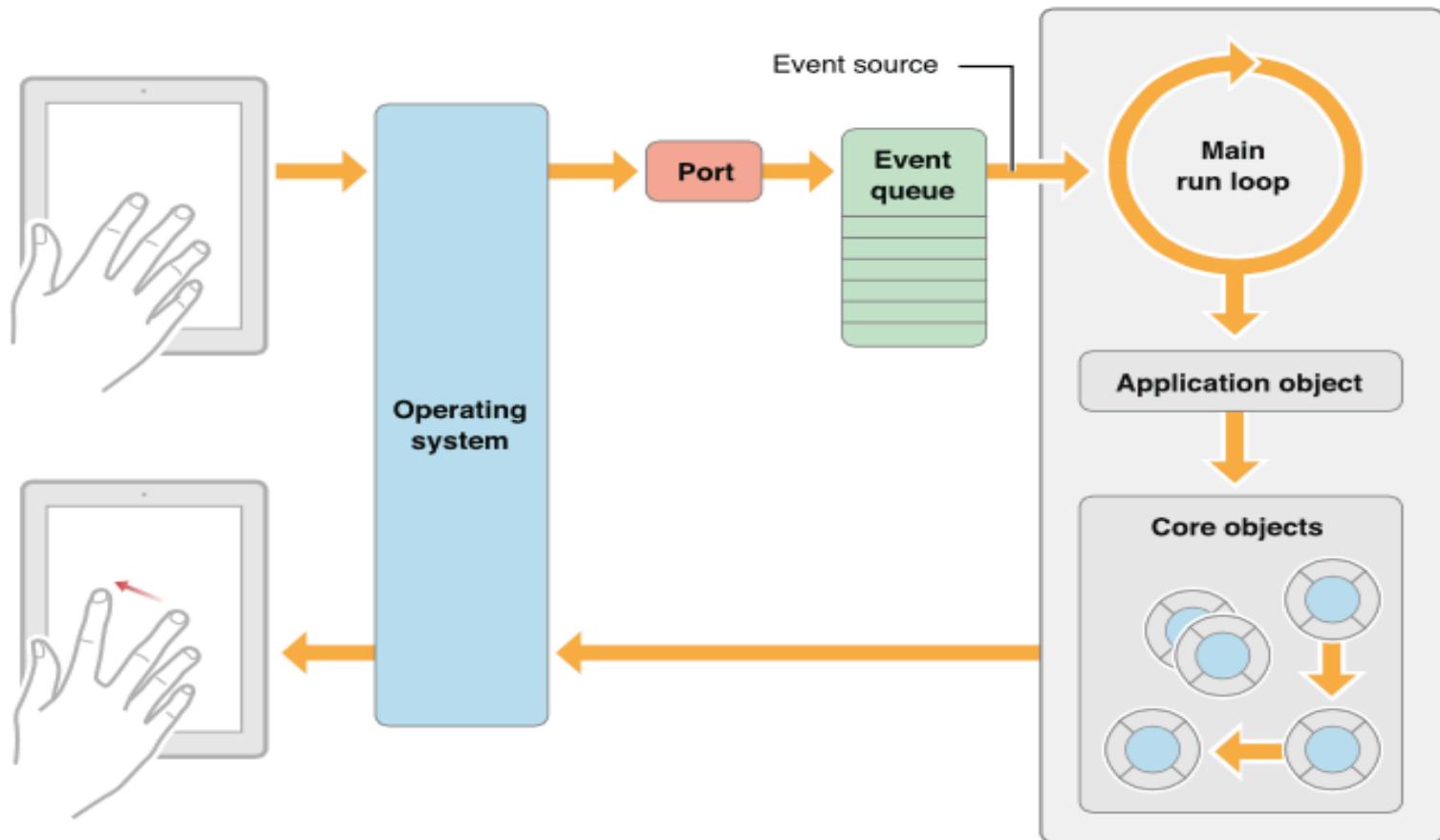


- Custom Objects
- System Objects
- Either system or custom objects

# THE MAIN RUN LOOP

- An app's **main run loop** processes all user-related events. The `UIApplication` object sets up the main run loop at launch time and uses it to process events and handle updates to view-based interfaces. As the name suggests, the main run loop executes on the app's main thread. This behavior ensures that user-related events are processed serially in the order in which they were received.

# MAIN RUN LOOP



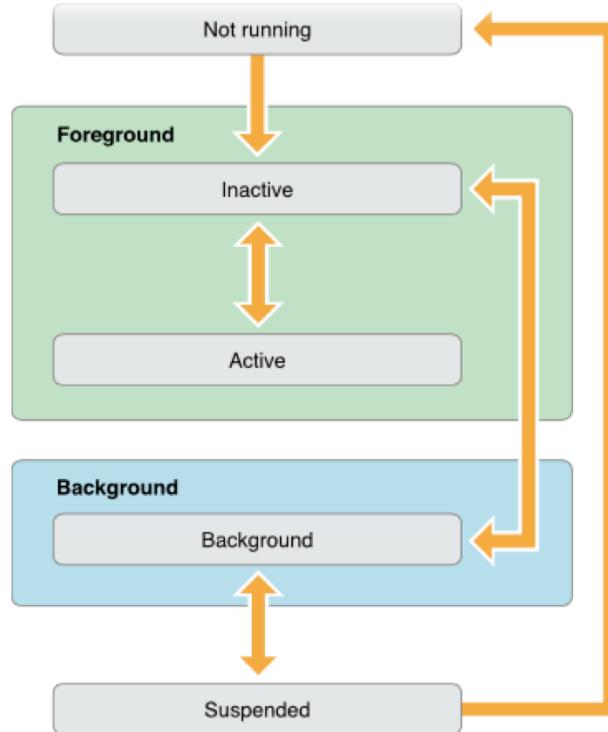
# EXECUTION STATES FOR APPS

- **Not running**
  - The app has not been launched or was running but was terminated by the system.
- **Inactive**
  - The app is running in the foreground but is currently not receiving events. (It may be executing other code though.) An app usually stays in this state only briefly as it transitions to a different state.
- **Active**
  - The app is running in the foreground and is receiving events. This is the normal mode for foreground apps

# EXECUTION STATES FOR APPS

- **Background**
  - The app is in the background and executing code. Most apps enter this state briefly on their way to being suspended. However, an app that requests extra execution time may remain in this state for a period of time. In addition, an app being launched directly into the background enters this state instead of the inactive state.
- **Suspended**
  - The app is in the background but is not executing code. The system moves apps to this state automatically and does not notify them before doing so. While suspended, an app remains in memory but does not execute any code.
  - When a low-memory condition occurs, the system may purge suspended apps without notice to make more space for the foreground app.

# EXECUTION STATES



# STATE TRANSITIONS

- Most state transitions are accompanied by a corresponding call to the methods of app delegate object.
- **application:willFinishLaunchingWithOptions:**
  - This method is your app’s first chance to execute code at launch time.
- **application:didFinishLaunchingWithOptions:**
  - This method allows you to perform any final initialization before your app is displayed to the user.
- **applicationDidBecomeActive:**
  - Lets your app know that it is about to become the foreground app. Use this method for any last minute preparation.

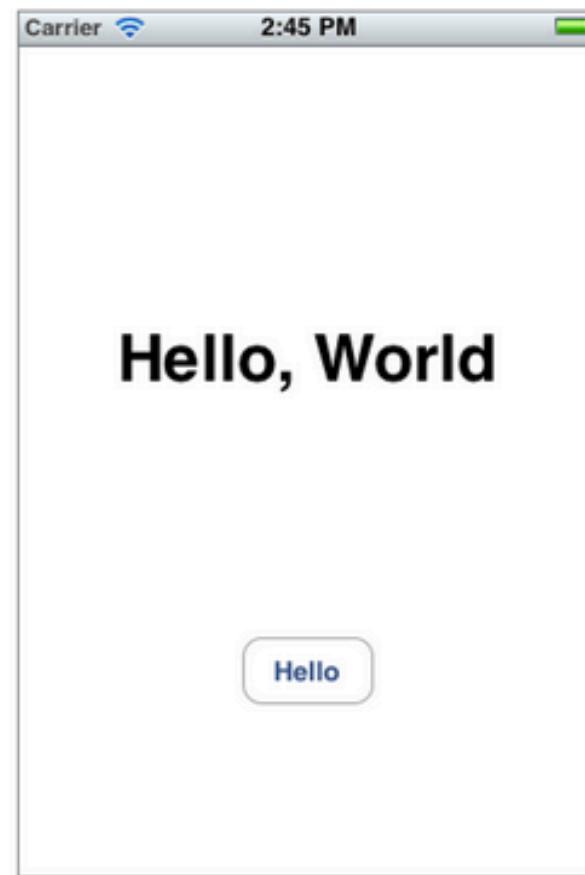
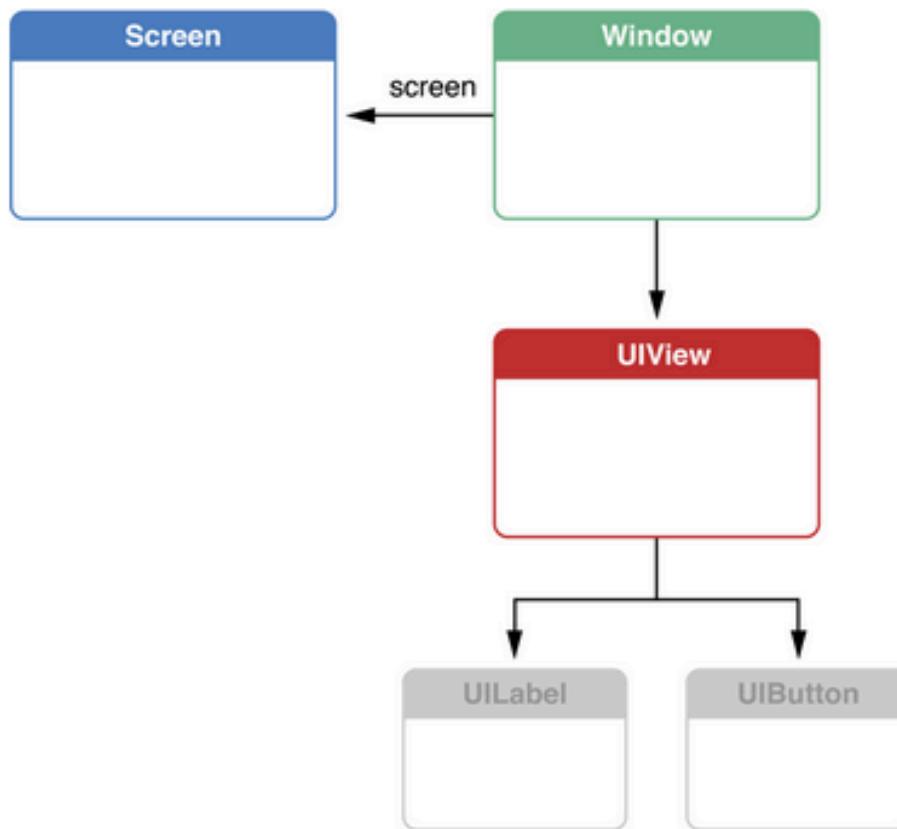
# STATE TRANSITIONS

- **applicationWillResignActive:**
  - Lets you know that your app is transitioning away from being the foreground app. Use this method to put your app into a quiescent state.
- **applicationDidEnterBackground:**
  - Lets you know that your app is now running in the background and may be suspended at any time.
- **applicationWillEnterForeground:**
  - Lets you know that your app is moving out of the background and back into the foreground, but that it is not yet active.
- **applicationWillTerminate:**
  - Lets you know that your app is being terminated. This method is not called if your app is suspended.

# HELLOWORLD APP PROGRAMMATICALLY

Demo

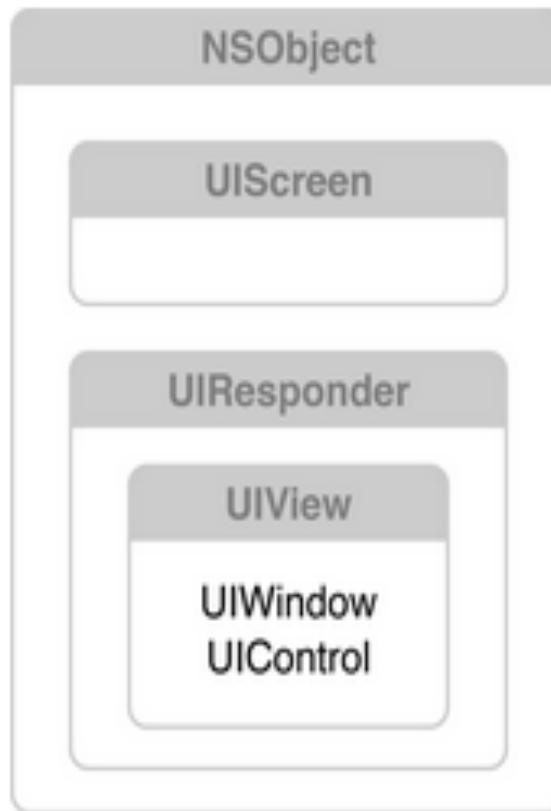
# HELLO WORLD APP



# MAJOR OBJECTS

- There are three major objects at work here:
- A UIScreen object that identifies a physical screen connected to the device.
- A UIWindow object that provides drawing support for the screen.
- A set of UIView objects to perform the drawing. These objects are attached to the window and draw their contents when the window asks them to.

# CLASSES IN THE VIEW SYSTEM



# VIEWS

- A view (an object whose class is `UIView` or a subclass of `UIView`) knows how to draw itself into a rectangular area of the interface. Your app has a visible interface thanks to views.
- A view is also a responder (`UIView` is a subclass of `UIResponder`). This means that a view is subject to user interactions, such as taps and swipes. Thus, views are the basis not only of the interface that the user sees, but also of the interface that the user touches

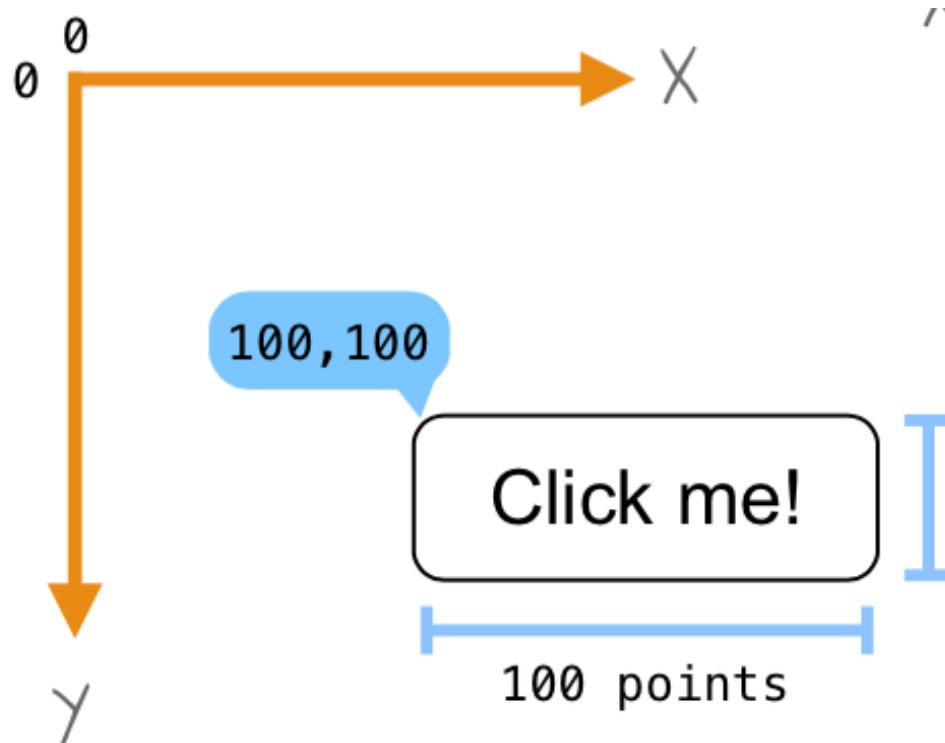
# VIEW HIERARCHY

- The *view hierarchy* is the chief mode of view organization. A view can have subviews; a subview has exactly one immediate superview. Thus there is a tree of views. This hierarchy allows views to come and go together. If a view is removed from the interface, its subviews are removed; if a view is hidden (made invisible), its subviews are hidden; if a view is moved, its subviews move with it; and other changes in a view are likewise shared with its subviews.
- The view hierarchy is also the basis of, though it is not identical to, the responder chain.

# SUBVIEW AND SUPERVIEW

- A `UIView` has a `Superview` property (a `UIView`) and a `subviews` property (an `NSArray` of `UIViews`, in back-to-front order)
- The method `addSubview:` makes one view a subview of another; `removeFromSuperview` takes a subview out of its superview's view hierarchy. In both cases, if the superview is part of the visible interface, the subview will appear or disappear; and of course this view may itself have subviews that accompany it.

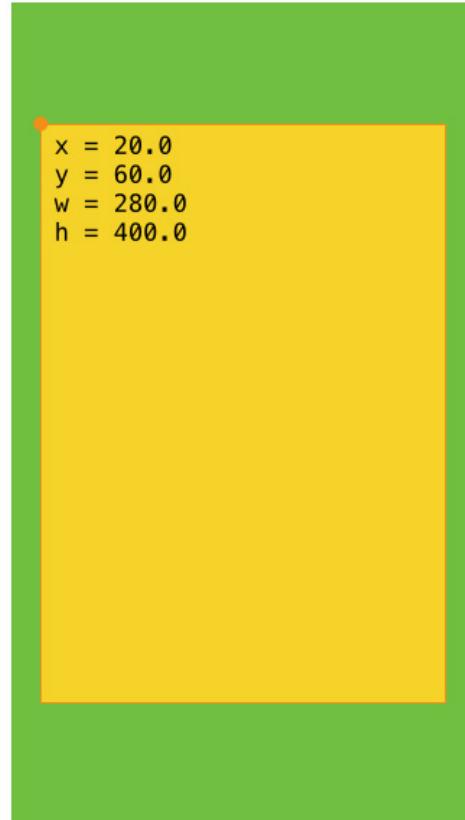
# COORDINATE SYSTEM



# FRAME

- As the documentation clarifies, the frame of a view is a structure, a `CGRect`, that defines the size of the view and its position in the view's superview, the superview's coordinate system.

# FRAME



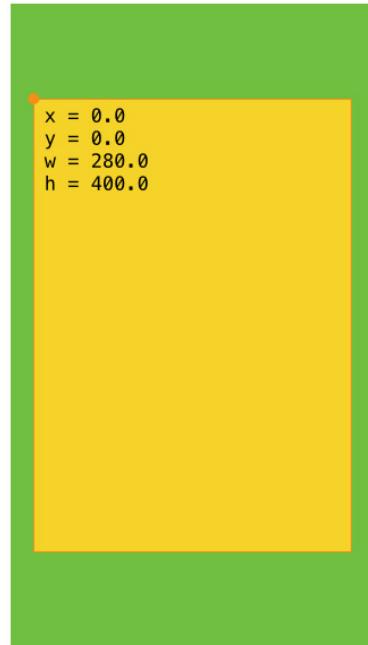
# FRAME

UIView's <b>frame</b> property:	The <b>CGPoint</b> data type:	The <b>CGSize</b> data type:
<pre>struct CGRect {     CGPoint origin;     CGSize size; };</pre>	<pre>struct CGPoint {     CGFloat x;     CGFloat y; };</pre>	<pre>struct CGSize {     CGFloat width;     CGFloat height; };</pre>

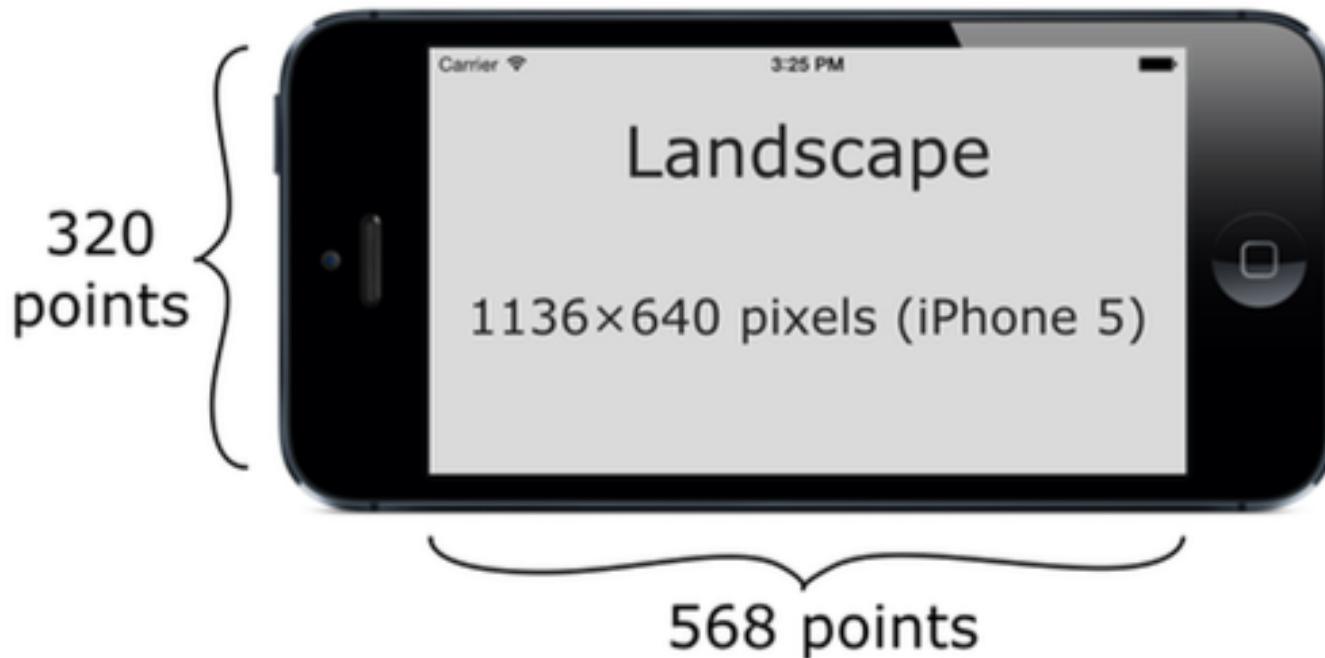
# BOUNDS

- The bounds property of a view defines the size of the view and its position in the view's own coordinate system. This means that in most cases the origin of the bounds of a view are set to  $\{0,0\}$  as shown in the following diagram. The view's bounds is important for drawing the view.
- When the frame property of a view is modified, the view's center and/or bounds are also modified.

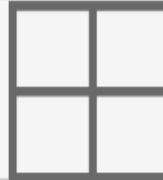
# BOUNDS



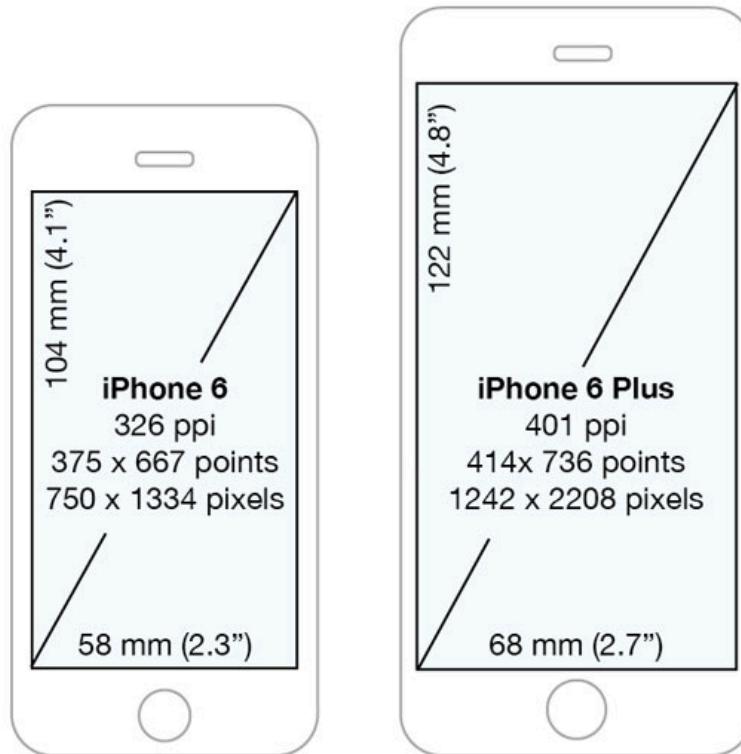
# POINTS AND PIXELS



# PRE IOS8

	Points	Pixels
<b>Non Retina iPhone</b> <b>1 point = 1 pixel</b>		<b>320x480</b> 
<b>iPhone 4 (Retina Screen)</b> <b>1 point = 4 pixels</b>		<b>320x480</b> 
<b>iPhone 5 (Retina Screen)</b> <b>1 point = 4 pixels</b>		<b>320x568</b> 

# IOS 8



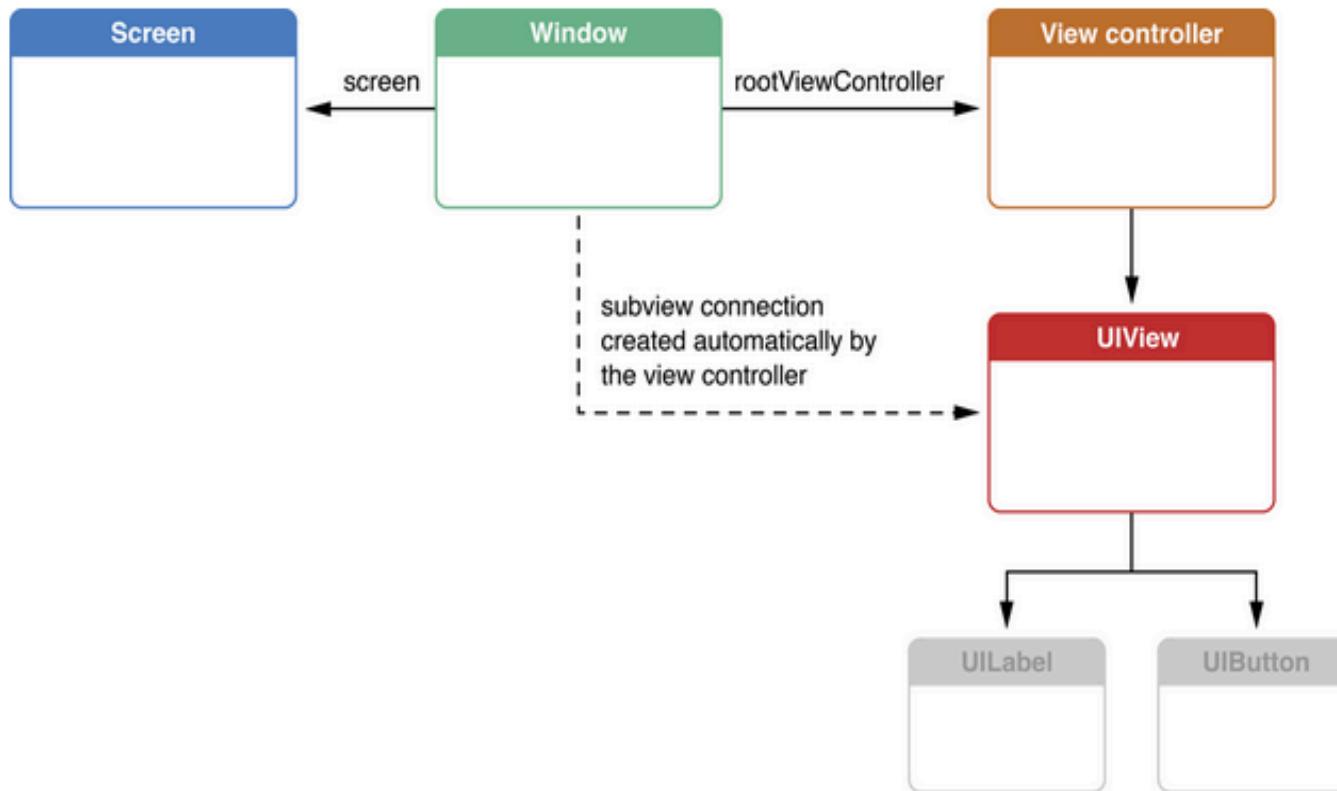
# POST IOS 8

<b>Device</b>	<b>Form factor</b>	<b>Screen dimension in points</b>
iPhone 3GS and older	3.5"	320×480
iPhone 4 and 4S	3.5"	320×480
iPhone 5, 5c and 5s	4"	320×568
iPhone 6	4.7"	375×667
iPhone 6 Plus	5.5"	414×736
iPad (all models)	7.9" and 9.7"	768×1024

# VIEW CONTROLLER MANAGES VIEWS

- Each view controller organizes and controls a view; this view is often the root view of a view hierarchy. View controllers are controller objects in the MVC pattern, but a view controller also has specific tasks iOS expects it to perform.
- These tasks are defined by the UIViewController class that all view controllers inherit from.
- All view controllers perform view and resource management tasks; other responsibilities depend on how the view controller is used.

# VIEW CONTROLLER IN HELLO WORLD APP



# WINDOW IS LIKE CANVAS

Canvas



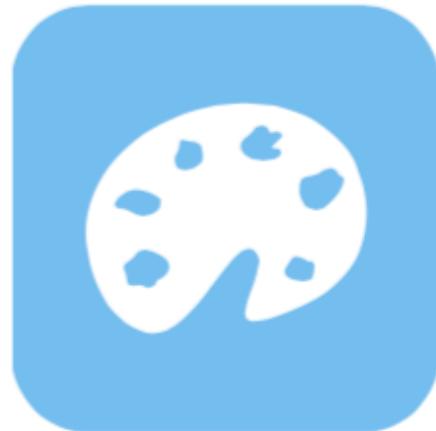
Window

Paint Brush



ViewController

Paint



View

# UIRESPONDER

**UIResponder**

Interface for responding to events

**UIViewController**

Inherits from  
parent class



View Controller



View

Gives us access to touch events

- touchesBegan:withEvent:
- touchesMoved:withEvent:
- touchesEnded:withEvent:
- touchesCancelled:withEvent:

# CREATE A BUTTON

```
UIButton *firstButton = [UIButton buttonWithType:UIButtonTypeRoundedRect];
```

UIResponder



UIView



UIButton

Inherits from



**UIButtonTypeRoundedRect**



**UIButtonTypeDetailDisclosure**



**UIButtonTypeInfoLight**



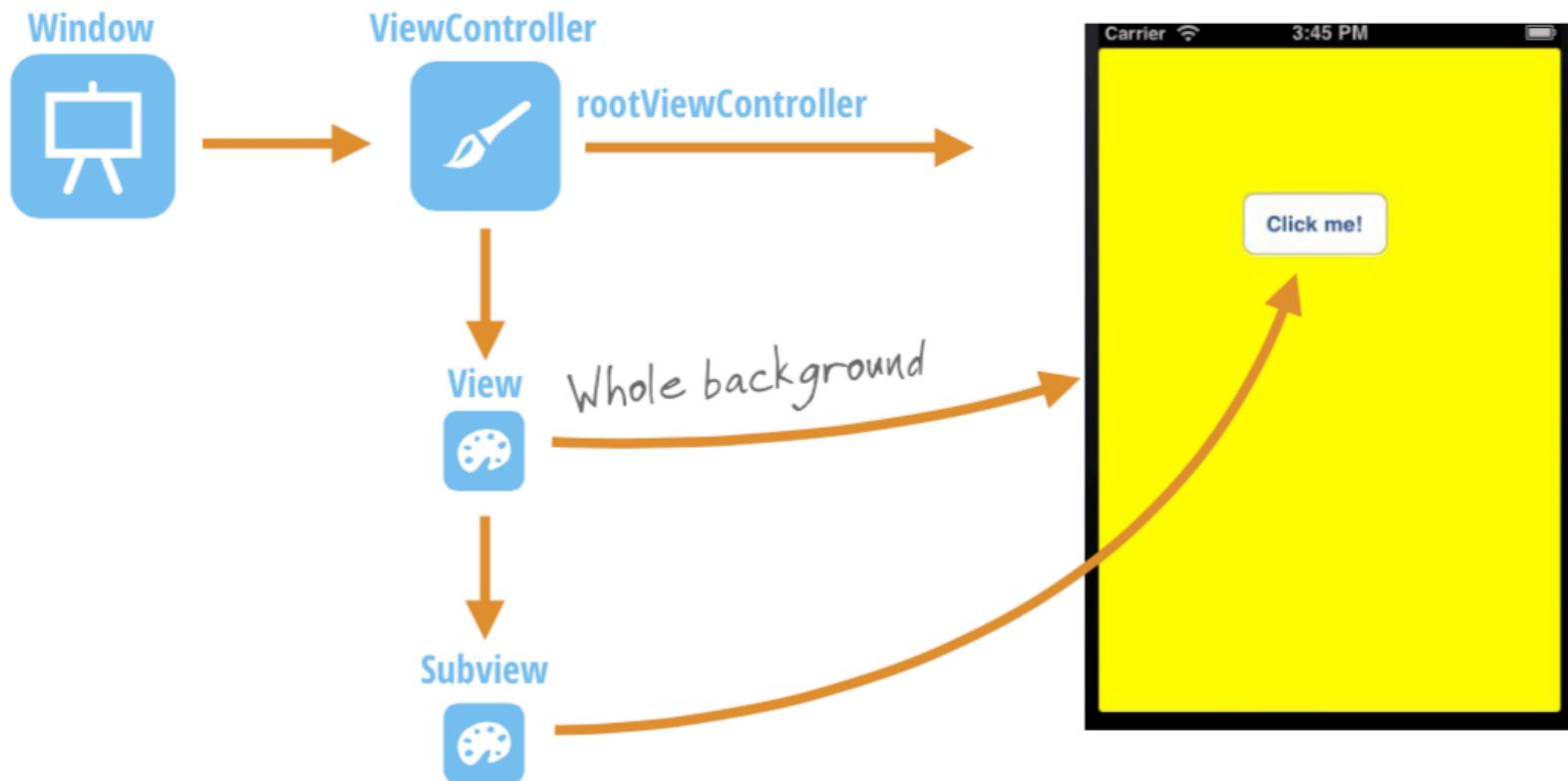
**UIButtonTypeInfoDark**



**UIButtonTypeContactAdd**



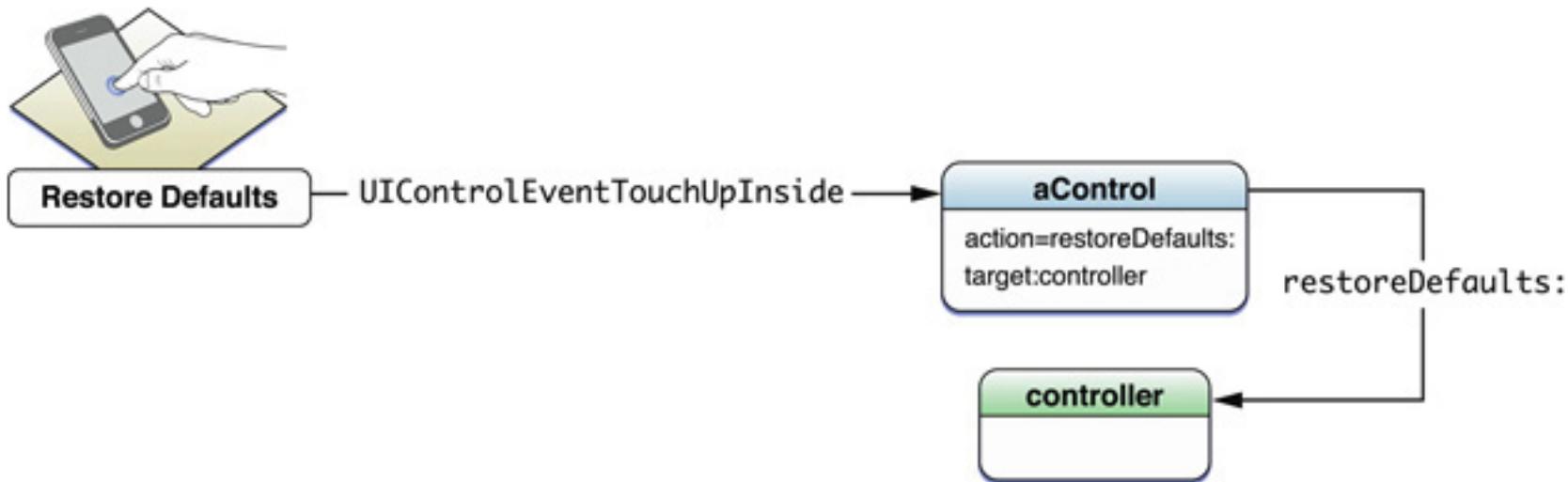
# REMEMBER THIS



# CONTROL OBJECT

- A control is a type of view in a user interface that sends a message to another object when a user manipulates it in a certain way, such as tapping a button or dragging a slider.
- A control is the agent in the target-action model.
- A control stores the information necessary for sending the message: a reference to the object to receive the message (the target) and a selector that identifies the method to invoke on the target (the action). When a user manipulates the control in a specific way, it sends a message to the application object, which then forwards the action message to the target.

# CONTROL OBJECT



# BUTTON CONTROL



UIControl

base class for control objects which convey user intent

`addTarget:action:forControlEvents:`

Can send this to the UIButton object to listen for events

Inherits from

# TARGET-ACTION

`addTarget:action:forControlEvents:`

```
- (void)addTarget:(id)target           instance method  
          action:(SEL)action  
forControlEvents:(UIControlEvents)controlEvents;
```

**target** the target object to which the message will be sent

**action** the message to send to the target

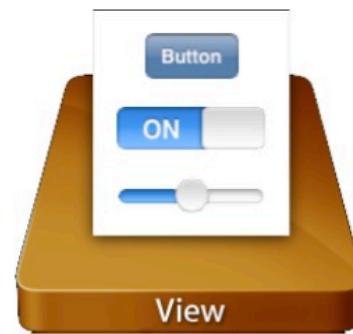
**controlEvents** the event to listen for

# IOSDESIGN PATTERNS

# DESIGN PATTERNS

- Model-View-Controller
- View Controllers
- Actions and Outlets
- Delegation

# MODEL-VIEW-CONTROLLER



# MODEL

- **Custom Classes**
- **Responsibilities**
  - Domain / Business Logic (Independent of the View!)
  - Data Storage (Properties, KVC, NSCoding, CoreData)
- **Examples**
  - CUser Data (Document, Person, BankAccount, ...)
  - Application Data (Preferences, State, ...)
  - Functionality (Libraries, Algorithms, ...)

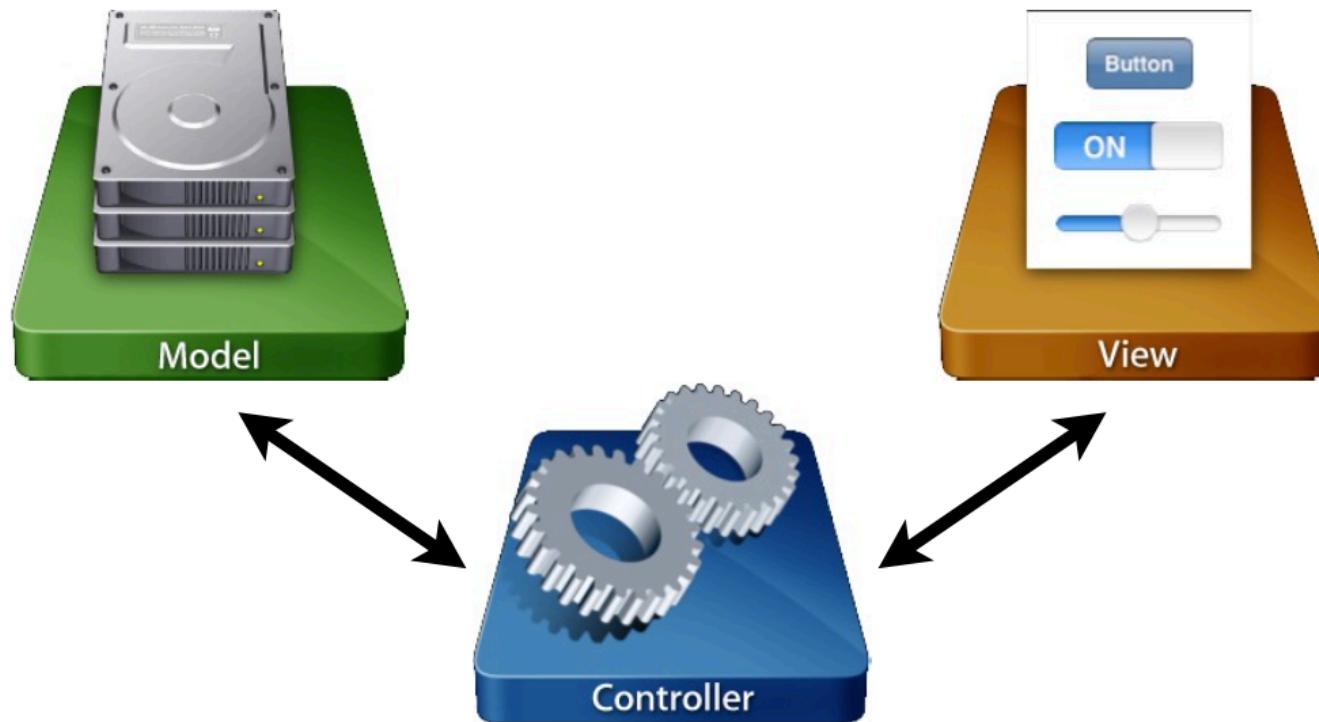
# VIEW

- User Interface Elements
- Subclass of UIView
- Responsibilities
  - Look and Feel of the UI
  - Data Visualization
  - Generate UI Events
- Examples
  - UIButton, UILabel, UITableView, MyCustomView, ...

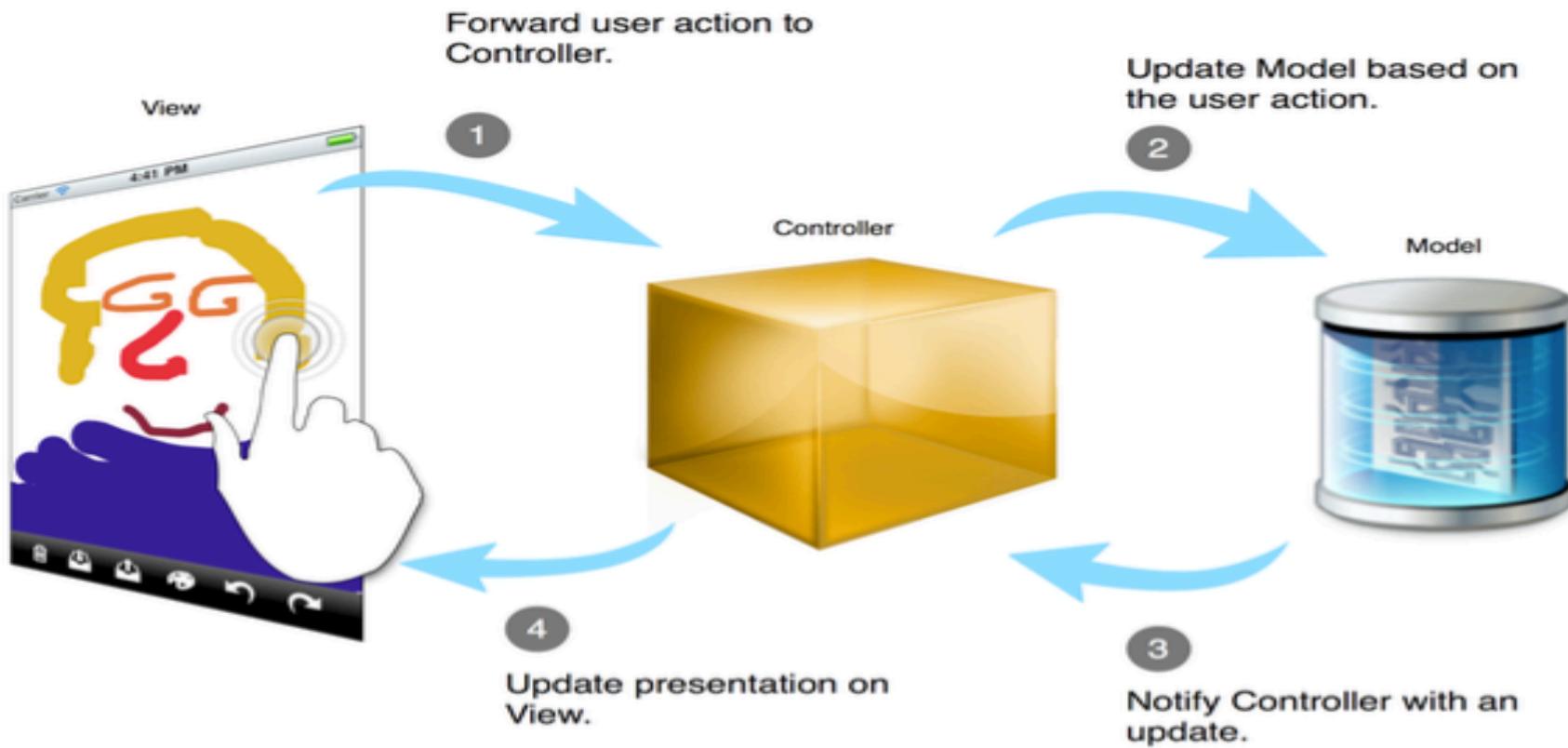
# CONTROLLER

- Glue-code between Model and View
- Custom Classes, Subclass of UIViewController
- Responsibilities:
  - Configure View according to Model
  - React to UI Events and update Model
  - Application Logic (Navigation, flow of control)
- Examples
  - UIViewController, MyViewController, ...

# MODEL-VIEW-CONTROLLER



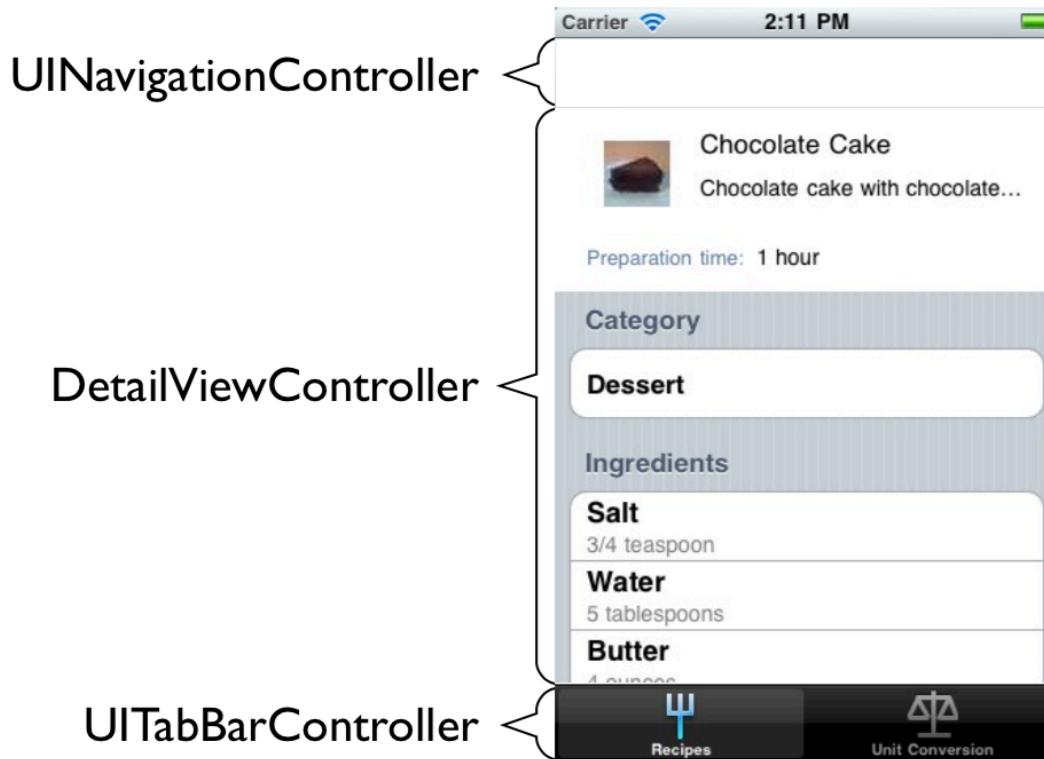
# MODEL-VIEW-CONTROLLER



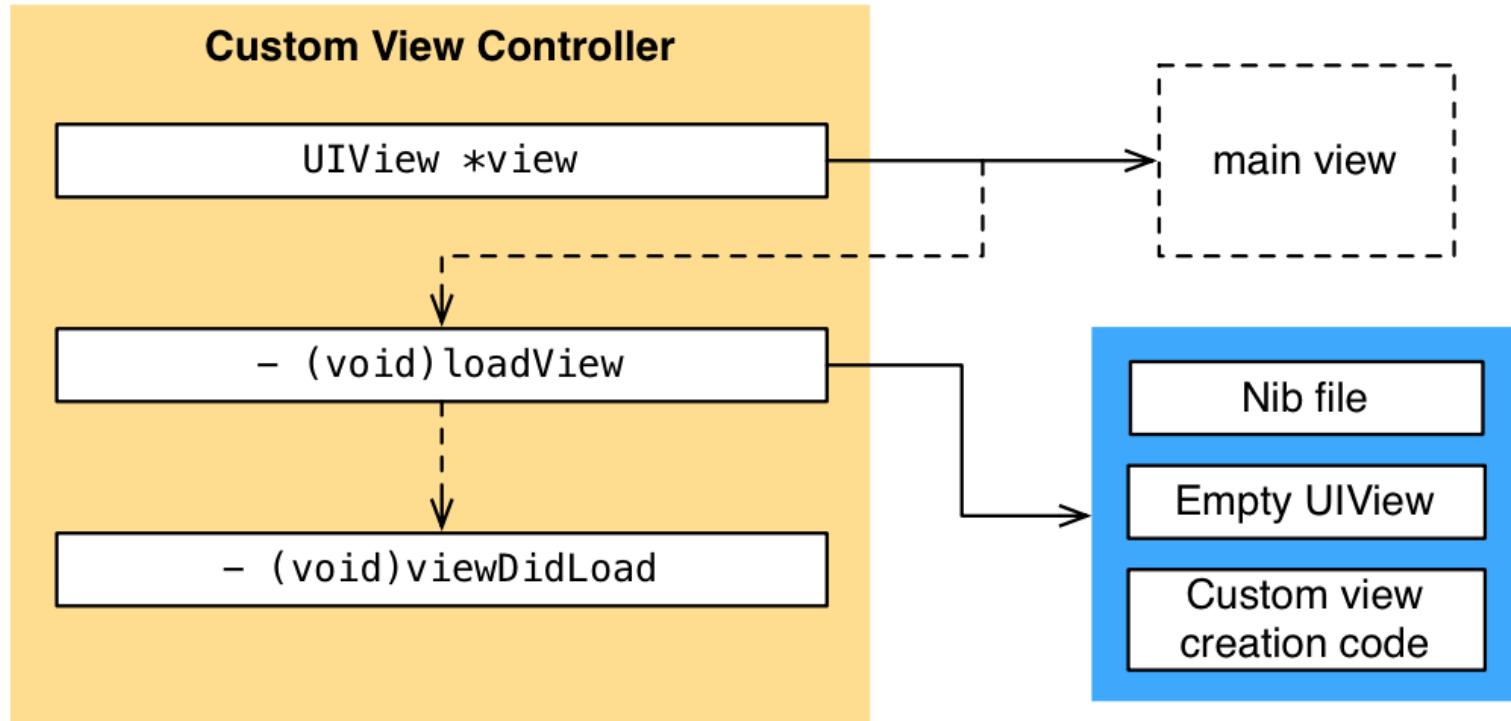
# VIEW CONTROLLERS

- Base class for a controller that manages a view
- **UIViewController simplifies standard behavior:**
  - load resources from a Nib file
  - configure navigation bar, tab bar, and tool bars
  - pluggable architecture
  - handle events and memory warnings
  - manage interface orientation changes

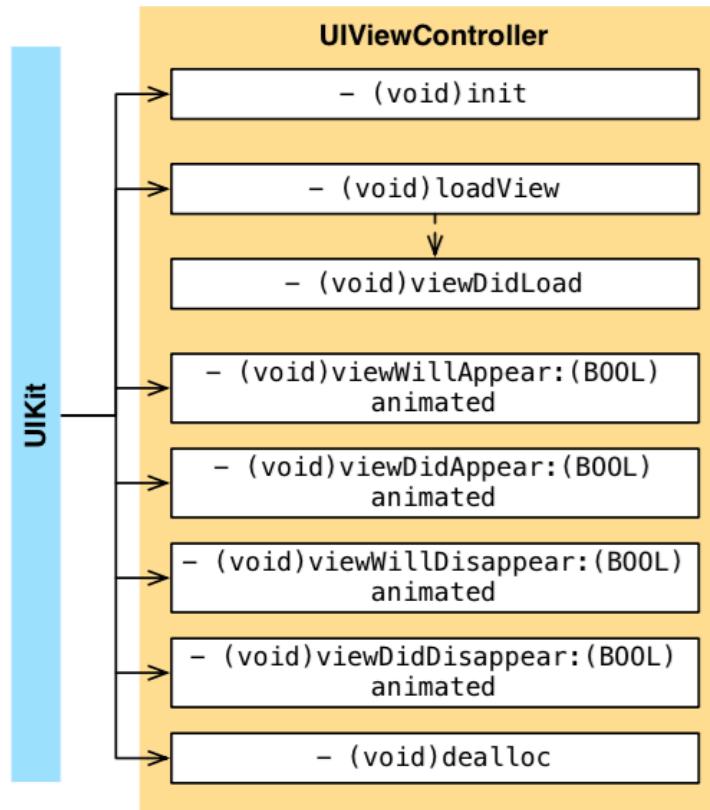
# VIEWCONTROLLER



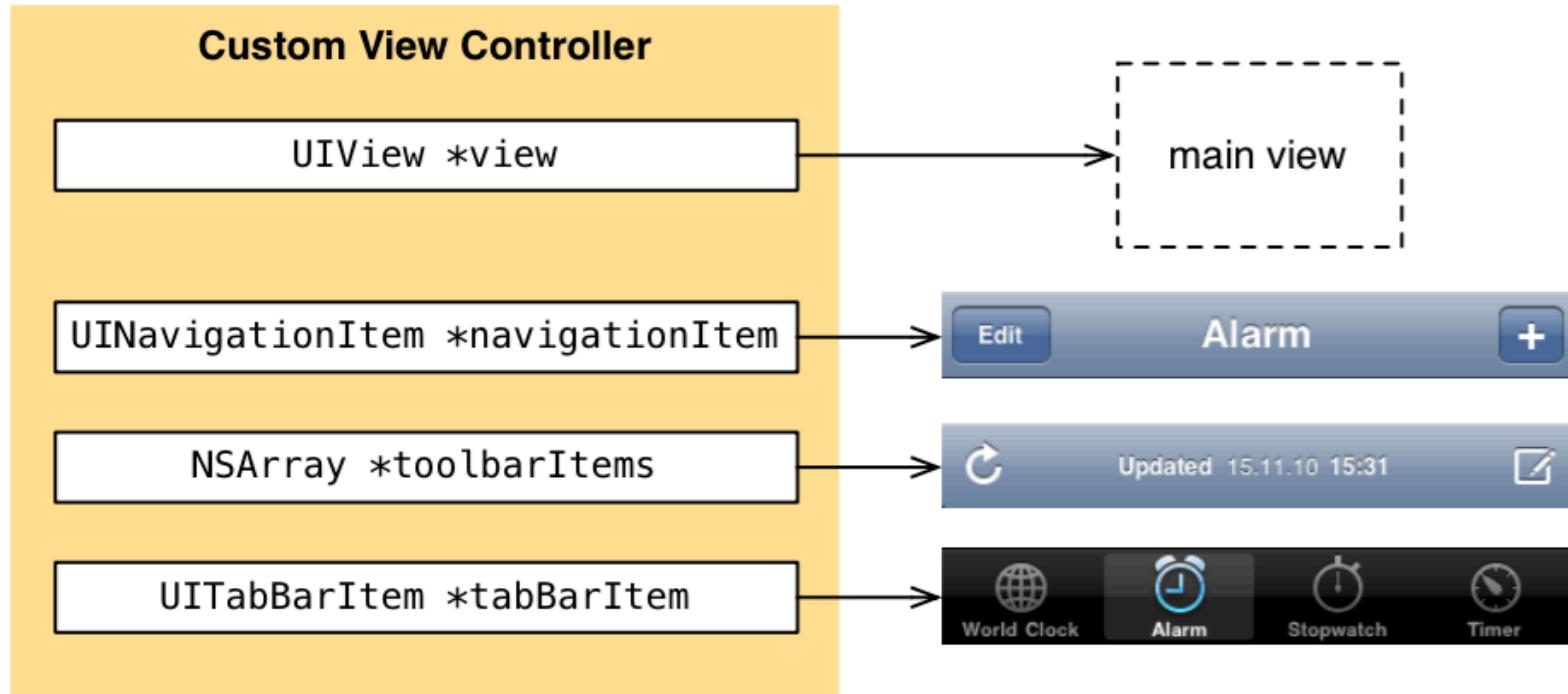
# VIEW MANAGEMENT CYCLE



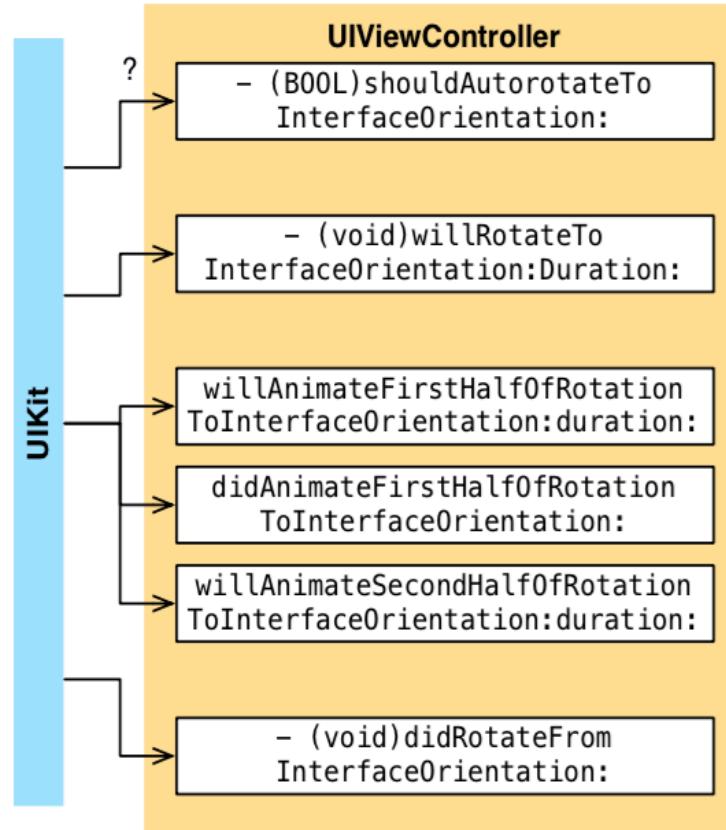
# VIEW CONTROLLER LIFE CYCLE



# STANDARD INTERFACES

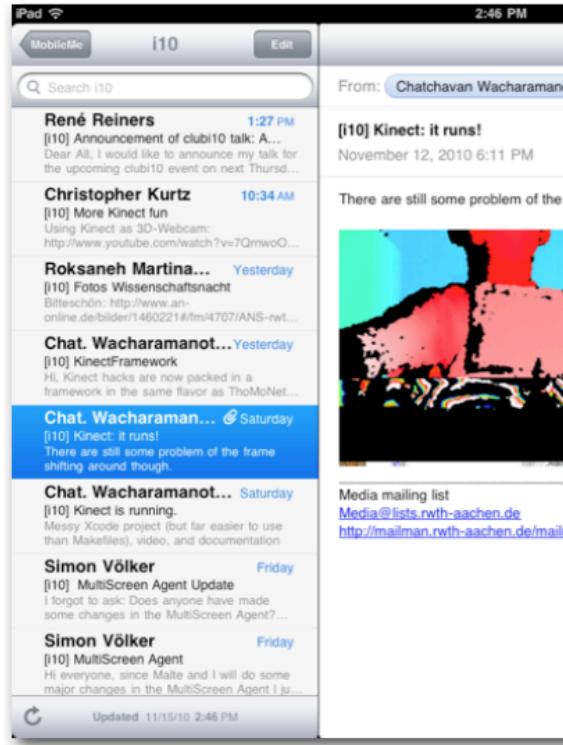


# INTERFACE ORIENTATION



# BUILT-IN VIEW CONTROLLERS

- **UINavigationController**
- **UITabBarController**
- **UISplitViewController**
- **UITableViewController**
- **Framework-specific**



# ACTIONS AND OUTLETS

- User Interfaces in iOS are defined in Nib files
  - A nib file stores an archived collection of objects
  - Visual UI editor integrated into Xcode
- Actions: methods that can be triggered from UI events
- Outlets: instance variables in the controller that allow accessing view elements

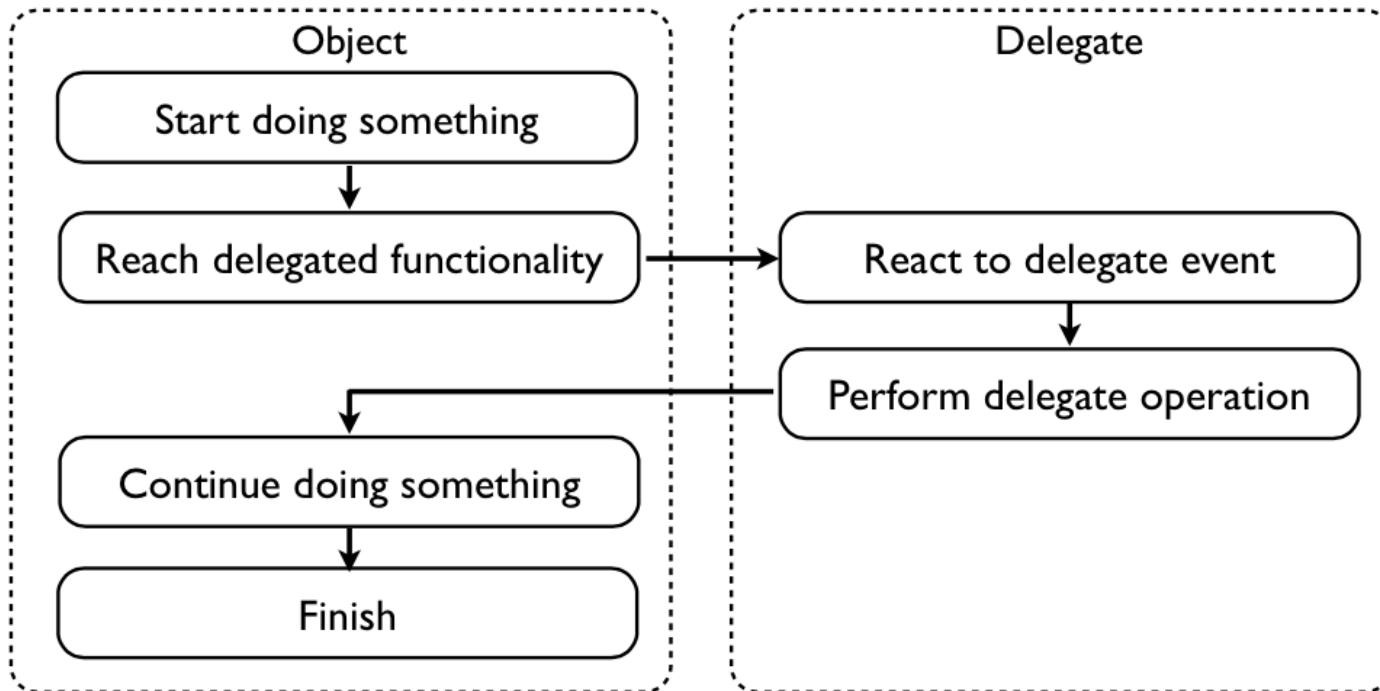
# “DRAWING” CONNECTIONS

- **Actions**
  - Declare a method as IBAction
    - - (IBAction)doSomething:(id)sender
  - Right-click drag from the UI widget to the controller
- **Outlets**
  - Declare an instance variable or property as IBOulet
    - @property(assign) IBOutlet UIView \*view
  - Right-click drag from the controller to the UI widget

# DELEGATION

- Alternative to Subclassing
- Delegate Behavior to Another Class

# DELEGATE



# IMPLEMENTING A DELEGATE

- Conform to the Delegate Protocol
  - Implement custom behavior in delegate methods
- Assign Delegate Object
  - Usually via property
  - The delegate object is not retained (retain-cycle)

# DELEGATES IN THE IOS SDK

<b>Object</b>	<b>Delegate</b>
UIApplication	UIApplicationDelegate
UITableView	UITableViewDelegate
UITextField	UITextFieldDelegate
UIPickerController	UIPickerControllerDelegate
CMMotionManager	CMMotionManagerDelegate

and many more...

# COMMAND PATTERN

- Pattern encapsulates a request as an object.
- The request object binds together one or more actions on a specific receiver.
- The Command pattern separates an object making a request from the objects that receive and execute that request.
- Action-Target mechanism
  - The target-action mechanism enables a control object—that is, an object such as a button, slider, or text field—to send a message to another object that can interpret the message and handle it as an application-specific instruction. The receiving object, or the target, is usually a custom controller object. The message—named an action message—is determined by a selector, a unique runtime identifier of a method.

# OBSERVER DESIGN PATTERN

- The observer pattern is used when one object wants to know when another object changes
- This pattern is build into every NSObject via Key-Value Observing
- This is also often used with MVCs because when a model changes you often will want to update the views
- Encourages weak coupling between objects
- Cocoa examples are NSNotification, KVO, etc.
- NSNotificationCenter provides loose coupling by allowing one object to register to be notified of events defined by string names

# OBSERVER

- The observer pattern is similar to the delegate pattern, however one key difference is that observable objects support multiple observers, while a delegate is just one object
- However, with this expanded possibility comes one big pitfall: you must remember to remove an object as an observer when that object is deallocated, otherwise there will be a code leak
- The following slide contains a code sample of what the Observable class looks like

# OBSERVER

```
@interface Observable: NSObject
- (void)addObserver:
    (id<NSObject>)observer;
- (void)removeObserver:
    (id<NSObject>)observer;
- (void)notifyObservers:
    (NSInvocation*)invocation;
@end
```

# SINGLETON PATTERN

- In many ways it is just a global variable. Provides a global way to access an object.
- The singleton pattern is very simple but extremely powerful
- It is a very common pattern, but developers have to be careful not to overuse it
- Because abuse of the singleton pattern is common, some developers avoid the pattern altogether
- Common in Cocoa. Most cases, by a class method that begins with shared as +sharedApplication
- When a class is restricted to just one instantiation, that one object is called a singleton
- In some situations it can be problematic to have two instances of a class running, this should be the only reason to use the singleton pattern

# SINGLETON PATTERN

```
+ (ExClass *) singleton{  
    static ExClass *sharedInstance = nil;  
    if ( sharedInstance == nil)  
    {  
        sharedInstance = [[ExClass alloc] init];  
    }  
    return sharedInstance;  
}
```

UIKIT VIEWS

# GENERAL CATEGORIES

Category	Purpose	Examples
	Display a particular type of content, such as an image or text.	Image view, label
Content		
	Display collections or groups of views.	Collection view, table view
Collections		
	Perform actions or display information.	Button, slider, switch
Controls		
	Navigate, or perform actions.	Toolbar, navigation bar, tab bar
Bars		
	Receive user input text.	Search bar, text view
Input		
	Serve as containers for other views.	View, scroll view
Containers		
Modal	Interrupt the regular flow of the app to allow a user to perform some kind of action.	Action sheet, alert view

# CUSTOMIZING UILABEL

- **shadowColor**
  - This property is of type UIColor and, as its name shows, it specifies the color of the drop shadow to render for your label. If you are setting this property, you should also set the shadowOffset property.”
- **shadowOffset**
  - This property is of type CGSize, and it specifies the offset of the drop shadow from the text. For instance, if you set this property to (1, 0), the drop shadow will appear 1 point to the right of the text. If you set this property to (1, 2), the drop shadow will appear 1 point to the right and 2 points down from the text. If you set this property to (-2, -10), the drop shadow will render 2 points to the left and 10 points above the text.
- **numberOfLines**
  - This property is an integer that specifies how many lines of text the label is able to render. By default, this property’s value is set to 1, meaning any label that you create by default can handle 1 line of text. If you want 2 lines of text, for instance, set this property to 2. If you want unlimited lines of text to be rendered in your text field or you simply don’t know how many lines of text you will end up displaying, set this property to 0. (I know, it’s really strange. Instead of NSIntegerMax or something similar, Apple has decided that 0 means unlimited!)

# CUSTOMIZING UILABEL

- **lineBreakMode**
  - This property is of type NSLineBreakMode and specifies how you want to line-wrap the text inside your text field. For instance, if you set this property to NSLineBreakByWordWrapping, words will be kept together, but the string will be wrapped to the next line if there is not enough space to display it. Alternatively, if you set this property to NSLineBreakByCharWrapping, words may be broken across lines when text is wrapped. You would probably use NSLineBreakByCharWrapping only if the space is very tight and you need to fit as much information as possible on the screen
- **textAlignment**
  - This property is of type NSTextAlignment and sets the horizontal alignment of the text in your label. For instance, you can set the value of this property to NSTextAlignmentCenter to horizontally center-align your text.
- **textColor**
  - This property is of type UIColor and defines the color of the text inside the label

# CUSTOMIZING UILABEL

- **font**
  - This property of type UIFont specifies the font with which the text inside your label will get rendered.
- **adjustsFontSizeToFitWidth**

This property is of type BOOL. When set to YES, it will change the size of the font to fit your label. For instance, if you have a small label and the text you are trying to set in it is too big to fit, if this property is set to YES, the runtime will automatically reduce the font size of your label to make sure the text will fit into the label. In contrast, if this property is set to NO, the current line/word/character wrapping option is taken into account and your text will be rendered in an incomplete manner with just a few words being displayed..

```
self.label.text = @“iOS SDK”;
self.label.font = [UIFont boldSystemFontOfSize:70.0f];
self.label.textColor = [UIColor blackColor];
self.label.shadowColor = [UIColor lightGrayColor];
self.label.shadowOffset = CGSizeMake(2.0f, 2.0f);
[self.label sizeToFit];
self.label.center = self.view.center;
[self.view addSubview:self.label];”
```

# UITEXTFIELD

- **borderStyle**
  - This property is of type UITextBorderStyle and specifies how the text field should render its borders.
- **contentVerticalAlignment**
  - This value is of type UIControlContentVerticalAlignment and tells the text field how the text should appear, vertically, in the boundaries of the control. If we didn't center the text vertically, it would appear on the top-left corner of the text field by default.
- **textAlignment**
  - This property is of type NSTextAlignment and specifies the horizontal alignment of the text in a text field.
- **text**
  - This is a read/write property: you can both read from it and write to it. Reading from it will return the text field's current text, and writing to it will set the text field's text to the value that you specify.

# UITEXTFIELD DELEGATE

- **textFieldShouldBeginEditing:**
  - A method that returns a BOOL telling the text field (the parameter to this method) whether it should start getting edited by the user or not. Return NO if you don't want the user to edit your text field. This method gets fired as soon as the user taps on the text field with the goal of editing its content (assuming the text field allows editing).
- **textFieldDidBeginEditing:**
  - Gets called when the text field starts to get edited by the user. This method gets called when the user has already tapped on the text field and the textFieldShouldBeginEditing: delegate method of the text field returned YES, telling the text field it is OK for the user to edit the content of the text field.
- **textFieldShouldEndEditing:**
  - Returns a BOOL telling the text field whether it should end its current editing session or not. This method gets called when the user is about to leave the text field or the first responder is switching to another data entry field. If you return NO from this method, the user will not be able to switch to another text entry field, and the keyboard will stay on the screen

# UITEXTFIELD DELEGATE

- **textFieldDidEndEditing:**
  - “Gets called when the editing session of the text field ends. This happens when the user decides to edit some other data entry field or uses a button provided by the supplier of the app to dismiss the keyboard shown for the text field.
- **textField:shouldChangeCharactersInRange:replacementString:**
  - Gets called whenever the text inside the text field is modified. The return value of this method is a Boolean. If you return YES, you say that you allow the text to be changed. If you return NO, the change in the text of the text field will not be confirmed and will not happen.
- **textFieldShouldClear:**
  - Each text field has a clear button that is usually a circular X button. When the user presses this button, the contents of the text field will automatically get erased. We need to manually enable the clear button, though. If you have enabled the clear button and you return NO to this method, that gives the user the impression that your app isn’t working, so make sure you know what you are doing. It is a very poor user experience if the user sees a clear button and presses “but doesn’t see the text in the text field get erased.”
- **textFieldShouldReturn:**
  - Gets called when the user has pressed the Return/Enter key on the keyboard, trying to dismiss the keyboard. You should assign the text field as the first responder in this method.

# UITEXTFIELD DELEGATE PROTOCOL

```
@protocol UITextFieldDelegate
@optional
- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField;
- (void)textFieldDidBeginEditing:(UITextField *)textField;
- (BOOL)textFieldShouldEndEditing:(UITextField *)textField;
- (void)textFieldDidEndEditing:(UITextField *)textField;
- (BOOL)textField:(UITextField *)textField shouldChangeCharactersInRange:
    (NSRange)range replacementString:(NSString *)string;
- (BOOL)textFieldShouldClear:(UITextField *)textField;
- (BOOL)textFieldShouldReturn:(UITextField *)textField;

@end
```

# UITEXTFIELD CLASS

```
@interface UITextField : UIControl <UITextInputTraits, NSCoding> {  
    id<UITextFieldDelegate> _delegate;  
    // Stuff...  
}
```

# IMPLEMENTING IN ANY VIEWCONTROLLER CLASS

```
@interface ViewController () <UITextFieldDelegate>  
@end
```

Mark the class as implementing  
the UITextFieldDelegate.

```
- (BOOL)textFieldShouldReturn:(UITextField *)textField {  
    int i = [self.textFields indexOfObject:textField];  
    if (i < [self.textFields count] - 1) {  
        UITextField *nextTextField = self.textFields[i+1];  
        [nextTextField becomeFirstResponder];  
    } else {  
        [textField resignFirstResponder];  
    }  
    return TRUE;  
}
```

Implement any methods you care about. Since  
you set the view controller as the text field's  
delegate in the storyboard, the view controller  
will receive these messages.

# UITEXTVIEW

- The UITextView class can display multiple lines of text and contain scrollable content, meaning that if the contents run off the boundaries of the text view, the text view's internal components allow the user to scroll the text up and down to see different parts of the text.

# UIALERTVIEW

```
UIAlertView *alertView = [[UIAlertView alloc]
    initWithTitle:@"Alert"
    message:@"You've been
delivered an alert"
    delegate:nil
    cancelButtonTitle:@"Cancel"
    otherButtonTitles:@"Ok", nil];
[alertView show];
```

# UIALERTVIEW STYLES

- **UIAlertViewStyleDefault**
  - This is the default style of an alert view
- **UIAlertViewStyleSecureTextInput**
  - With this style, the alert view will contain a secure text field, which hides the actual characters typed by the user. For instance, if you are asking the user for her online banking credentials, you might choose this style of alert view.
- **UIAlertViewStylePlainTextInput**
  - Under this style, the alert view will display a nonsecure text field to the user. This style is great if you simply want to ask the user for a plain-text entry, such as her phone number.
- **UIAlertViewStyleLoginAndPasswordInput**
  - With this style, the alert view will display two text fields: a nonsecure one for a username and a secure one for a password.

# CUSTOMIZING UISWITCH

There are two main ways of customizing a switch:

- **Tint Colors**
  - Tint colors are colors that you can apply to a UI component such as a UISwitch. The tint color will be applied on top of the current color of the component. For instance, in a normal UISwitch, you will be able to see different colors. When you apply the tint color on top, the normal color of the control will be mixed with the tint color, giving a flavor of the tint color on the UI control.
- **Images**
  - IOS 7+ ignores this and uses tintcolor

**On Image**

  - The image that represents the on state of the switch. The width of this image is 77 points, and its height is 22.

**Off Image**

  - The image that represents the switch in its off state. This image, like the on state of the switch, is 77 points in width and 22 points in height

# UIPICKERVIEW

The data source of an instance of UIPickerView must conform to the UIPickerViewDataSource protocol.

- **numberOfComponentsInPickerView:**
  - This method passes you a picker view object as its parameter and expects you to return an integer, telling the runtime how many components you would like that picker view to render.
- **pickerView:numberOfRowsInComponent:**
  - For each component that gets added to a picker view, you will need to tell the system how many rows you would like to render in that component. This method passes you an instance of picker view, and you will need to return an integer indicating the number of rows to render for that component.

# UIPICKERVIEW

The delegate of an instance of UIPickerView has to conform to the UIPickerViewDelegate protocol and must implement all the @required methods of that protocol.

- **pickerView:titleForRow:forComponent: method.**
  - This method will pass you the index of the current section and the index of the current row in that section for a picker view, and it expects you to return an instance of NSString. This string will then get rendered for that specific row inside the component.

# UIDATEPICKER

- UIDatePicker is very similar to the UIPickerView class. The date picker is in fact a prepopulated picker view.

```
typedef NS_ENUM(NSInteger, UIDatePickerMode) {  
    UIDatePickerModeTime,  
    UIDatePickerModeDate,  
    UIDatePickerModeDateAndTime,  
    UIDatePickerModeCountDownTimer,  
};
```

# UISLIDER

- **minimumValue**
  - Specifies the minimum value of the slider's range.
- **maximumValue**
  - Specifies the maximum value of the slider's range.
- **value**
  - The current value of the slider. This is a read/write property, meaning that you can both read from it and write to it. If you want the slider's knob to be moved to this value in an animated mode, you can call the `setValue:animated:` method of the slider and pass YES as the animated parameter.

# CUSTOMIZING UISLIDER

- **minimumTrackTintColor**
  - This property sets the tint color of the minimum value track view.
- **thumbTintColor**
  - This property, as its name shows, sets the tint color of the thumb view.
- **maximumTrackTintColor**
  - This property sets the tint color of the maximum value track view.
- **All these properties are of type UIColor.**

# UISEGMENTEDCONTROL

```
NSArray *segments = @[
    @"iPhone",
    @"iPad",
    @"iPod",
    @"iMac"
];
self.mySegmentedControl = [[UISegmentedControl alloc]
initWithItems:segments];
self.mySegmentedControl.center = self.view.center;
[self.view addSubview:self.mySegmentedControl];
[self.mySegmentedControl addTarget:self
    action:@selector(segmentChanged:)
    forControlEvents:UIControlEventValueChanged];
}
```

# UIIMAGEVIEW

- **UIViewContentMode** enumeration:
- **UIViewContentModeScaleToFill**
  - This will scale the image inside the image view to fill the entire boundaries of the image view.
- **UIViewContentModeScaleAspectFit**
  - This will make sure the image inside the image view will have the right aspect ratio and fits inside the image view's boundaries.
- **UIViewContentModeScaleAspectFill**
  - This will make sure the image inside the image view will have the right aspect ratio and fills the entire boundaries of the image view. For this value to work properly, make sure that you have set the clipsToBounds property of the image view to YES.

# UIWEBVIEW

```
self.myWebView = [[UIWebView alloc]
initWithFrame:self.view.bounds];
    self.myWebView.scalesPageToFit = YES;
    [self.view addSubview:self.myWebView];

    NSURL *url = [NSURL URLWithString:@"http://
www.apple.com"];
    NSURLRequest *request = [NSURLRequest
requestWithURL:url];
    [self.myWebView loadRequest:request];
```

# UIWEBVIEWDELEGATE

```
- (void)webViewDidStartLoad:(UIWebView *)webView{
    [[UIApplication sharedApplication]
    setNetworkActivityIndicatorVisible:YES];
}

- (void)webViewDidFinishLoad:(UIWebView *)webView{
    [[UIApplication sharedApplication]
    setNetworkActivityIndicatorVisible:NO];
}

- (void)webView:(UIWebView *)webView didFailLoadWithError:
(NSError *)error{
    [[UIApplication sharedApplication]
    setNetworkActivityIndicatorVisible:NO];
}
```

# UIPROGRESSVIEW

```
self.progressView = [[UIProgressView alloc]
    initWithFrame:[UIProgressViewStyle:UIProgressViewStyleBar];
self.progressView.center = self.view.center;
self.progressView.progress = 20.0f / 30.0f;
[self.view addSubview:self.progressView];
```

- **UIProgressViewStyle**
- **UIProgressViewStyleDefault**
  - This is the default style of the progress view.
- **UIProgressViewStyleBar**
  - This is similar to the UIProgressViewStyleDefault but is meant to be used for progress views that are to be added to a toolbar..

# ACTION SHEETS

- Action sheets display a set of buttons representing several alternative choices to complete a task initiated by the user. For example, when the user taps the Share button in an app's toolbar, an action sheet appears offering a list of choices, such as Email, Print, and so on.
- **Purpose.** Action sheets allow users to:
  - Quickly select from a list of actions
  - Confirm or cancel an action

# ACTION SHEETS

- `UIActionSheet *actionSheet =  
[[UIActionSheet alloc]  
 initWithTitle:nil delegate:self  
 cancelButtonTitle:@"Cancel"  
 destructiveButtonTitle:@"Delete  
 Note" otherButtonTitles:nil];`

# UIACTIONSHEETDELEGATE

```
- (void)actionSheet:(UIActionSheet
*)actionSheet clickedButtonAtIndex:
(NSUInteger)buttonIndex
{
    NSLog(@"The %@ button was tapped.",
[actionSheet
buttonTitleAtIndex:buttonIndex]);
}
```

# MORE VIEWS

- Activity Indicator
- UIStepper
- UITableView
- UICollectionView
- UIScrollView

# VIEW CONTROLLERS

# EXAMPLE

Recipes +

	<b>Chocolate Cake</b>	1 hour
	Chocolate cake with chocolate frosting	>
	<b>Crêpes Suzette</b>	20min
	Crêpes flambeées with grand marnier.	>
	<b>Gaufres de Liège</b>	1 hour
	Belgian-style waffles	>
	<b>Ginger snaps</b>	45 minutes
	Nana's secret recipe	>
	<b>Macarons</b>	1 hour
	Macarons français: chocolat, pistache, fram...	>
	<b>Tarte aux Fraises</b>	25 min
	Delicious tart	>
	<b>Three Berry Cobbler</b>	1.5 hours
	Raspberry, blackberry, and blueberry cobbler	>

Recipes      Unit Conversion

List controller

Recipes      Chocolate Cake      Edit

Chocolate Cake  
Chocolate cake with chocolate...

Preparation time: 1 hour

Category

Dessert

Ingredients

**Salt**  
3/4 teaspoon

**Milk**  
5 tablespoons

**Water**  
5 tablespoons

Recipes      Unit Conversion

Detail controller

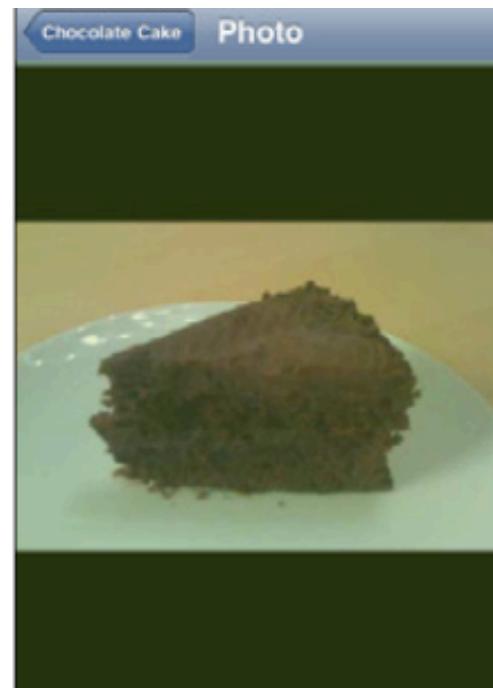
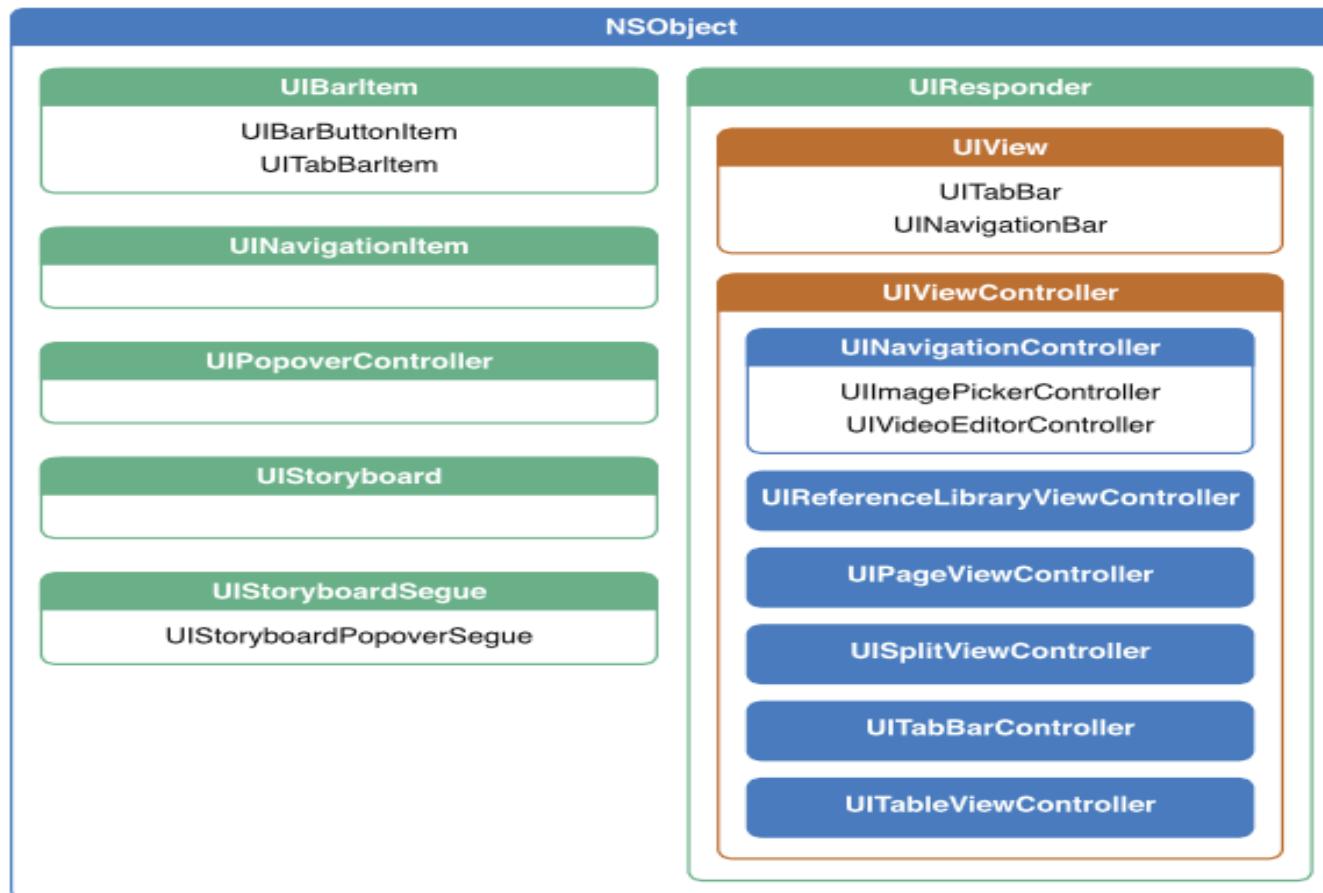


Photo controller

# UIKIT VIEW CONTROLLER CLASSES



# VIEW CONTROLLERS LIFECYCLE

- A sequence of messages is sent to them as they progress through it.
- At each stage, iOS invokes method(s) on the Controller .
  - Loading view hierarchy
  - Appearing and disappearing.
  - Geometry changes
  - Low-memory situations.
- You very commonly override these methods to do certain work.

# LIFECYCLE METHODS

## Methods in UIViewController

viewDidLoad

Called first time  
viewController's view property  
accessed.

viewWillAppear

Often a good place to populate  
views from model.

viewDidAppear

viewWillDisappear

viewDidDisappear

didReceiveMemoryWarning

Happens when device is low on  
memory.

# VIEWDIDLOAD

- After instantiation and outlet-setting, `viewDidLoad` is called. This is an exceptionally good place to put a lot of setup code.
  - ```
(void)viewDidLoad {
    [super viewDidLoad]; // always let super have a chance in lifecycle
    methods
    // do some setup of my MVC
}
```
- 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 an iPhone sized screen or an iPad or ??. So do not initialize things that are geometry-dependent here.

# VIEW{WILL,DID}APPEAR

- Just before the view appears on screen, you get notified (argument is just whether you are appearing instantly or over time via animation)
  - `(void)viewWillAppear:(BOOL)animated;`
- Your view will only get “loaded” once, but it might appear and disappear a lot. So don’t put something in this method that really wants to be in `viewDidLoad`. Otherwise, you might be doing something over and over unnecessarily.
- Do something here if things you display are changing while your MVC is off-screen. Later we will use this to optimize performance by waiting until this method (as opposed to `viewDidLoad`) to kick off an expensive operation (probably in another thread). View’s geometry is set here, but there are other (better?) places to react to geometry.

# VIEW{WILL,DID}DISAPPEAR

- And you get notified when you will disappear off screen too This is where you put "remember what's going on" and cleanup code.

```
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated]; // call super in all the viewWill/Did...
    // methods // let's be nice to the user and remember the scroll position they
    // were at ...
    [self rememberScrollPosition]; // we'll have to implement this, of course //
    // do some other clean up now that we've been removed from the screen
    [self saveDataToPermanentStore]; // maybe do in did instead?
    // but be careful not to do anything time-consuming here, or app will be
    // sluggish
    // maybe even kick off a thread to do what needs doing here (again, we'll
    // cover threads later)
}
```

# VIEW{WILL, DID} LAYOUTSUBVIEWS

- Geometry changed?
- Most of the time this will be automatically handled with Autolayout (next week).
  - (void)view{Will,Did}LayoutSubviews;

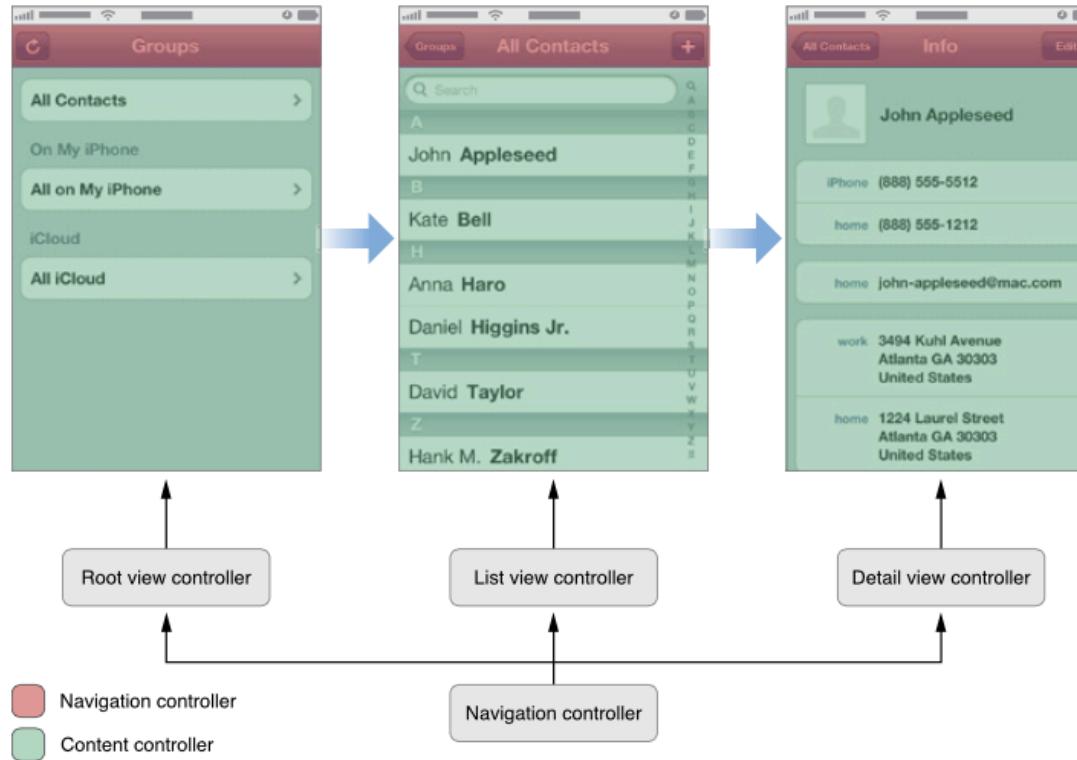
Called any time a view's frame changed and its subviews were thus re-layed out.

For example, autorotation.

You can reset the frames of your subviews here or set other geometry-affecting properties. Between "will" and "did", autolayout will happen

UINavigationController

# UINavigationController



# NAVIGATION CONTROLLER

- Provides easy way to “drill down” to view controllers
- Set Up with root view controllers
- LIFO stack based model
  - Each controller is pushed or popped into or from the navigation stack
    - `pushViewController:` ...
    - `popViewController:` ...

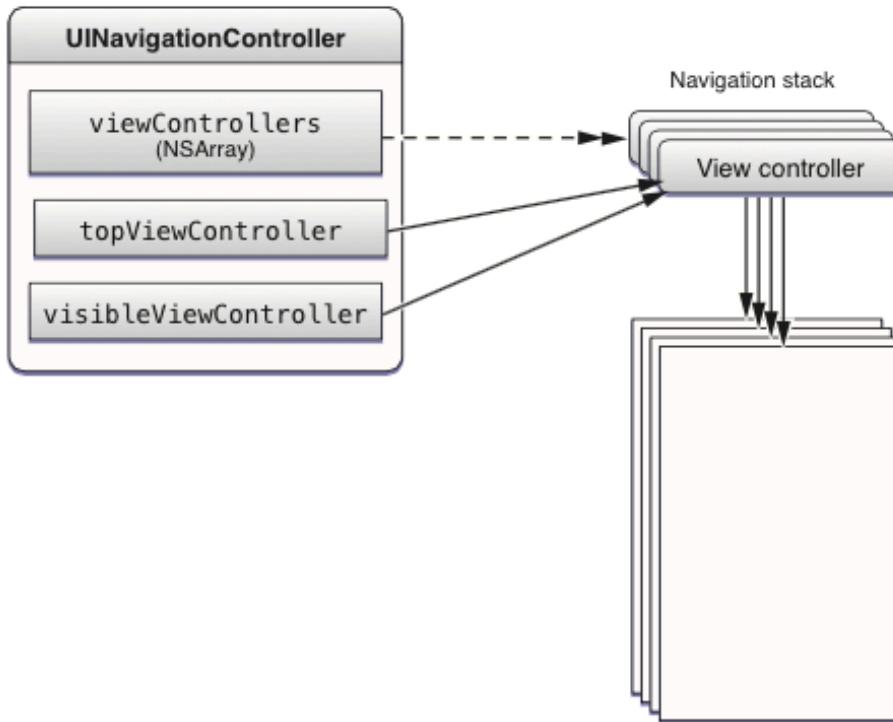
# CREATING PROGRAMMATICALLY

- ```
(void)applicationDidFinishLaunching:(UIApplication
*)application {
    UIViewController *myViewController = [[MyViewController
alloc] init];
    self.navigationController = [[UINavigationController alloc]
initWithRootViewController:myViewController];
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen
mainScreen] bounds]];
    self.window.rootViewController = self.navigationController;
    [self.window makeKeyAndVisible];
}
```

# NAVIGATION CONTROLLER

- **Important properties**
  - self.navigationController
  - self.navigationItem.leftBarButtonItem
  - self.navigationItem.rightBarButtonItem
  - self.navigationItem.title
  - self.title
  - self.navigationBar

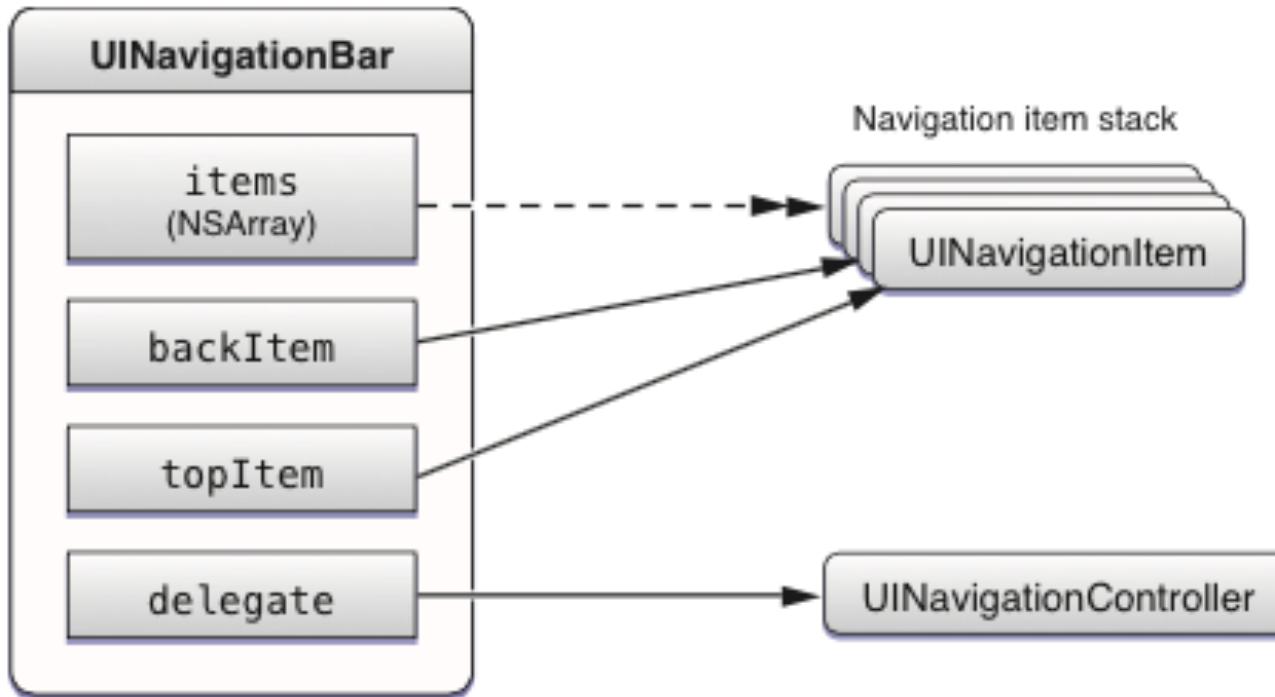
# NAVIGATION STACK



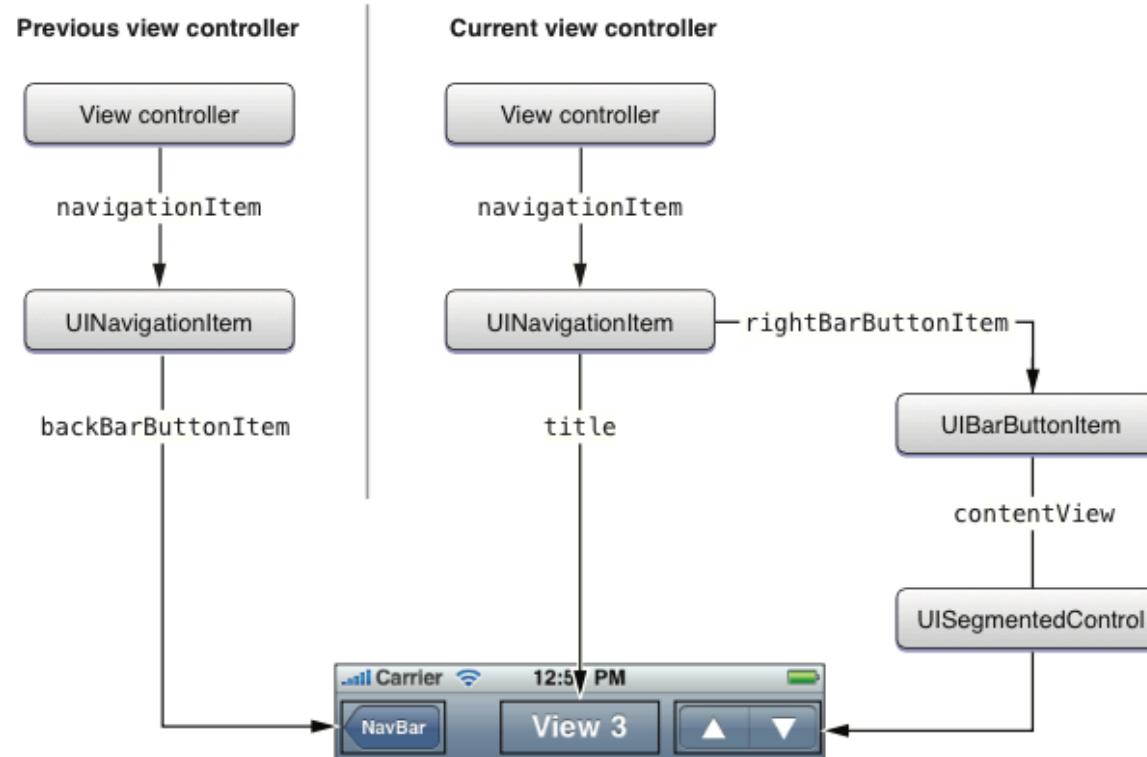
# NAVIGATION STACK

- **pushViewController:animated: method**
  - To push a new view controller onto the navigation stack. The new view controller is responsible for presenting the contents of the selected item.
- **popViewController:animated: method.**
  - Programmatically remove the opmost view controller
- **setViewControllers:animated:**
  - To set a new stack of view controllers
- **popToRootViewControllerAnimated: method.**
  - This method removes all but the root view controller from the navigation stack.
- **popToViewController:animated: method**
  - This method removes everything until the view controller object that has been passed from the navigation stack.

# NAVIGATION BAR



# NAVIGATION BAR STRUCTURE



# NAVIGATION BAR PROPERTIES

- **backBarButtonItem/ leftBarButtonItem**
  - In a navigation interface, the navigation controller assigns a Back button to the left position by default.
  - To assign a custom button or view to the left position, and thereby replace the default Back button, assign a UIBarButtonItem object to the leftBarButtonItem property.
- **titleView**
  - In a navigation interface, the navigation controller displays a custom view with the title of your content view controller by default. You can replace this view as desired with your own custom view.
- **rightBarButtonItem**
  - This position is empty by default. It is typically used to place buttons for editing or modifying the current screen. You can also place custom views here by wrapping the view in a UIBarButtonItem object.

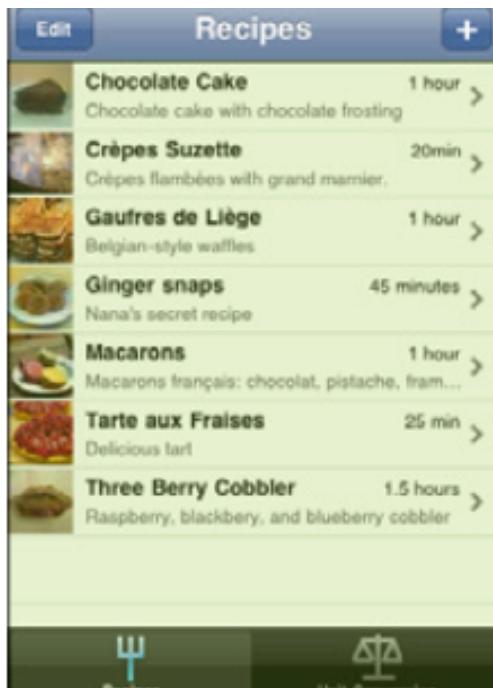
UITABBARCONTROLLE  
R

# TAB BAR CONTROLLER

Recipes +

- Chocolate Cake** 1 hour >  
Chocolate cake with chocolate frosting
- Crêpes Suzette** 20min >  
Crêpes flambeées with grand marnier.
- Gaufres de Liège** 1 hour >  
Belgian-style waffles
- Ginger snaps** 45 minutes >  
Nana's secret recipe
- Macarons** 1 hour >  
Macarons français: chocolat, pistache, fram...
- Tarte aux Fraises** 25 min >  
Delicious tart
- Three Berry Cobbler** 1.5 hours >  
Raspberry, blackberry, and blueberry cobbler

Recipes Unit Conversion



List controller

Recipes Chocolate Cake Edit

Chocolate Cake  
Chocolate cake with chocolate...  
Preparation time: 1 hour

Category  
**Dessert**

Ingredients  
**Salt**  
3/4 teaspoon  
**Milk**  
5 tablespoons  
**Water**  
5 tablespoons

Recipes Unit Conversion



Detail controller

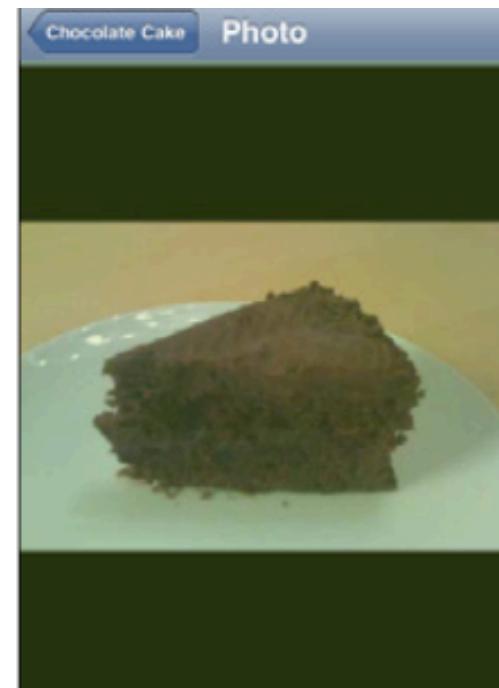
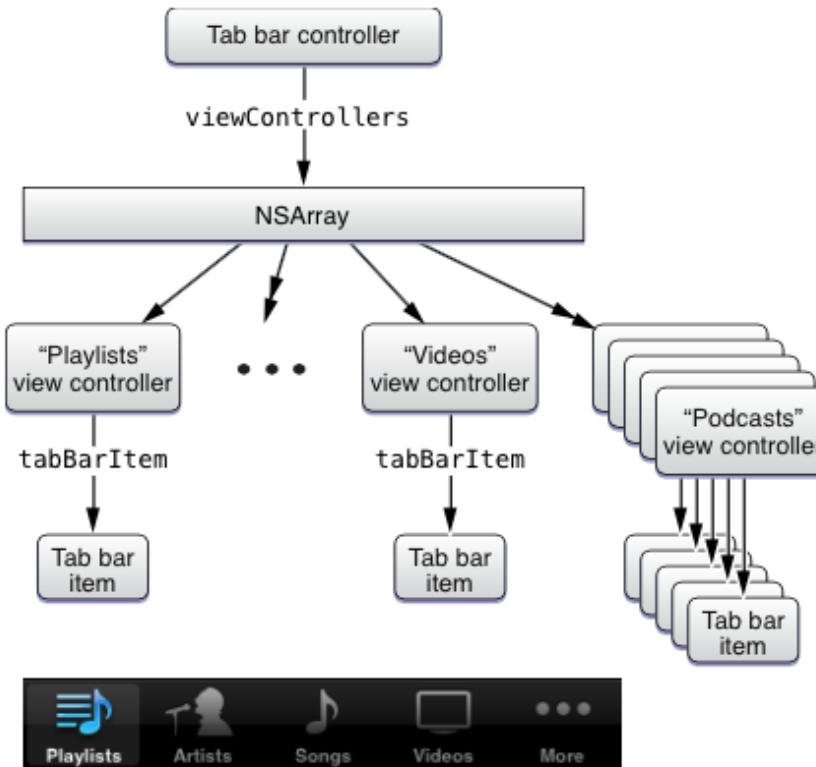


Photo controller

# TABBARCONTROLLER

- Allows access to several peer view controllers
- Best used as initial view controller
- Maximum 5 tabs
- Tabs can have badges

# TAB BAR STRUCTURE



# TAB BAR PROGRAMMATICALLY

- `(void)applicationDidFinishLaunching:(UIApplication *)application`  
`{`  
`tabBarController = [[UITabBarController alloc] init];`  
`MyViewController* vc1 = [[MyViewController alloc] init];`  
`MyOtherViewController* vc2 =`  
`[[MyOtherViewController alloc] init];`  
`NSArray* controllers = [NSArray`  
`arrayWithObjects:vc1, vc2, nil];`  
`tabBarController.viewControllers = controllers;`  
`window.rootViewController = tabBarController;`  
`}`

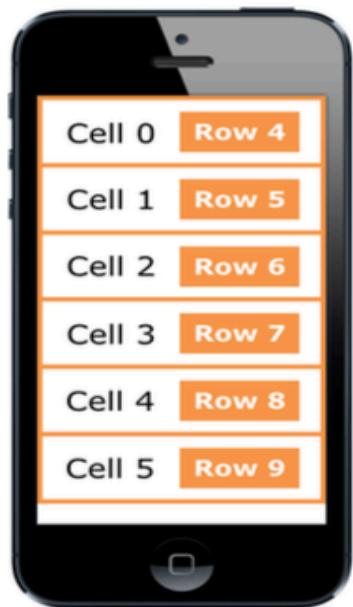
# TAB BAR CONTROLLER

- Important properties
  - self.tabBarController
  - self.tabBarItem.title
  - self.tabBarItem.image
  - self.tabBarItem.badgeValue

# TABLE VIEWS

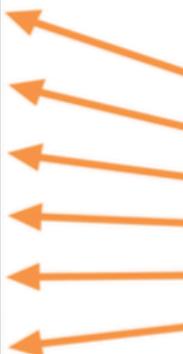
# ROWS AND CELLS

Table view  
with cells



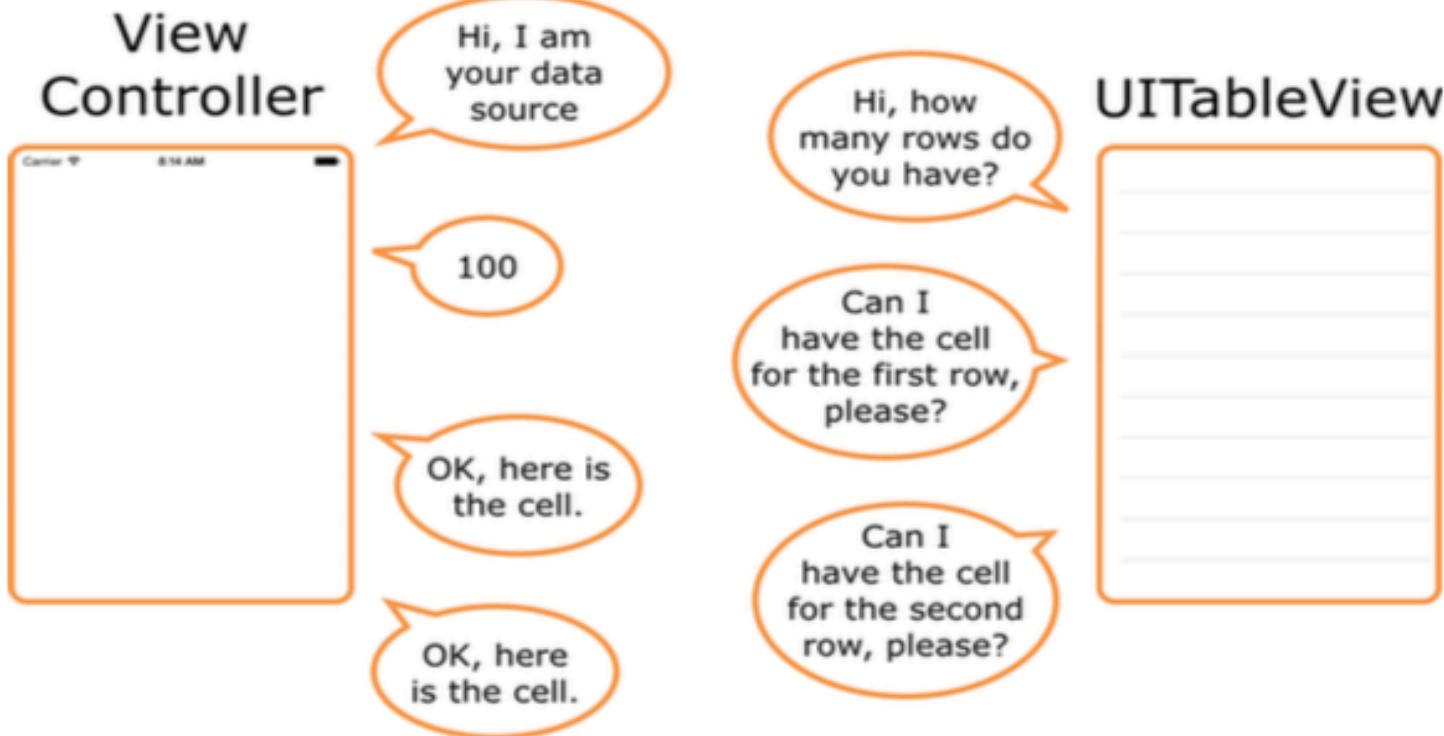
Rows of data

Row 0
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7
Row 8
Row 9
Row 100

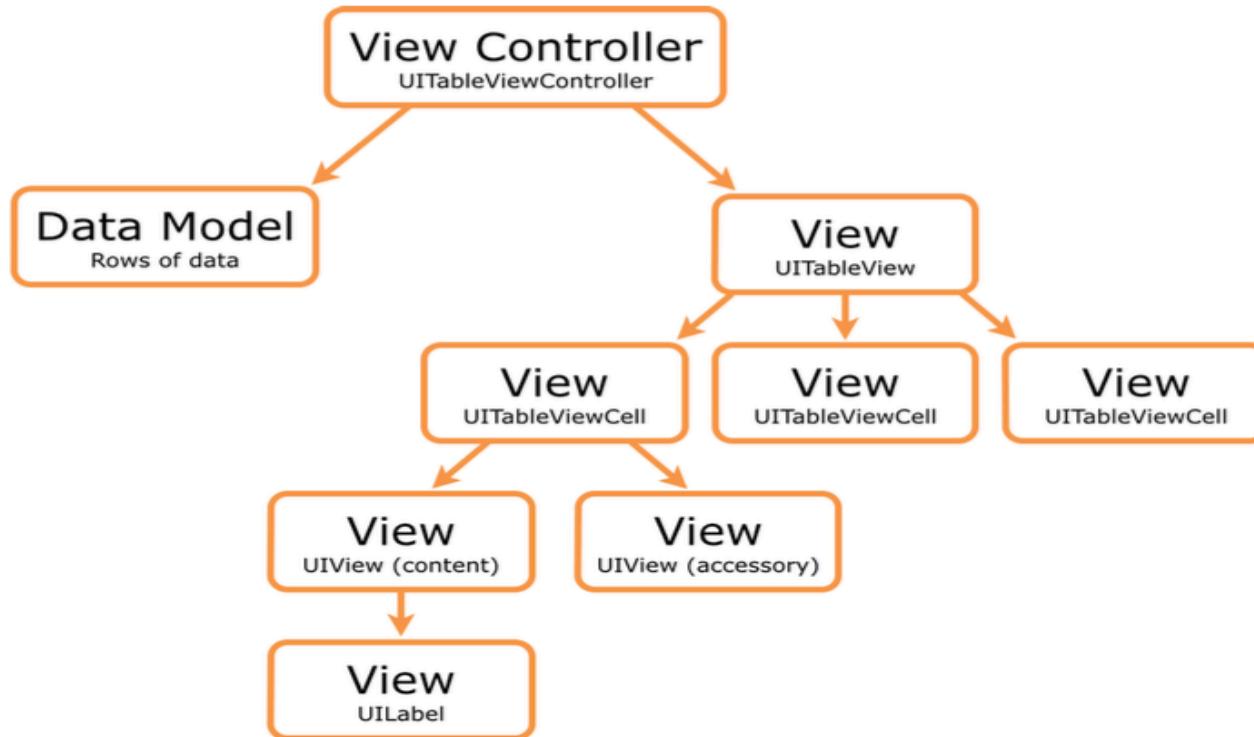


Cells display the contents of rows

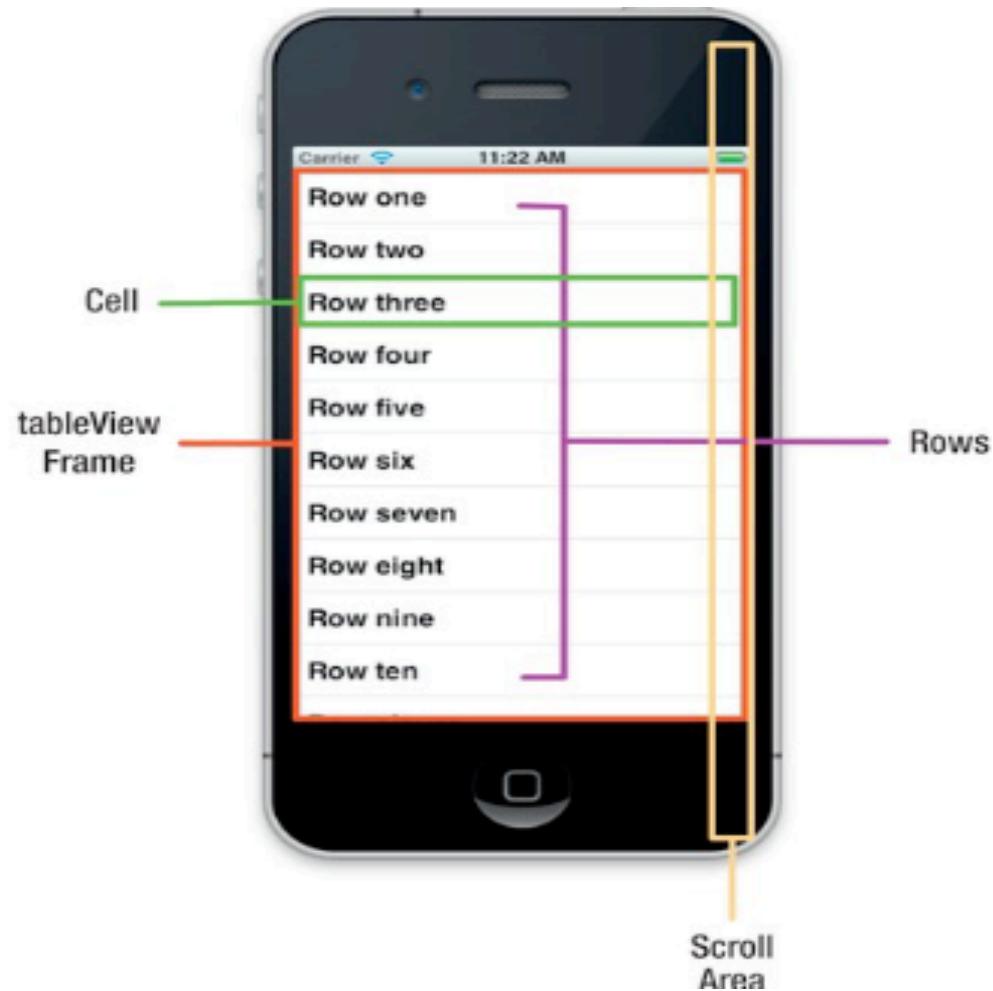
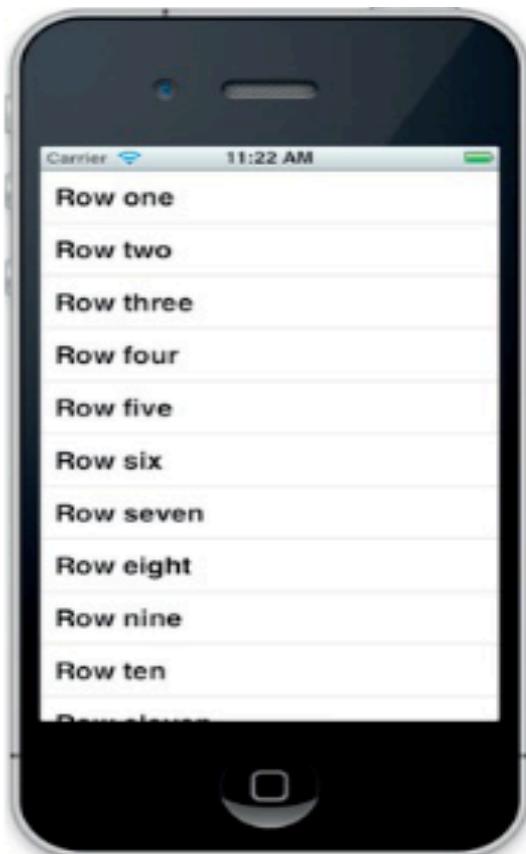
# GETTING CELLS ONE AT A TIME



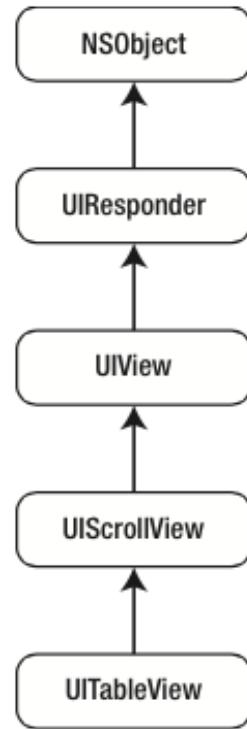
# UITABLEVIEWCONTROLLER



# ABOUT TABLE VIEWS



# CLASS UITABLEVIEW



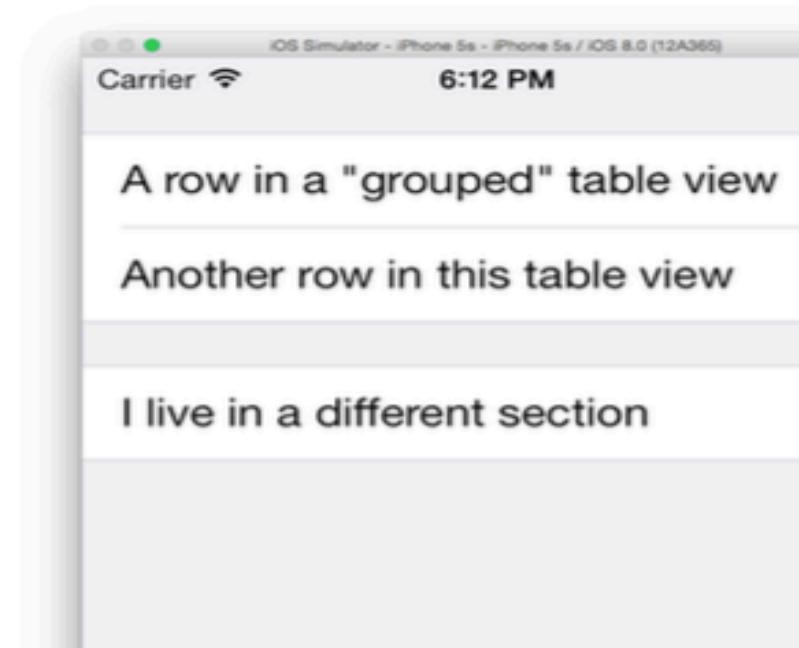
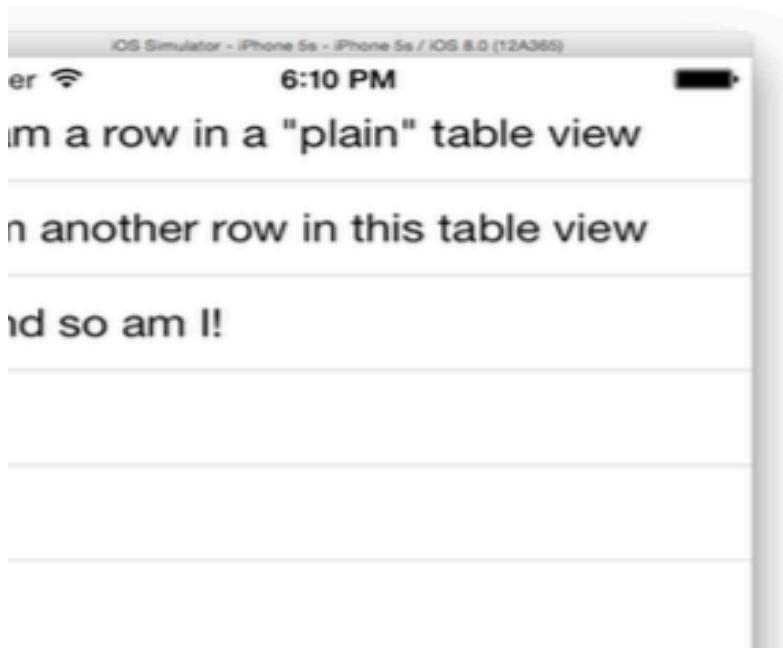
# ABOUT TABLE VIEWS

- UI Component that presents data in scrollable list.

Holds rows that may divided into sections

- Many purposes
  - Navigate data
  - Present data
  - Selectable list of opFons
- Two styles: plain or grouped

# UITABLEVIEW TYPES



A plain-style table (left) and a grouped table (right)

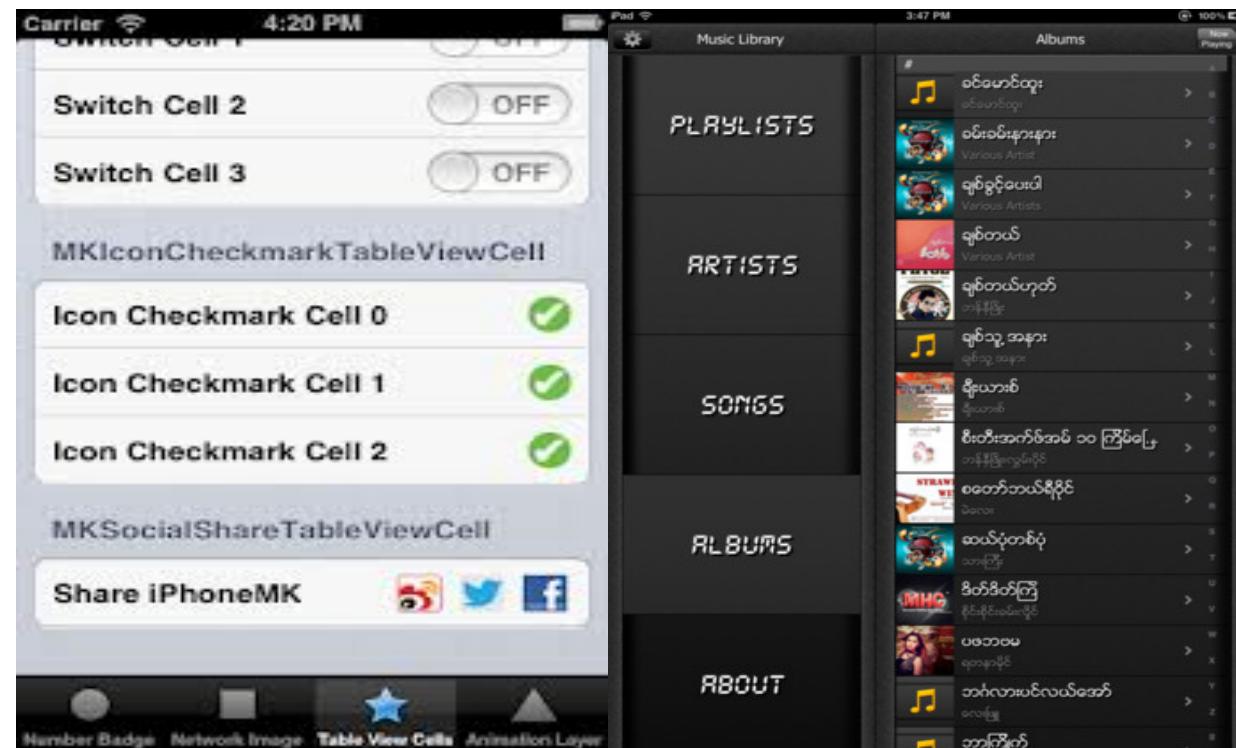
# PLAIN TABLE VIEW

More	Genres
	Alternative >
	Alternative & Punk >
	Classical >
	Electronica/Dance >
	Jazz >
	New Age >
	Pop >
	Soundtrack >
	

Carrier	6:22 PM
	Time
A	A
Abidjan	B C D E F G H I J K L M N O P Q R S T U V W Y Z
Accra	
Adak	
Addis_Ababa	
Adelaide	
Aden	
Algiers	
Almaty	
Amman	

Carrier	4:55 PM
	Pizza Toppings
Extra cheese	✓
Pepperoni	
Black olive	✓
Sausage	✓
Mushroom	✓
Pepper	

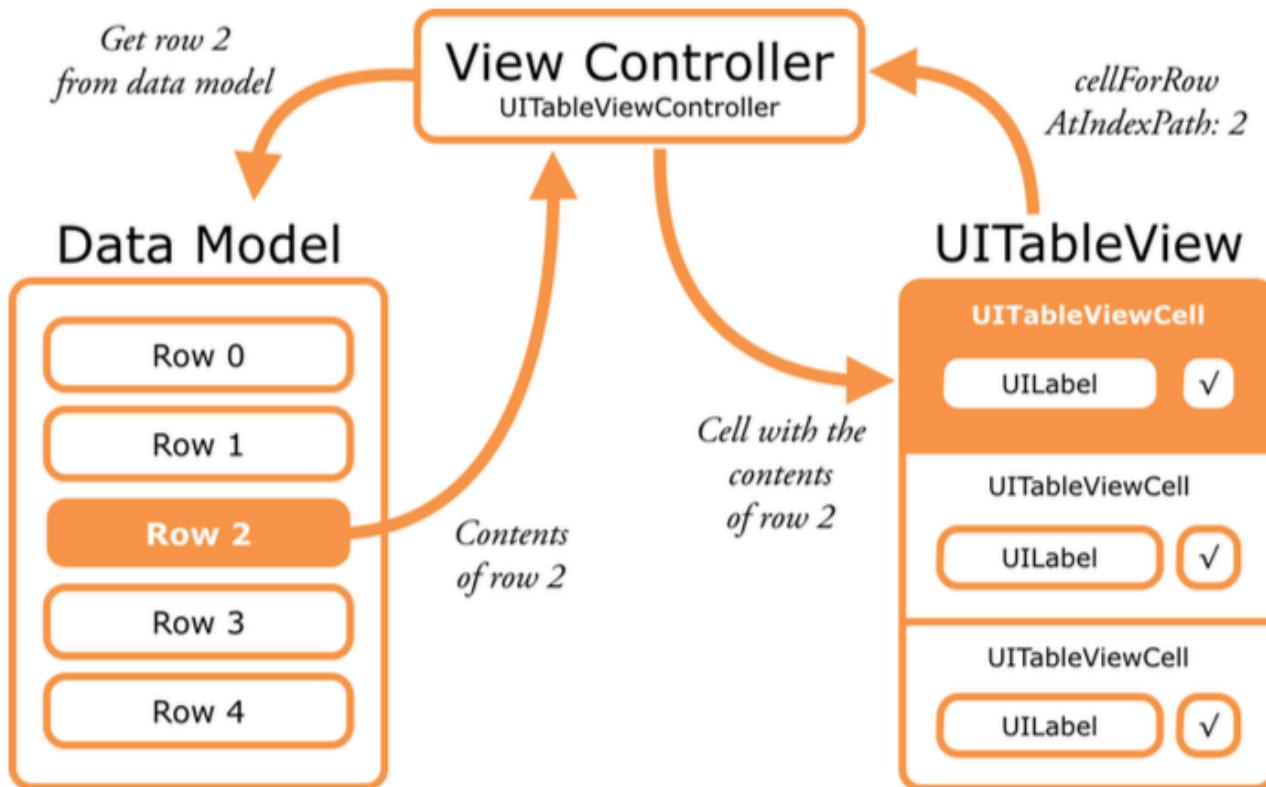
# GROUPED TABLE VIEW



# UITABLEVIEW PROTOCOLS

- **UITableViewDatasource** provides the table view with the data that it needs to construct and configure itself, as well as providing the cells that the table view displays.
- **UITableViewDelegate** handles most of the methods concerned with user interaction, such as selection and editing.

# DATA SOURCE



# UITABLEVIEWDATASOURCE PROTOCOL

- The `UITableViewDataSource` protocol is adopted by an object that mediates the application's data model for a `UITableView` object.
- The data source provides the table-view object with the information it needs to construct and modify a table view.
- As a representative of the data model, the data source supplies minimal information about the table view's appearance
- The table-view object's delegate—an object adopting the `UITableViewDelegate` protocol—provides that information.
- The required methods of the protocol provide the cells to be displayed by the table-view as well as inform the `UITableView` object about the number of sections and the number of rows in each section.
- The data source may implement optional methods to configure various aspects of the table view and to insert, delete, and reorder rows.

# UITABLEVIEWDATASOURCE REQUIRED INSTANCE METHODS

- 1. `tableView:cellForRowIndexPath:`
- This methods asks the data source for a cell to insert it in a particular location in the table view. Syntax for method is given by:
  - `- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath`
- **Parameters:** 1.tableView A table-view object requesting the cell. 2.indexPath An index path locating a row in `tableView`.
- **Return Value:** An object inheriting from `UITableViewCell` that the table view can use for the specified row. An assertion is raised if you return nil.
- **Discussion:** The returned `UITableViewCell` object is frequently one that the application reuses for performance reasons. You should fetch a previously created cell object that is marked for reuse by sending a `dequeueReusableCellWithIdentifier:` message to `tableView`. Various attributes of a table cell are set automatically based on whether the cell is a separator and on information the data source provides, such as for accessory views and editing controls.

# UITABLEVIEWDATASOURCE REQUIRED INSTANCE METHODS

## tableView:numberOfRowsInSection:

Tells the data source to return the number of rows in a given section of a table view. (required)

- (**NSInteger**)tableView:(**UITableView** \*)tableView numberOfRowsInSection:(**NSInteger**)section

### Parameters

*tableView*

The table-view object requesting this information.

*section*

An index number identifying a section in *tableView*.

### Return Value

The number of rows in section.

# UITABLEVIEWDELEGATE PROTOCOL

- The delegate of a UITableView object must adopt the UITableViewDelegate protocol. Optional methods of the protocol allow the delegate to manage selections, configure section headings and footers, help to delete and reorder cells, and perform other actions.

# EXPLAINING THE METHODS

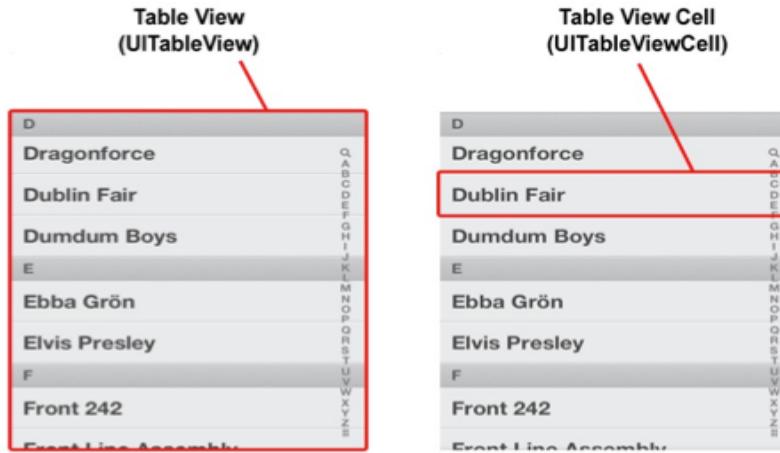
- `- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath`
- This method is called by the table view when it needs to draw one of its rows.
- `static NSString *SimpleTableIdentifier = @"SimpleTableIdentifier"`
- This string will be used as a key to represent the type of our table cell. Our table will use only a single type of cell.
- A table view can display only a few rows at a time on the iPhone's small screen, but the table itself can conceivably hold considerably more. Remember that each row in the table is represented by an instance of UITableViewCell, a subclass of UIView, which means each row can contain subviews. With a large table, this could represent a huge amount of overhead if the table were to try to keep one table view cell instance for every row in the table, regardless of whether that row was currently being displayed.
- Instead, as table view cells scroll off the screen, they are placed into a queue of cells available to be reused. If the system runs low on memory, the table view will get rid of the cells in the queue. But as long as the system has some memory available for those cells, it will hold on to them in case you want to use them again.
-

# EXPLAINING THE METHODS

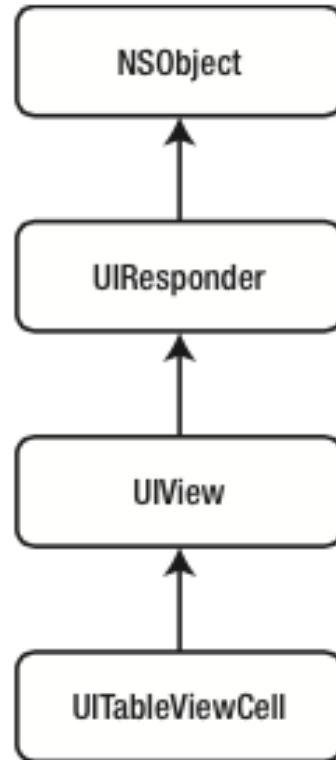
- Every time a table view cell rolls off the screen, there's a pretty good chance that another one just rolled onto the screen on the other side. If that new row can just reuse one of the cells that has already rolled off the screen, the system can avoid the overhead associated with constantly creating and releasing those views. To take advantage of this mechanism, we'll ask the table view to give us a previously used cell of the specified type. Note that we're making use of the NSString identifier we declared earlier. In effect, we're asking for a reusable cell of type SimpleTableIdentifier.
- `UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:SimpleTableIdentifier];`
- Now, it's completely possible that the table view won't have any spare cells (when it's being initially populated, for example), so we check `cell` after the call to see whether it's nil. If it is, we manually create a new table view cell using that identifier string. At some point, we'll inevitably reuse one of the cells we create here, so we need to make sure that we create it using `SimpleTableIdentifier`.
- `if (cell == nil) { cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:SimpleTableIdentifier]; }`

# TABLE VIEWS AND TABLE VIEW CELLS

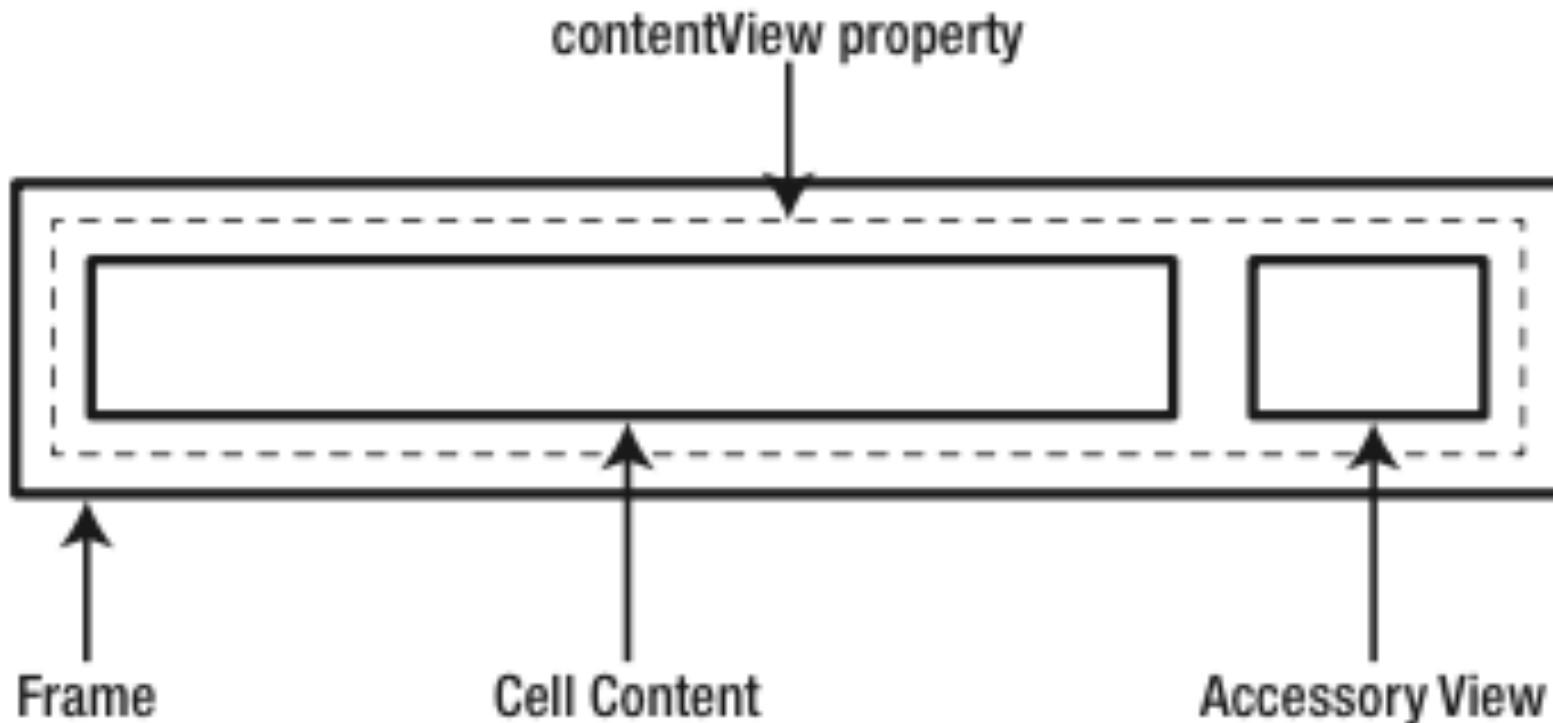
- A table view is the view object that displays a table's data and is an instance of the class UITableView.
- Each visible row of the table is implemented by the class UITableViewCell.
- So, a table view is the object that displays the visible part of a table, and a table view cell is responsible for displaying a single row of the table



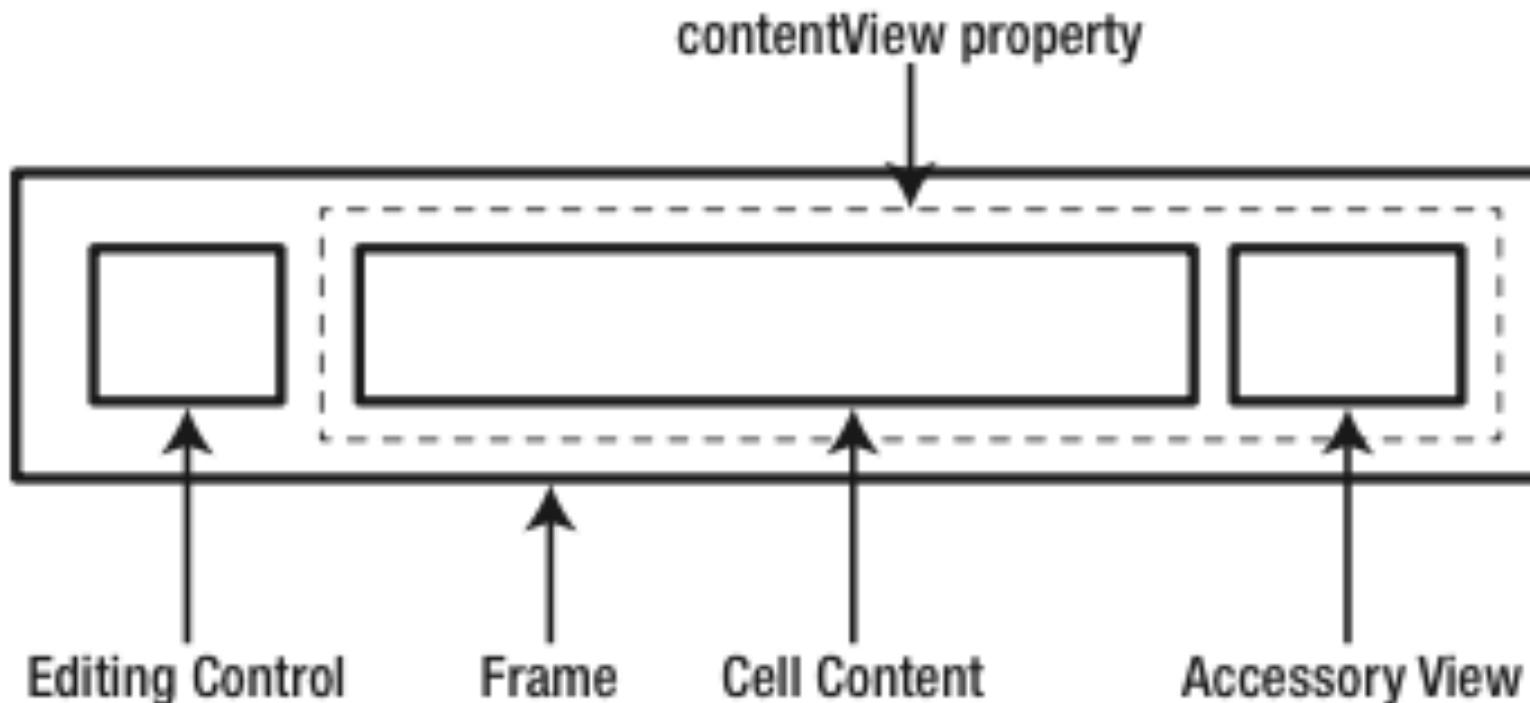
# CLASS UITABLEVIEWCELL



# UITABLEVIEWCELL STRUCTURE



# UITABLEVIEWCELL STRUCTURE IN EDITING



# DISPLAYING DATA IN UITABLEVIEWCELL

- We now have a table view cell that we can return for the table view to use. So, all we need to do is place whatever information we want displayed in this cell. Displaying text in a row of a table is a very common task, so the table view cell provides a `UILabel` property called `textLabel` that we can set in order to display strings. That just requires getting the correct string from our `listData` array and using it to set the cell's `textLabel`.
- To get the correct value, however, we need to know which row the table view is asking for. We get that information from the `indexPath`'s `row` property. We use the row number of the table to get the corresponding string from the array, assign it to the cell's `textLabel.text` property, and then return the cell.
- To get the correct value, however, we need to know which row the table view is asking for. We get that information from the `indexPath`'s `row` property. We use the row number of the table to get the corresponding string from the array, assign it to the cell's `textLabel.text` property, and then return the cell.
- `cell.textLabel.text = self.dwarves[indexPath.row]; return cell;`

# ADDING AN IMAGE

- Each cell has an imageView property.
- Each imageView has an image property, as well as a highlightedImage property. The image appears to the left of the cell's text and is replaced by the highlightedImage, if one is provided, when the cell is selected.
- You just set the cell's imageView.image property to whatever image you want to display.
- Add following code in- `(UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath` methodd , the code is :
- Adding Normal image:  
`UIImage *image = [UIImage imageNamed:@"normal.png"];`
- `cell.imageView.image = image;`
- Adding highlighted image:  
`UIImage *image = [UIImage imageNamed:@"highlighted.png"];`

# CHOOSING ACCESSORY TYPE

- Add following code in same function:
- `cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;`
- You Can also give custom image to the accessor indicator.To do so add follwing code.
- `cell.accessoryView = [[UIImageView alloc] initWithImage:[UIImage imageNamed:@"disc.png"]];`

# CHANGING BACKGROUND COLOR OF THE SELECTED CELL

- To change the background color of a selected Cell we do so.
- ```
UIView *selectedBackgroundViewForCell  
= [UIView new];  
[selectedBackgroundViewForCell  
setBackgroundColor:[UIColor  
blackColor]];  
cell.selectedBackgroundView  
=selectedBackgroundViewForCell;
```

# CHANGING FONT COLORS AND FONT STYLE AND FONT SIZE

- Setting Text color for cell Text Label
- `cell.textLabel.textColor=[UIColor blackColor];`
- Setting Text color when cell is selected
- `cell.textLabel.highlightedTextColor = [UIColor whiteColor];`
- setting font and text size
- `cell.textLabel.font = [UIFont fontWithName:@"Times New Roman" size:18.0f];`

# SETTING TABLE VIEW BACKGROUND COLOR

- Make an Outlet of Table and Connect it with the table in interface Builder.
- Synthesize the table outlet
- First set cell color to clear color as in order to enable table background color work.
- `Cell.backgroundColor=[UIColor clearColor];`
- Then in viewDidLoadMethod add follwing code
- `[table setBackgroundColor:[UIColor colorWithRed:(255/255.0) green:(193/255.0) blue:(37/255.0) alpha:1]]`

# SETTING ROW HEIGHT OF THE TABLE VIEW CELL

- We have a special method to accomplish this task
- Add this method to **BIDTableViewController.m** file
- Add the following code which includes method and expected row height.
- ```
- (CGFloat)tableView:(UITableView *)tableView  
heightForRowAtIndexPath:(NSIndexPath *)indexPath  
{  
return 70;  
}
```

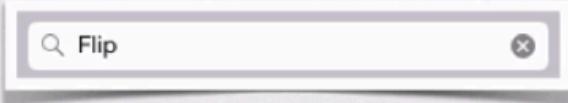
# SETTING INDENT LEVEL

- We again have a special method to add indent level to each row of UITableView.
- Method and Code is given by. Add in same class as did in last slide
- -(NSInteger)tableView:(UITableView \*)tableView indentationLevelForRowAtIndexPath: (NSIndexPath \*)indexPath

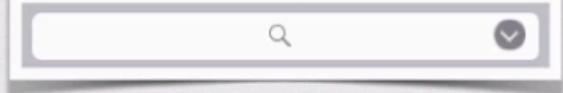
```
{  
return indexPath.row;  
}
```

# SEARCH BAR OPTIONS

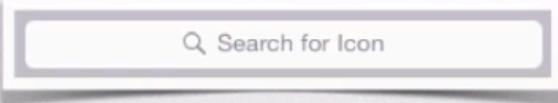
**text**



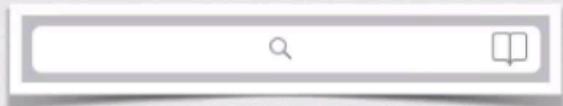
**show search results button**



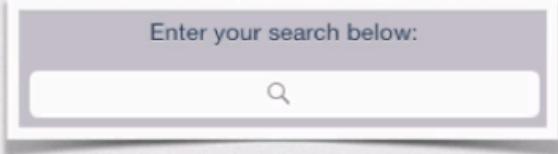
**placeholder**



**show bookmarks button**



**prompt**



# SEARCH BAR DELEGATES

## **UISearchBarDelegate**

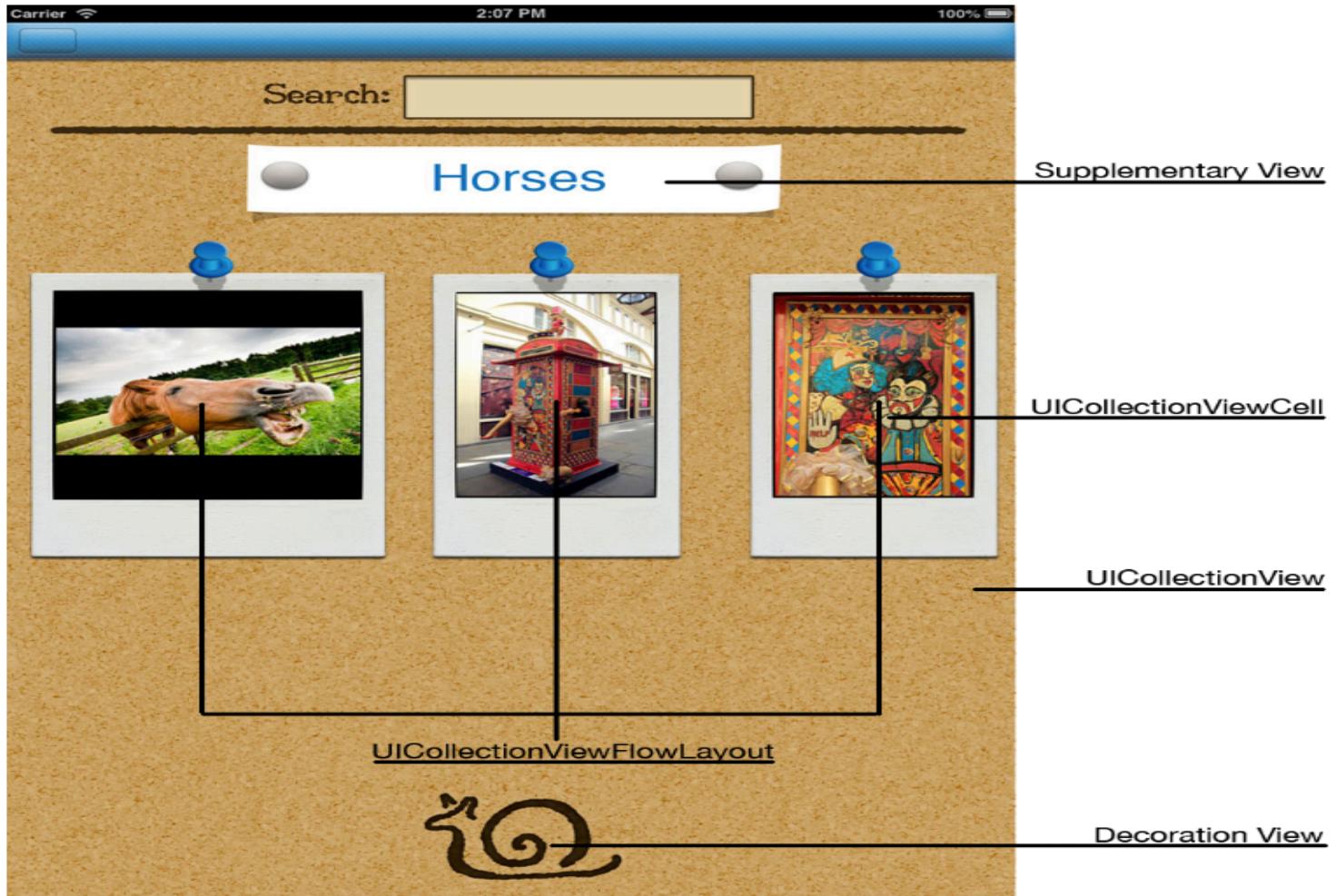
- ... search bar text changed
- ... search bar button clicked (bookmark, results,
- ... scope changed

## **UISearchDisplayDelegate**

- ... when user starts/stops searching
- ... when the search results are shown/hidden
- ... if a search string should cause a table reload

**COLLECTION VIEW**

# COLLECTION VIEW



# COLLECTION VIEW

- **UICollectionView** - the main view in which the content is displayed, similar to a **UITableView**. Note that it doesn't necessarily have to take up the entire space inside the view controller - in the screenshot above, there's some space above the collection view where the user can search for a term.
- **UICollectionViewCell** - similar to a **UITableViewCell** in **UITableView**. These cells make up the content of the view and are added as subviews to the **UICollectionView**. Cells can either be created programmatically, inside Interface Builder, or via a combination of the two methods.

# COLLECTION VIEW

- **Supplementary Views** - if you have extra information you need to display that shouldn't be in the cells but still somewhere within the UICollectionView, you should use supplementary views. These are commonly used for headers or footers of sections.
- **Decoration View** – if you want to add some extra views to enhance the visual appearance of the UICollectionView (but don't really contain useful data), you should use decoration views. Background images or other visual embellishments are good examples of decoration views.

# NON VISUAL COMPONENTS

- **UICollectionViewLayout** – UICollectionView does not know anything about how to set up cells on screen. Instead, its UICollectionViewLayout class handles this task. It uses a set of delegate methods to position every single cell in the UICollectionView. Layouts can be swapped out during runtime and the UICollectionView can even automatically animate switching from one layout to another!
- **UICollectionViewFlowLayout** – You can subclass UICollectionViewLayout to create your own custom layouts (as you'll learn about in the next chapter), but Apple has graciously provided developers with a basic "flow-based" layout called UICollectionViewFlowLayout. It lays elements out one after another based on their size, quite like a grid view. You can use this layout class out of the box, or subclass it to get some interesting behavior and visual effects.

# UICOLLECTIONVIEWDATASOURCE

- `collectionView:numberOfItemsInSection:`: returns the number of cells to be displayed for a given section.
- `numberOfSectionsInCollectionView:`: returns the total number of sections
- `collectionView:viewForSupplementaryElementOfKind:atIndexPath:`: is very simple, even though it has a crazy signature. It is responsible for returning a view for either the header or footer for each section of the UICollectionView. The variable “kind” is an NSString that determines which view (header or footer) the class is asking for.

# UICOLLECTIONVIEWDATAS OURCE

- `collectionView:cellForItemAtIndexPath:`: is responsible for returning the cell at a given index path. Similarly to table view cells, collection view cells are put into a reuse queue and are de-queued using a reuse identifier. Unlike `UITableViewCell`, `UICollectionViewCell` doesn't have a default cell style. So the layout of the cell has be specified by you.

# FLOW LAYOUT

- **minimumLineSpacing**
  - A floating point value that dictates to the flow layout the minimum number of points it has to reserve between each row. The layout object may decide to allocate more space in order to make the layout look good, but it must not allocate less. If your collection view is too small for the items to fit into it, your items will be clipped, just like any other view in the iOS SDK.
- **minimumInteritemSpacing**
  - A floating point value to indicate the minimum number of points that the layout should reserve between cells on the same row. Again, this is the minimum number of points, and the layout, depending on the size of the collection view, may decide to increase this number.
- **itemSize**
  - A CGSize that specifies the size of every cell in the collection view.

# FLOW LAYOUT

- **scrollDirection**
  - A value of type UICollectionViewScrollDirection that tells the flow layout how the collection view’s contents have to be scrolled. You can have the contents scroll either vertically or horizontally, but not both. The default value of this property is UICollectionViewScrollDirectionVertical, but you can change it to UICollectionViewScrollDirectionHorizontal.
- **sectionInset**
  - A value of type UIEdgeInsets that specifies the margins around every section. The margins are basically spaces that will not be occupied by any cells. You can use the `UIEdgeInsetsMake` function to create these insets, which are made out of top, left, bottom, and right edges, each of type float. Don’t worry if you find this explanation confusing; we will look at an example soon.

# FLOW LAYOUT

- (UICollectionViewFlowLayout \*) flowLayout{

```
UICollectionViewFlowLayout *flowLayout =  
[[UICollectionViewFlowLayout alloc] init];
```

```
flowLayout.minimumLineSpacing = 20.0f;  
flowLayout.minimumInteritemSpacing = 10.0f;  
flowLayout.itemSize = CGSizeMake(80.0f, 120.0f);  
flowLayout.scrollDirection = UICollectionViewScrollDirectionVertical;  
flowLayout.sectionInset = UIEdgeInsetsMake(10.0f, 20.0f, 10.0f, 20.0f);  
  
return flowLayout;  
}
```

# SQLITE DATABASE

# INTRODUCTION

- A lightweight yet powerful relational database engine that is easily embedded into an application.
- SQLite3 is very efficient at storing and retrieving large amounts of data. It's also capable of doing complex aggregations on your data, with much faster results than you would get doing the same thing using objects.
- Why SQLite?
  - First, it's quite easy to pull out only a subset of the data available from the database at a time, which is good for memory usage and speed.
  - Secondly, the database engine can do a lot of neat work for you such as giving you counts of rows or sums of fields quite easily.

# SQLITE3

- SQLite is an ACID compliant database that implements most of the SQL standard.
- The entire database is stored in a single file and "runs" in the same process as your app.
- The database is accessed through a C API.

# STORAGE TYPES

- SQLite uses five storage classes to represent all the different types.  
These types are:
- Null,
- Integer,
- Real,
- Text,
- Blob

# CREATE A NEW DATABASE

- Create a new database in code in the command line and copy the database file into app bundle.
  - \$ sqlite3 studentdb.sqlite
- Or else use dbmanager to create db and put the db file in app folder

```
if ([filemgr fileExistsAtPath:_databasePath] == NO)
{
// The database does not exist, so copy the default to the appropriate
location.
NSString *defaultDBPath = [[NSBundle mainBundle]
pathForResource:@"contacts.db" ofType:nil];
BOOL success = [fileManager copyItemAtPath:defaultDBPath
toPath:writableDBPath error:&error];
if (!success)
{
    // Failed to copy db from bundle to documents folder.
}}
```

# CREATE A NEW TABLE

```
char * errorMessage;
const char * sql = "create table if not exists
Student(ID Integer Primary key AutoIncrement,
Fname Text, Lname Text, DOB Date);";
!
sqlite3_exec(db, sql, NULL, NULL,
&errorMessage);
```

- **sqlite3\_exec** is convenience method for calling multiple statements of SQL without using lots of C code.
- The NULL arguments are positions for callbacks (if necessary).

# INSERTING DATA

```
sqlite3_stmt * statement; const char * sql = "insert into
Student values (0, 'Daniel', 'Charles', '2014-08-10
11:00:00'); sqlite3_prepare_v2(db, sql, -1,
&statement, NULL); if (sqlite3_step(statement) ==
SQLITE_DONE) {
    NSLog(@"Inserted");
}
sqlite3_finalize(statement);
```

- **sqlite3\_prepare\_v2** - compiles the statement.
- **sqlite3\_step** - runs the statement.
- **sqlite3\_finalize** - cleans up the memory.

# SELECTING DATA

```
sqlite3_stmt * selectStatement; const char * select_sql = "select * from Student";
if (sqlite3_prepare_v2(db, select_sql, -1, &selectStatement, NULL) == SQLITE_OK)
{
    while (sqlite3_step(selectStatement) == SQLITE_ROW)
    {
        int rowId = sqlite3_column_int(selectStatement, 0);
        char * fname= (char *) sqlite3_column_text(selectStatement, 1);
        char * lname= (char *) sqlite3_column_text(selectStatement, 2);
        char * date = (char *) sqlite3_column_text(selectStatement, 3); NSLog(@"%@", "%d %s %s %s", rowid, fname, lname, date);
    }
}
```

- **sqlite3\_column\_text** - gets the column by type.

# UPDATE

```
const char * update = "update Student  
set fname= 'Rahul' where ID = 0"; char  
* errorMessage;  
  
if (sqlite3_exec(db, update, NULL,  
NULL, &errorMessage) != SQLITE_OK)  
{  
}  
}
```

# DELETE

```
const char * rmSql = "delete from Student  
where ID = 0";  
  
char * errorMessage;  
  
if (sqlite3_exec(db, rmSql, NULL, NULL,  
&errorMessage) != SQLITE_OK)  
{  
}  
}
```

# SANITIZE USER INPUT

- When dealing with user input, you should parameterize the queries through prepared statements

```
const char * sql = "select * from videos  
where title like '%?%';
```

- Bind text to each of your substitutions.  

```
sqlite3_bind_text(sql, 1, [_title UTF8String],  
-1, SQLITE_TRANSIENT)
```

# BINDING

- **Text** : int sqlite3\_bind\_text(sqlite3\_stmt\*, int, const char\*, int n, void(\*), void(\*));
- **Text16** : int sqlite3\_bind\_text16(sqlite3\_stmt\*, int, const char\*, int n, void(\*), void(\*));
- **Null** : int sqlite3\_bind\_text(sqlite3\_stmt\*, int);
- **Blob** : int sqlite3\_bind\_blob(sqlite3\_stmt\*, int, const char\*, int n, void(\*), void(\*));
- **Int** : int sqlite3\_bind\_int(sqlite3\_stmt\*, int, int);
- **Int64** : int sqlite3\_bind\_int64(sqlite3\_stmt\*, int, int);
- **Double** : Int64 : int sqlite3\_bind\_double(sqlite3\_stmt\*, int, double);
- **Value** : int sqlite3\_bind\_value(sqlite3\_stmt\*, int, const sqlite3\_value);
- **Zeroblob** : int sqlite3\_bind\_blob(sqlite3\_stmt\*, int, int n);

CORE DATA

# PER THE DOCUMENTATION:

- The Core Data framework provides generalized and automated solutions to common tasks associated with object life-cycle and object graph management, including persistence.

# WHAT ISN'T: CORE DATA?

- It's not an RDBMS.
- If you want a database and SQL, use Sqlite:
- It's not just an ORM (Object Relational Mapper)
- It may look like there's SQL under the hood, but that's not necessarily the case.

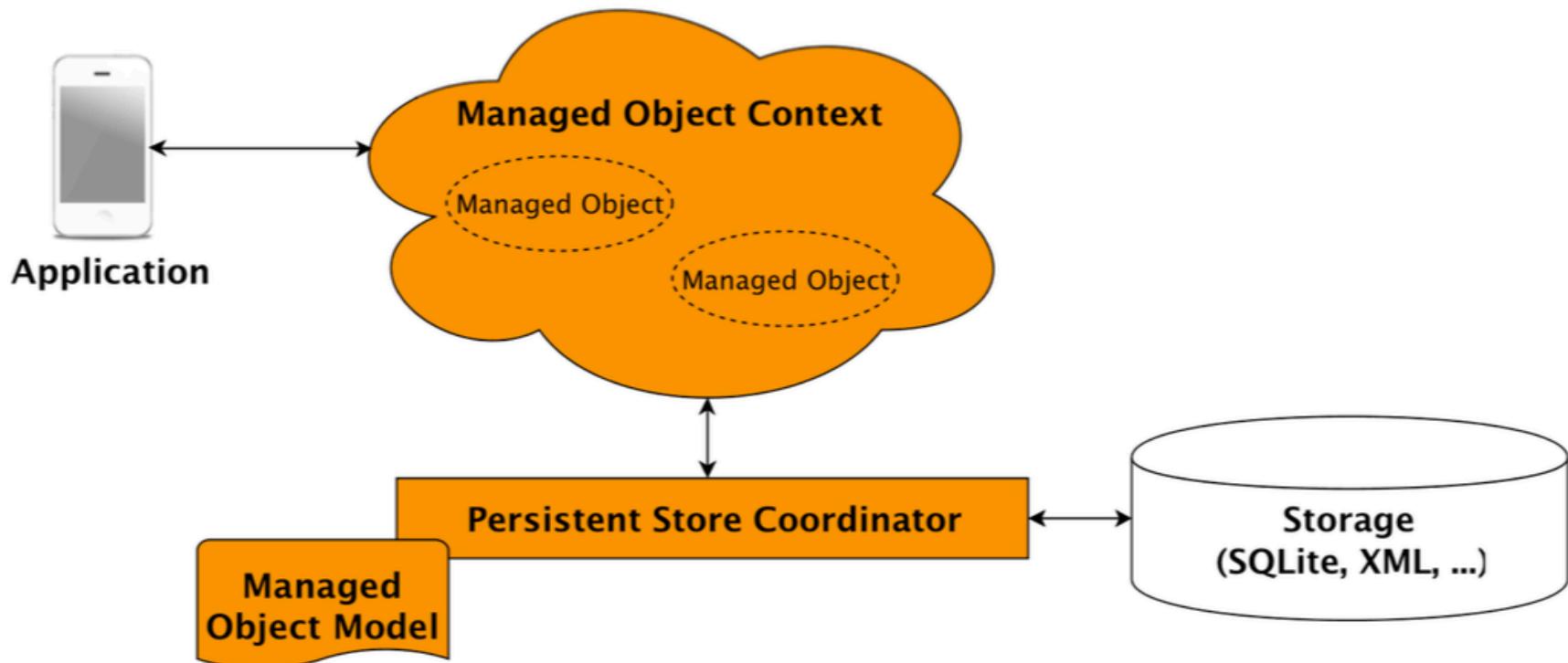
# So, what is it then?

- Core Data provides:
  - persistence
  - change tracking
  - relations (object graph)
  - lazy loading ( “faulting” )
  - validation
  - works well with Cocoa (KVO, KVC)

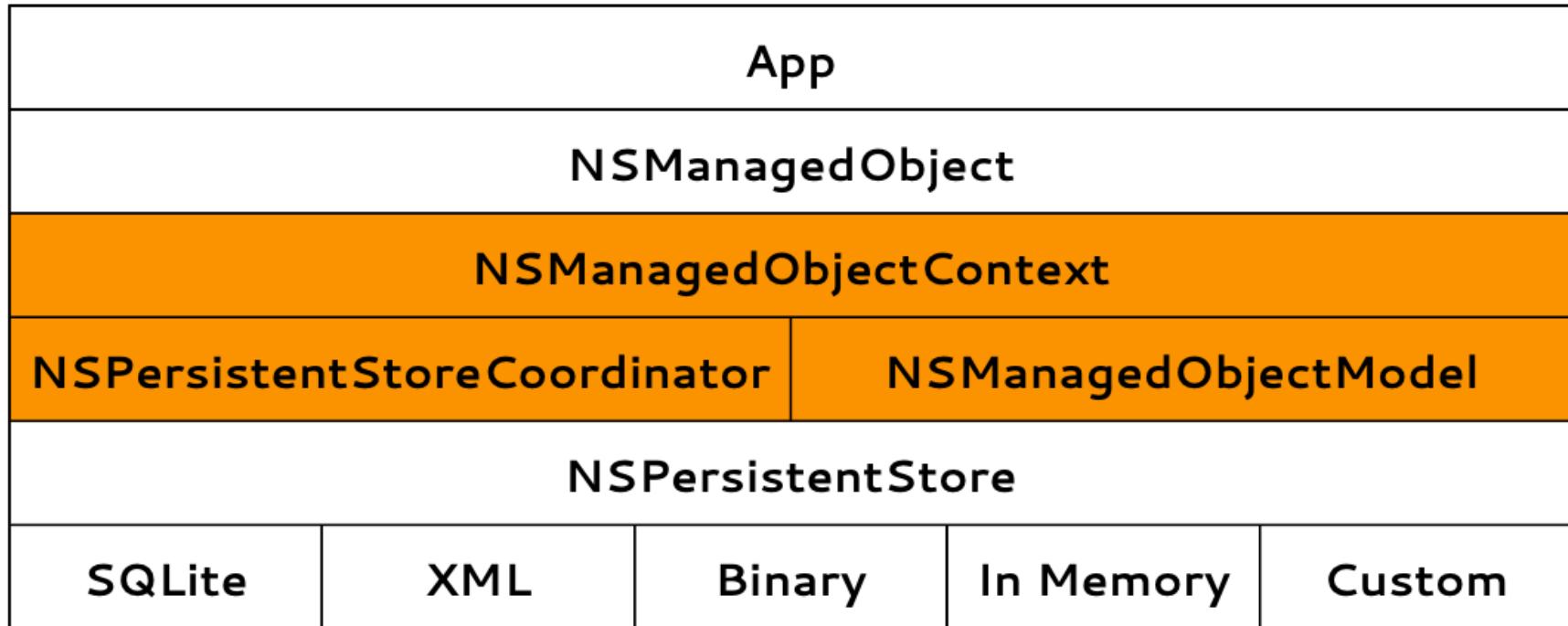
# BASICALLY:

- A system to store data
- Persistence agnostic (local storage, iCloud, ...)
- No need to write SQL to query
- You can keep to Objective-C

# CORE DATA



# CORE DATA



# MANAGED OBJECT MODEL

- The first step is to create a **managed object model**, which spells out the way Core Data represents data on disk.
- By default, Core Data uses an SQLite database as the persistent store, so you can think of the data model as the database schema.
- This is a simple file on disk that will represent our data model. Think about it as your database schema.

# NSMANAGEDOBJECT

- NSManagedObject represents a single object stored in Core Data—you must use it to create, edit, save and delete from your Core Data persistent store.
- NSManagedObject is a shape-shifter. It can take the form of any entity in your data model, appropriating whatever attributes and relationships you defined.

# NSMANAGEDOBJECT

- Represents an "entity" in the database.
- Has a reference to the actual entity description.

Managed objects are managed by the Managed Object Context.

- Can be subclassed and new methods and properties can be added.

# NSMANAGEDOBJECTCONT EXT

- managed object context is an in-memory “scratchpad” for working with managed objects.
- first, you insert a new managed object into a managed object context; then, after you’re happy with your shiny new managed object, you “commit” the changes in your managed object context to save it to disk.
- Managed Object Context contains reference to actual data “store”
- The context can be queried for certain objects using what are called Fetch Requests.

# ENTITY

- An entity is a class definition in Core Data. The classic example is an Employee or a Company.
- In a relational database, an entity corresponds to a table.

# ATTRIBUTE

- An **attribute** is a piece of information attached to a particular entity. For example, an Employee entity could have attributes for the employee's name, position and salary.
- In a database, an attribute corresponds to a particular field in a table.

# RELATIONSHIP

- A **relationship** is a link between multiple entities. In Core Data, relationships between two entities are called **to-one relationships**, while those between one and many entities are called **to-many relationships**. For example, a Manager can have a to-many relationship with a set of employees, whereas an individual Employee will have a to-one relationship with his manager.

# NSENTITYDESCRIPTION

- NSManagedObject a “shape-shifter” class because it can represent any entity.
- An entity description is the piece that links the entity definition from your data model with an instance of NSManagedObject at runtime.

# FETCH REQUEST

- Retrieves data from the managed object context.
- Returned object types are determined by supplying an `NSEntityDescription`.
- Use an `NSPredicate` to add constraints to your search. Provide a sort descriptor to organize your sort results.

# NSFetchRequest

- NSFetchedRequest is the class responsible for fetching from Core Data. Fetch requests are both powerful and flexible. You can use requests to fetch a set of objects that meet particular criteria (e.g., "give me all employees that live in Wisconsin and have been with the company at least three years"), individual values (e.g., "give me the longest name in the database") and more.

# PERSISTENT STORE

- This is where the data is actually stored. SQLite by default, but can be other formats as well.
- You can provide your own custom store.
- Core Data also provide the ability to have an in memory store.

# PERSISTENT STORE COORDINATOR

- The object that coordinates reading and writing of information from and to the persistent store.
- The coordinator is the bridge between the managed object context and the persistent store. Can connect multiple persistent stores into a single unified place
- Data fetched from the persistent store coordinator unless a specific store is designated.
- It links to managed object model that which is used to define objects

# MODELING OBJECTS

- Use a graphical tool to visually design your managed objects.
- Objects are defined using basic types.
- Relationships can be set up between objects.

# NSFETCHEDRESULTSCONT ROLLER

- Acts as a datasource for a table view.
- Can optionally cache the results.
- As the model changes, it will automatically update the table.

# WHY NSFETCHEDRESULTSCONT ROLLER ?

- After a fetched results controller is created on a managed object context, any change (insertion, deletion, modification, etc.) will immediately be reflected on the fetched results controller as well.
- For instance, you could create your fetched results controller to read the managed objects of the Person entity. Then in some other place in your application, you might insert a new Person managed object into the context (the same context the fetched results controller was created on). Immediately, the new managed object will become available in the fetched results controller.
- With a fetched results controller, you can manage cache more efficiently. For instance, you can ask your fetched results controller to keep only N number of managed objects in memory per controller instance.
- Fetched results controllers are exactly like table views in the sense that they have sections and rows, as explained before. You can use a fetched results controller to present managed objects in the GUI of your application with table views with ease.

# IMPORTANT PROPERTIES

- **sections (property, of type NSArray)"**
  - A fetched results controller can group data together using a key path. The designated initializer of the NSFetchedResultsController class accepts this grouping filter through the sectionNameKeyPath parameter. The sections array will then contain each grouped section. Each object in this array conforms to the NSFetchedResultsSectionInfo protocol.

# IMPORTANT PROPERTIES

- **objectAtIndexPath:** (instance method, returns a managed object)
  - Objects fetched with a fetched results controller can be retrieved using their section and row index. Each section's rows are numbered 0 through N-1, where N is the total number of items in that section. An index path object comprises a section and row index and perfectly matches the information needed to retrieve objects from a fetched results controller. The objectAtIndexPath: instance method accepts index paths. Each index path is of type NSIndexPath. If you need to construct a table view cell using a managed object in a fetched results controller, simply pass the index path object in the tableView:cellForRowAtIndexPath: delegate method of a table view. If you want to construct an index path yourself anywhere else in your application, use the indexPathForRow:inSection: class method of NSIndexPath.

# IMPORTANT PROPERTIES

- **fetchRequest (property, of type NSFetchedResultsController)**
  - If at any point in your application you believe you have to change the fetch request object for your fetched results controllers, you can do so using the fetchRequest property of an instance of NSFetchedResultsController. This is useful, for example, if you want to change the sort descriptors of the fetch request object after you have allocated and initialized your fetched results controllers.

# NSFETCHEDRESULTSCONTROLLERDELEGATE

- `controllerWillChangeContent:`
  - Gets called on the delegate to let it know that the context that is backing the fetched results controller has changed and that the fetched results controller is about to change its contents to reflect that. We usually use this method to prepare our table view for updates by calling the `beginUpdates` method on it.
- `controller:didChangeObject:atIndexPath:forChangeType:newIndexPath:`
  - Gets called on the delegate to inform the delegate of specific changes made to an object on the context. For instance, if you delete an object from the context, this method gets called, and its `forChangeType` parameter will contain the value `NSFetchedResultsControllerChangeDelete`. Alternatively, if you insert a new object into the context, this parameter will contain the value `NSFetchedResultsControllerChangeInsert`.
  - This method also gets called on your fetched results controller's delegate method when a managed object is updated, after the context is saved using the `save:` method of the context

# NSFETCHEDRESULTSCONTROLLERDELEGATE

- `controllerDidChangeContent:`
  - Gets called on the delegate to inform it that the fetched results controller was refreshed and updated as a result of an update to a managed object context. Generally, programmers issue an `endUpdates` call on their table view within this method to ask the table view to process all the updates that they submitted after a `beginUpdates` method.

# WEB SERVICE & XML PARSERS

# WEB SERVICE

- Web Services can convert your application into a Web-application, which can publish its function or message to the rest of the world.
- The basic Web Services platform is XML + HTTP.
  - XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions.
  - The HTTP protocol is the most used Internet protocol.
- Two types
  - Reusable application-components.
  - Connect existing software.

# SOAP SERVICE

- SOAP is a protocol for accessing a Web Service.
- SOAP stands for Simple Object Access Protocol
- A SOAP message is an ordinary XML document containing the following elements:
  - An Envelope element that identifies the XML document as a SOAP message
  - A Header element that contains header information
  - A Body element that contains call and response information
  - A Fault element containing errors and status information

# SOAP SERVICE (CONTINUED...)

- Skeleton SOAP Message

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Header>
    ...
  </soap:Header>
  <soap:Body>
    ...
    <soap:Fault>
      ...
    </soap:Fault>
  </soap:Body>
</soap:Envelope>
```

# REST SERVICE

- REST stands for Representational State Transfer, this basically means that each unique URL is a representation of some object. HTTP protocol is used to get the content (HTTP GET/POST)
- REST allows better support for browser clients due to it's support for JSON.
- REST dominates their architecture for web services.
  - REST will give human-readable results. REST is stateless. REST services are easily cacheable.
- REST is limited by HTTP itself which can't provide two-phase commit across distributed transactional resources, but SOAP can.

# XML PARSING

# NSXMLPARSER

- It is the default XML parser included in iOS  
It is a **SAX** (Simple API for XML) based XML parser.
- It is fast and must be tuned for the XML feed that you are parsing.
- Uses a delegate to handle events as they occur while parsing an XML document.

# NSXMLPARSER CONS

- NSXMLParser does not perform any validation on your XML document.
- It is not meant to modify the XML document and write it to disk.
- To construct your own tree representation of the XML document, you will have to do it by hand.
- Can be a bit unwieldy.

# USING XMLPARSER

- Your class must implement the NSXMLParserDelegate protocol.

```
self.xmlParser = [[NSXMLParser alloc]
initWithData:xmlData];
self.xmlParser.delegate = self;
[self.xmlParser parse];
```

# IMPORTANT METHODS

- `parser:didStartElement:elementName:namespace:qualifiedName:attributes`
- `parser:foundCharacters:`
- `parser:didEndElement:elementName:namespace:qualifiedName`

# DOM BASED PARSER

- DOM based parsing builds an entire tree of the XML inside of your app.
- Allows you to run XPath queries on the XML tree.
- Some libraries give you the ability to change the actual XML data and write it back to disk.

JSON

# WHAT IS JSON?

- JSON stands for JavaScript Object Notation.  
It is a human readable format used to transmit data. JSON is formatted in attribute-value pairs.
- It is an alternative to XML for sending data across the network.

# WHAT IS JSON?

- JSON is a simple human readable format that is often used to send data over a network connection.
- For example, if you have an array of three strings, the JSON representation would simply be:  
["test1", "test2", "test3"]
- That is actually pretty similar to declaring a new array in modern Objective-C:  
@[@"test1", @"test2", @"test3"]
- If you have a Pet object with member variables name, breed, and age, the JSON representation would simply be:  
{"name" : "Dusty", "breed": "Poodle", "age": 7}
- Again very, very similar to the corresponding Objective-C code:  
@{@name : @"Dusty", @breed: @"Poodle", @age: @7}

# JSON TYPES

- Primitive JavaScript Types: String, Boolean, Number, Null Arrays are indicated by brackets - [ ]
- Objects are indicated by braces - { }  
Objects are a collection of name-value pairs.
- Use commas to add additional fields.

# NSJSONSERIALIZATION

- Converts JSON to foundation objects or can convert foundation objects to JSON.
- The top level object is an NSArray or an NSDictionary.
- All objects are instances of NSString, NSNumber, NSArray, NSDictionary, or NSNull.
- All dictionary keys are NSStrings.

# JSON DECODING

- `+JSONObjectWithData:options:error:`  
`+JSONObjectWithStream:options:error:`
- The JSON must be encoded in the following formats: UTF-8, UFT-16LE, UTF-16BE, UTF-32LE, UTF-32BE
- Takes three different options:
- `NSJSONReadingMutableContainers`  
`NSJSONReadingMutableLeaves`
- `NSJSONReadingAllowFragments`

# JSON ENCODING

- `dataWithJSONObject:options:error`  
`+writeJSONObject:toStream:options:error:`
- If method produce invalid JSON, then an exception will be thrown.  
Use `isValidJSONObject` to determine if object can be
- encoded into JSON.  
Setting the option `NSJSONWritingPrettyPrinted` will use whitespace to make the JSON human readable.

# JSON VS. XML

## Use JSON

Seems easy ; if you want to serialize a data structure that's not too text-heavy and all you want is for the receiver to get the same data structure with minimal effort, and you trust the other end to get the i18n right, JSON is the way to go.

Use XML · If you want to provide general-purpose data that the receiver might want to do unforeseen weird and crazy things with, or if you want to be really paranoid and picky about i18n, or if what you're sending is more like a document than a struct, or if the order of the data matters, or if the data is potentially long-lived (as in, more than seconds) XML is the way to go

# AUTOLAYOUT

# WHAT IS AUTOLAYOUT?

- Correctly using autolayout requires you to change thinking
- Autolayout works in a different way than what we were used to
- Don't go calculating frames anymore
- Define how an element should behave according to it's peers:
- parent › siblings
- Don't think in absolute terms, but think in relational terms
- Think **flexible**: it's not because an element has a certain size at one point, it will have the same size at another point

# AUTO LAYOUT

- Using Auto Layout, you can create a dynamic and versatile interface that responds appropriately to changes in screen size, device orientation, and localization.

# CONSTRAINTS

- Constraints are rules that describe a layout
- Constraints can be added between views or on just one view
- Constraints must collectively create an unambiguous layout

# CONSTRAINT TERMINOLOGY

- Top Space
- Bottom Space
- Leading Space
- Trailing Space
- Top Layout Guide
- Bottom Layout Guide

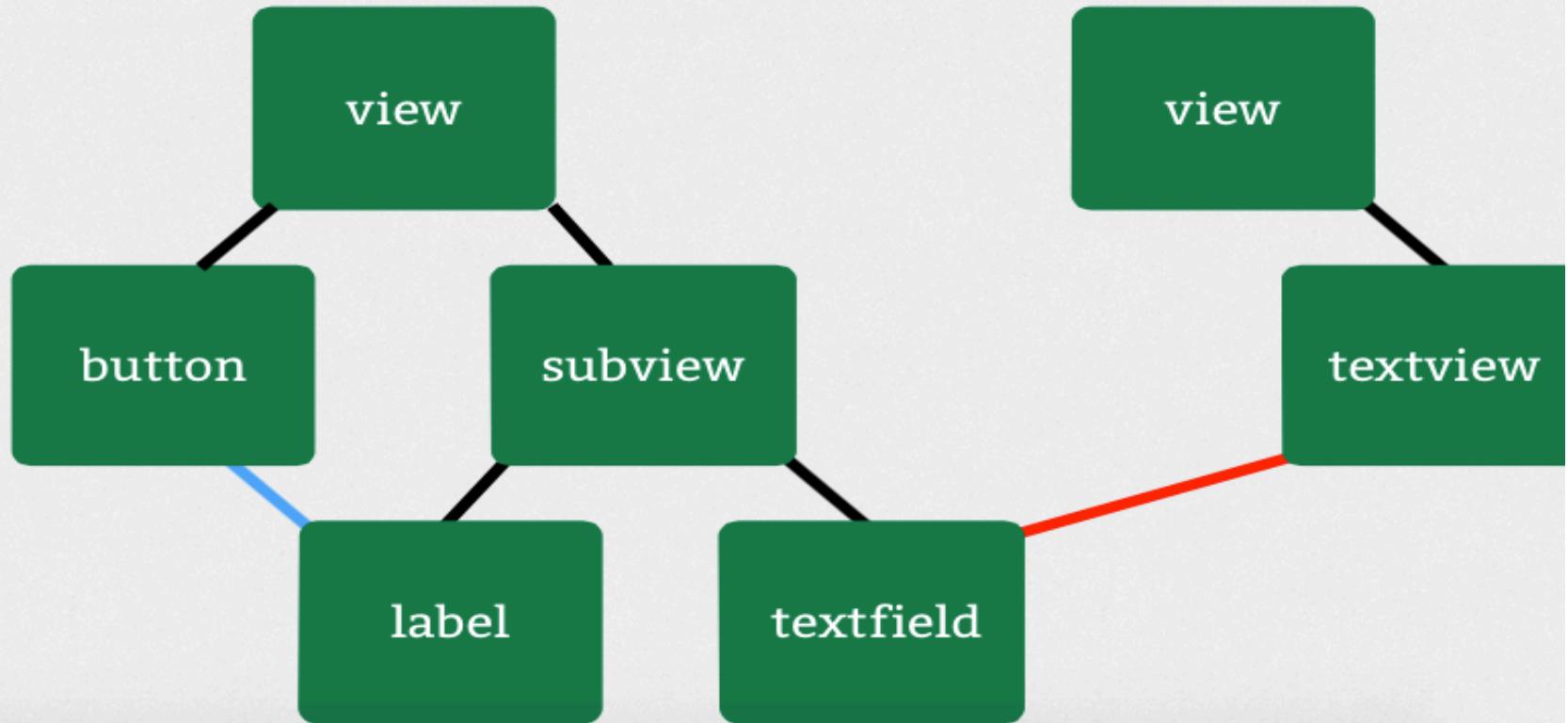
# INTERFACE BUILDER AND CONSTRAINTS

- Constraints must be manually added, updated, and/ or removed
- By default, moving or resizing a view will not change its constraints
- Interface Builder will provide constraints for "under constrained" layouts.

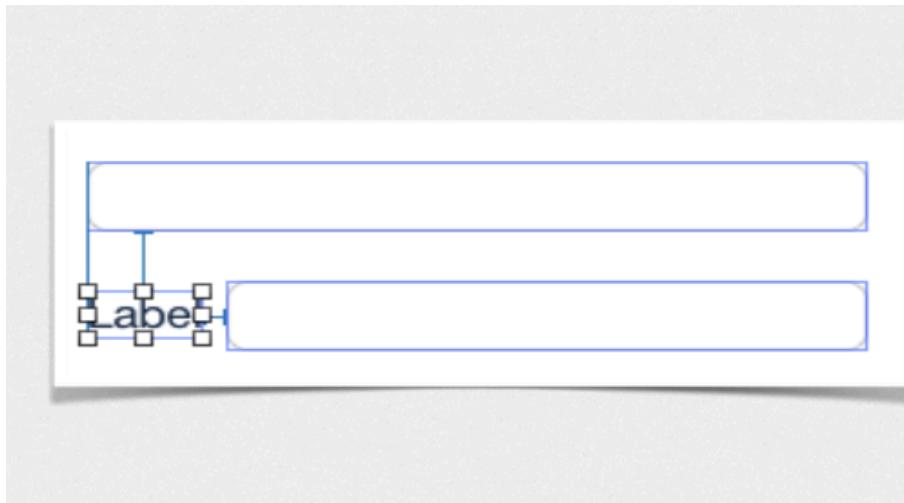
# **WHERE TO ADD CONSTRAINTS**

- **Between a view and its superview:**  
Add to the superview.
- **Between two views with the same superview:** Add to the superview.
- **On just the view:** Add to the view itself.

# VIEW TREE



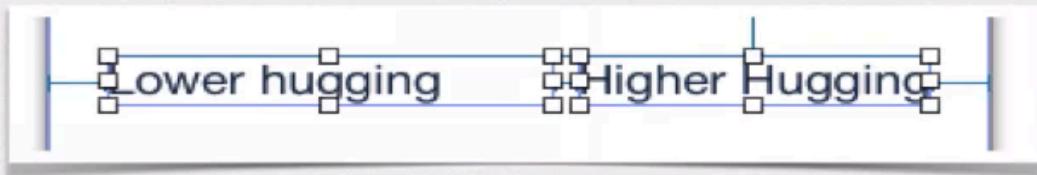
# INTRINSIC CONTENT SIZE



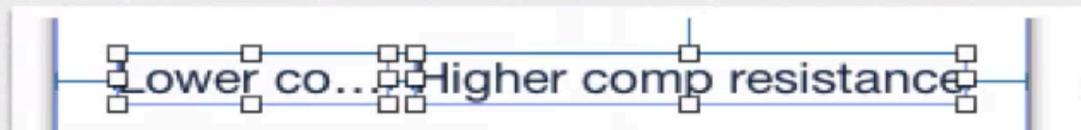
# CONTENT HUGGING AND COMPRESSION RESISTANCE

## Hugging vs. Resistance

**Content Hugging:** Don't want to grow



**Content Compression Resistance:** Don't want to shrink



# CONSTRAINTS

- **Layout** `NSLayoutConstraint`
- **Content** `NSContentSizeLayoutConstraint`
- **Autosizing** `NSAutoresizingMaskLayoutConstraint`

# CONSTRAINTS

a mathematical representation of a human-expressable statement

"the left edge should be 20pt of the  
superviews left edge"



view.left = (superview.left + 20)



$$y = m * x + c$$

(**y** = attribute1 = view.left)  
(**x** = attribute2 = superview.left)  
(**m** = multiplier = 1)  
(**c** = constant = 20)

# CONSTRAINTS ATTRIBUTE

- `view1.attribute1 RELATION multiplier * view2.attribute2 + constant`
- `NSLayoutAttributeLeft`
- `NSLayoutAttributeRight`
- `NSLayoutAttributeTop`
- `NSLayoutAttributeBottom`
- `NSLayoutAttributeLeading`
- `NSLayoutAttributeTrailing`
- `NSLayoutAttributeWidth`
- `NSLayoutAttributeHeight`
- `NSLayoutAttributeCenterX`
- `NSLayoutAttributeCenterY`
- `NSLayoutAttributeBaseline`
- `NSLayoutRelationEqual`
- `NSLayoutRelationGreaterThanOrEqualTo`
- `NSLayoutRelationLessThanOrEqualTo`
- `NSLayoutAttributeNotAnAttribute`

# constraints

## Tasks

### NSLayoutConstraint

#### Creating Constraints

```
+ constraintsWithVisualFormat:options:metrics:views:  
+ constraintWithItem:attribute:relatedBy:toItem:attribute:multiplier:constant:
```

#### Accessing Constraint Data

```
priority property  
firstItem property  
firstAttribute property  
relation property  
secondItem property  
secondAttribute property  
multiplier property  
constant property
```

```
[NSLayoutConstraint  
constraintWithItem: button  
attribute: NSLayoutAttributeCenterX  
relatedBy: NSLayoutRelationEqual  
toItem: superview  
attribute: NSLayoutAttributeCenterX  
multiplier: 1.0  
constant: 0.0]
```

#### Controlling Constraint Archiving

```
shouldBeArchived property
```

# CONSTRAINTS CODE

```
view1.attribute1 RELATION multiplier * view2.attribute2 + constant  
constraint = [NSLayoutConstraint  
    constraintWithItem: view  
        attribute: NSLayoutAttributeWidth  
        relatedBy: NSLayoutRelationEqual  
            toItem: nil  
        attribute: NSLayoutAttributeNotAnAttribute  
            multiplier: 1.0  
            constant:100.0];  
[view addConstraint: constraint];  
  
constraint = [NSLayoutConstraint  
    constraintWithItem: view  
        attribute: NSLayoutAttributeWidth  
        relatedBy: NSLayoutRelationEqual  
            toItem: nil  
                attribute:  
NSLayoutAttributeNotAnAttribute  
            multiplier: 1.0  
            Constant: 80.0];  
  
[view addConstraint: constraint];
```

# CHECK FOR AMBIGUOUS LAYOUT

```
•view.hasAmbiguousLayout
•view.exerciseAmbiguityInLayout
•for (UIView *view in self.subviews) {
•  if ([view hasAmbiguousLayout]) {
•    NSLog(@"%@", view.description, (int)self);
•  }
•}
```

# INTRINSIC CONTENTSIZE

- Suggested size for the view.

```
• - (CGSize) intrinsicContentSize {  
•     return mySize;  
• }  
• [self invalidateIntrinsicContentSize];  
• UIImage *img = UIImage imageNamed:@"Icon.png";  
• UIImageView *iv = [[UIImageView alloc] initWithImage:img];  
• NSLog(@"%@", NSStringFromCGSize(iv.intrinsicContentSize));
```

**VISUAL FORMAT  
LANGUAGE**

# VISUAL FORMAT LANGUAGE

## Visual Format Language

Name of view in brackets

@“[button1]-[button2]”

Standard spacing between views.

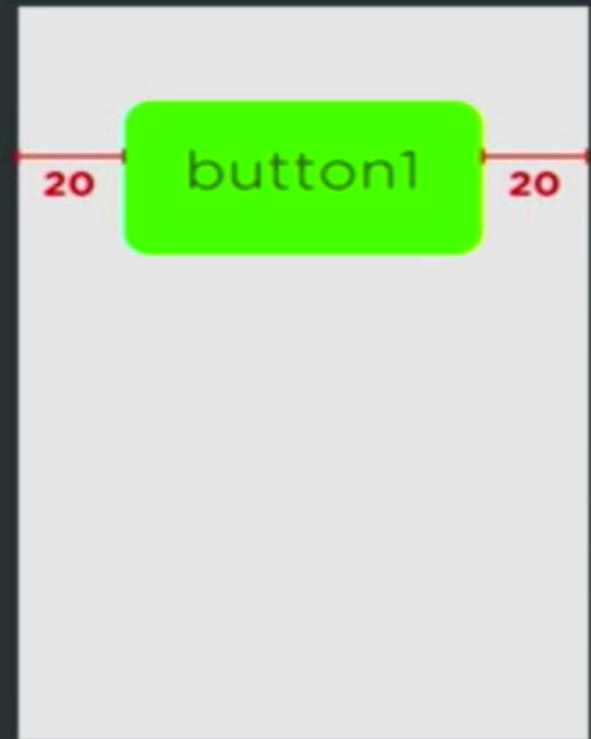


# VISUAL FORMAT LANGUAGE

## Visual Format Language

Pipe represents superview

@“|-20-[button1]-20-|”

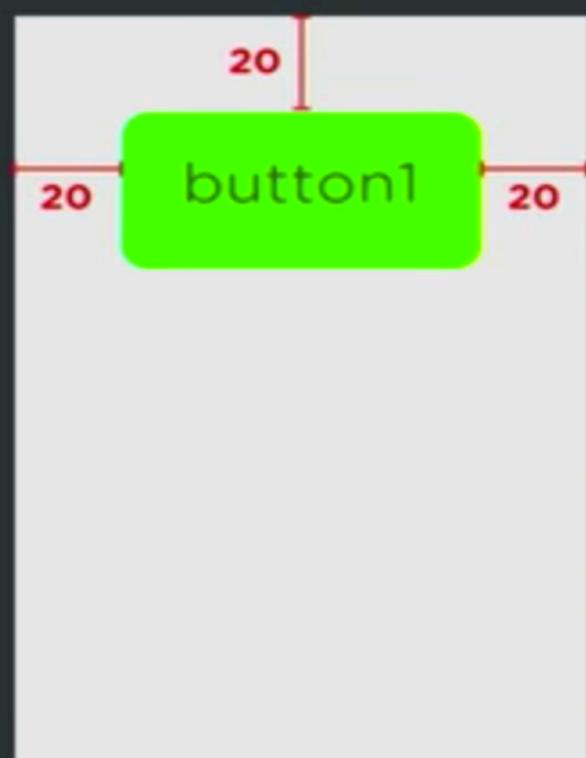


# VISUAL FORMAT LANGUAGE

## Visual Format Language

```
@“H:|-20-[button1]-20-|”
```

```
@“V:|-20-[button1]”
```



# SOCIAL NETWORKING

Very easy since iOS6

# UIACTIVITYVIEWCONTROLLER

- General purpose interface
- Supports facebook, twitter, San Weibo
- User is shown screen to choose which service
- For Posting
  - Preview of post is shown after service selection

# **SLCOMPOSEVIEWCONTROLLER**

- Allows specific service to be targeted
- Supports Facebook, Twitter, Sab Wiebo
- For Posting
  - Preview of post is shown after service selection

# SLREQUEST API

- Allows specific service to be targeted
- Supports Facebook, Twitter, Sab Wiebo
- Allows posting and reading
- Works through service API using HTTP requests
- Uses iOS Accounts framework for access

# NOTIFICATIONS

# NOTIFICATIONS

- User notifications are really just things that the user can interact with, see or hear on their device.
- We have badges, sounds and alerts
- They can come directly from your app using `UILocalNotification API`
- Alternatively, you could also have user notifications that are triggered in response to remote push payloads.

# SILENT/REMOTE NOTIFICATIONS

- Silent notifications, they are just push payloads that are sent from your APNs server that, instead of presenting a user notification like an alert or a sound or a badge on the screen, iOS, when it receives that push, will instead wake up your app in the background so that your app can do some background image processing or information processing.

# LOCAL NOTIFICATIONS

- Need to register notification settings before using it from iOS8

```
[[UIApplication  
sharedApplication]registerUserNotificatio  
nSettings:notificationSettings];
```

# GENERATE NOTIFICATION

UILocalNotifiaction =…

```
[[UIApplication  
sharedApplication]scheduleLocalNotificati  
on:notification];
```

# HANDLE NOTIFICATIONS

```
-(void)application:(UIApplication  
*)application didReceiveLocalNotification:  
(UILocalNotification *)notification{  
  
if(application.applicationState ==  
UIApplicationStateActive){  
...  
}
```

MAPKIT

Apple Maps!

# FRAMEWORKS

- Add Mapkit Framework
- Add CoreLocation Framework too

# LOCATION ON EARTH

- Earth map is a grid of points
- Every point has a **latitude/longitude** coordinate
- **latitude** - runs east/west
- **longitude** - runs north/south

# PROJECTIONS

- A **projection** is the 2D version of a 3D surface
- **Equirectangular map projection**  
Top left corner : (lat,long) : 90.00 , -180.0
- **Mercator Map Projection**  
Top left corner : (lat,long) : 85.05, -180.0

# AN ARRAY TO HOLD LOCATIONS

```
self.mapLocations =  
[@[@{@name:@"Eiffel  
Tower",@"lat":@48.8582,@"long":@2.29  
45}];
```

# MAPITEM

MKMapItem \*

MKPlacemark \*placemark

CLLocation \*location

CLLocationCoordinate2D coordinate

struct

double

double

latitude

longitude

# CREATE THE COORDINATE

```
- double lat = [self.mapLocations[0]
   [@"lat"] doubleValue]; double lng =
    [self.mapLocations[0][@"lng"]
    doubleValue]; CLLocationCoordinate2D
coord =
CLLocationCoordinate2DMake(lat,lng)
```

# CREATING MAP ITEM

- MKPlacemark \*placemark =  
[[MKPlacemark alloc]  
initWithCoordinate:coord  
addressDictionary:nil]
- MKMapItem \*mapItem = [[MKMapItem  
alloc] initWithPlacemark:placemark];

# SETTING MAPITEM PROPERTIES

- mapItem.name = self.mapLocations[0]  
["name"]
- mapItem.phoneNumber =  
@” 234234 ” ;
- mapItem.url =  
@” www.coderistitute.comN ” ;

# MAPITEM LAUNCH OPTIONS

key	value
MKLaunchOptionsMapTypeKey	MKMapTypeStandard (@0) MKMapTypeSatellite (@1) MKMapTypeHybrid (@2)
MKLaunchOptionsMapCenterKey	CLLocationCoordinate2D (NSValue)
MKLaunchOptionsMapSpanKey	MKCoordinateSpan (NSValue)
MKLaunchOptionsShowsTrafficKey	BOOL (@YES or @NO)
MKLaunchOptionsDirectionsModeKey	MKLaunchOptionsDirectionsModeDriving MKLaunchOptionsDirectionsModeWalking

# OPENING MAP WITH OPTIONS

- [mapItem openInMapsWithLaunchOptions:@{ MKLaunchOptionsMapTypeKey: @2, MKLaunchOptionsMapCenterKey: [NSValue valueWithMKCoordinate:placemark.coordinate], MKLaunchOptionsShowsTrafficKey: @YES}];

# MAPKIT INSIDE THE VIEW CONTROLLER OR APP

- Import MapKit Framework
- Adopt protocol  
`MKMapViewDelegate`
- Create property
  - `@property (strong, nonatomic)  
MKMapView *mapView`

# CREATE AND ADD AN MKMAPVIEW

- Create the frame of mapview or an outlet from storyboard
- Set the delegate

# NO ZOOM IN MAPKIT

- Two options to display a map area:
  1. Regions **MKCoordinateRegion**  
needs a center lat/lng and a width/  
height span
  2. Rects **MKMapRect**  
needs a rectangle whose  
dimensions are map points

# SET REGION FOR VISIBLE MAP

```
CLLocationCoordinate2D startCenter =  
CLLocationCoordinate2DMake(28.5407,-81.3786);  
CLLocationDistance regionWidth = 1500;  
CLLocationDistance regionHeight = 1500;  
MKCoordinateRegion startRegion =  
MKCoordinateRegionMakeWithDistance(startCenter,r  
egionWidth,regionHeight);  
  
[self.mapView setRegion:startRegion animated:YES];
```

# SHOWING THE DEVICE LOCATION ON THE MAP

- (void)viewDidLoad  
{ self.mapView.showsUserLocation = YES; }
- Drains the battery due to constant tracking
- Stays on until it is set to NO

# CENTER ON THE USER LOCATION

- In ViewDidLoad
  - [self.mapView setCenterCoordinate:self.mapView.userLocation.location.coordinate animated:YES];
- Not possible,since user location is not found immediately

# CENTER ON THE USER LOCATION

- In delegate method

```
(void)mapView:(MKMapView *)mapView  
didUpdateUserLocation:(MKUserLocation  
*)userLocation { [self.mapView  
setCenterCoordinate:  
userLocation.location.coordinate  
]}
```

- Now, it will constantly recenters on user location

# CENTER ON THE USER LOCATION

- In delegate method

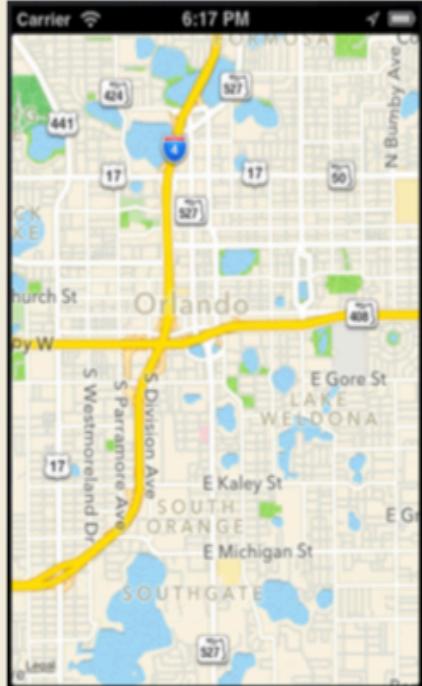
```
if(!self.userLocationUpdated) {  
    [self.mapView setCenterCoordinate:  
        userLocation.location.coordinate];  
    self.userLocationUpdated = YES;  
}
```

- Now the map stops re-centering after the first time

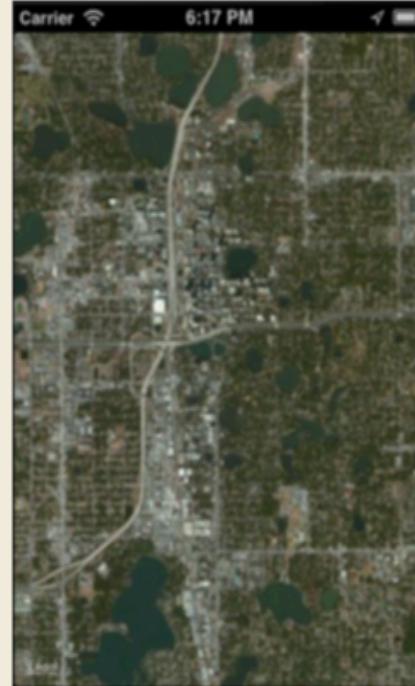
# SETTING THE MAPTYPE

```
self.mapView.mapType =  
MKMapTypeStandard;//default
```

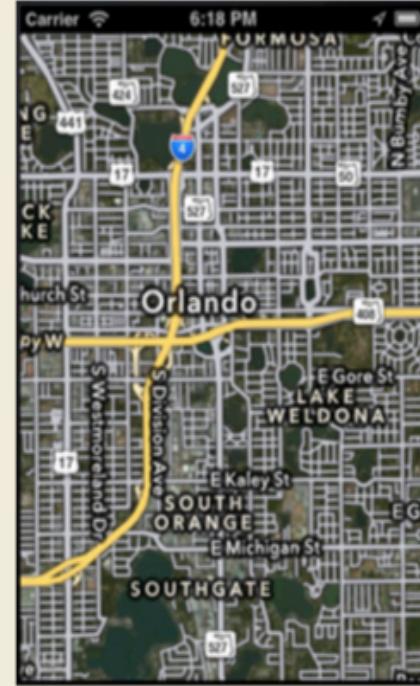
MKMapTypeStandard



MKMapTypeSatellite



MKMapTypeHybrid

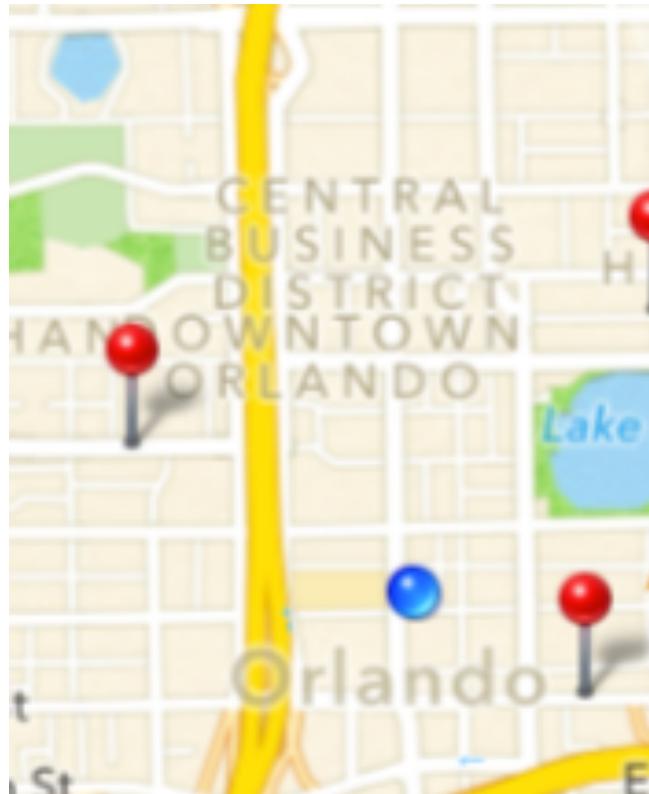


# MKMAPVIEW PROPERTIES

```
self.mapViewzoomEnabled = YES;  
self.mapView.scrollEnabled = YES;
```

- Enable or disable user control
- Zooming and scrolling can still be done programmatically

# SHOWING LOCATIONS ON THE MAP AS ANNOTATIONS



# ANNOTATION?

- Simplest annotation is an **NSObject** that:
  - Adopts the <MKAnnotation> protocol
  - Has a **CLLocationCoordinate2D** property named **coordinate**

# SIMPLEST ANNOTATION MODEL CLASS

- Adopt the protocol  
`@interface MyAnnotation :  
NSObject <MKAnnotation>`
- Must include the below property  
`@property  
CLLocationCoordinate2D coordinate;`

# MAPVIEW:VIEWFORANNOTATION DELEGATE METHOD

- **this delegate method gets called once for each added annotation**
  - `(MKAnnotationView *)mapView:(MKMapView *)mapView viewForAnnotation:(id<MKAnnotation>)annotation { }`

# MKANNOTATIONVIEW

```
-(MKAnnotationView *)mapView:(MKMapView *)mapView  
viewForAnnotation:(id<MKAnnotation>)annotation  
{  
    MKAnnotationView *view = [self.mapView  
dequeueReusableAnnotationViewWithIdentifier:@"annoView"]  
;  
    If(!view) {  
  
view = [[MKAnnotationView alloc] initWithAnnotation:annotation  
reusableIdentifier:@"annoView"];  
}  
return view;
```

# ADD AN IMAGE

- Add an image to the project
- `view.image = [UIImageView  
imageNamed:@"altPin.png"];`

# NO ANNOTATION FOR USER LOCATION

- if([annotation isKindOfClass: [MKUserLocation class]]) { return nil; }
- don't do any customization if this annotation is the blue userLocation dot!

# MKANNOTATION OBJECT WITH CALLOUT BUBBLE

- MyAnnotation.h

- Add two properties

- ```
@property (weak, nonatomic) NSString  
*title;
```

- ```
@property (weak, nonatomic) NSString  
*subtitle;
```

# ADDING CALLOUT BUBBLE

- **viewDidLoad**
  - annotation.title = @"Lake Eola";
  - annotation.subtitle = @"Cool swans";
- **Annotation view method**
  - view.canShowCallout = YES;

# ENUMERATE THRU ALL LOC AND MAKE ANNOTATIONS

```
for(NSDictionary *location in locations) {  
    CLLocationCoordinate2D annotationCoordinate =  
        CLLocationCoordinate2DMake([location[@"lat"]  
            doubleValue],  
            [location[@"lng"] doubleValue]);  
    MyAnnotation *annotation = [[MyAnnotation alloc] init];  
    annotation.coordinate = annotationCoordinate;  
    annotation.title = location[@"name"];  
    annotation.subtitle = nil;  
    [self.mapView addAnnotation:annotation];  
}
```

# FIXING MISSING ANNOTATIONS

- We need the region that contains all added annotations
- we need the center and span
- then we need to create a region out of those dimensions and apply it to the mapView

# CREATE SEPARATE LAT AND LNG MUTABLE ARRAYS

```
- (void)viewDidLoad {
    // annotations have already been added here
    NSMutableArray *lats = [[NSMutableArray alloc] init];
    NSMutableArray *lngs = [[NSMutableArray alloc] init];
    for(NSDictionary *location in locations) {
        [lats addObject:[NSNumber numberWithDouble:
            [location[@"lat"] doubleValue]]];
        [lngs addObject:[NSNumber numberWithDouble:
            [location[@"lng"] doubleValue]]];
    }
}
```

- **Sort the lat/lng values in ascending order**
  - [lats sortUsingSelector:@selector(compare:)];  
[lngs  
sortUsingSelector:@selector(compare:)];
  - double smallestLat = [lats[0] doubleValue];  
double smallestLng = [lngs[0] doubleValue];  
double biggestLat = [[lats lastObject]  
doubleValue]; double biggestLng = [[lngs  
lastObject] doubleValue];

# MIDPOINT

```
CLLocationCoordinate2D  
annotationsCenter =  
CLLocationCoordinate2DMake((biggestLat + smallestLat) / 2, (biggestLng +  
smallestLng) / 2);
```

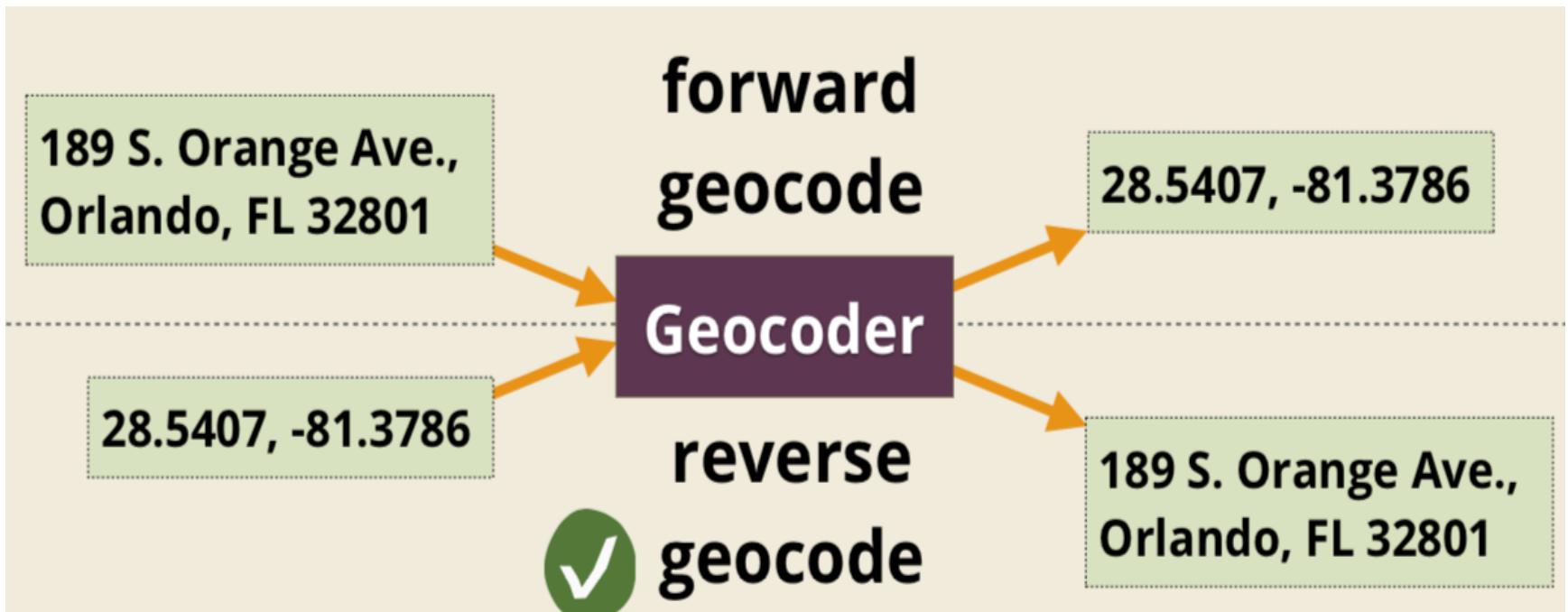
# CALCULATE THE SPAN

```
MKCoordinateSpan annotationsSpan =  
MKCoordinateSpanMake((biggestLat -  
smallestLat), (biggestLng -  
smallestLng));  
  
MKCoordinateRegion region =  
MKCoordinateRegionMake(annotationsC  
enter, annotationsSpan);  
[self.mapView setRegion:region];
```

# HOW DO WE GET A ZIP FROM A LAT/LNG?

- self.mapView.userLocation contains lat/lng
- convert from lat/lng to zip code
- SOLUTION: Geocoding

# GEOCODER



# GETTING DEVICE ZIP

```
- (void)getDeviceZip:(id)sender
{ CLGeocoder *geocoder = [[CLGeocoder
alloc] init]; pass in our current location
[geocoder
reverseGeocodeLocation:self.mapView.user
Location.location
completionHandler:^(NSArray *placemarks,
NSError *error) {
}];
}
```

# SHOW LIST VIEW

- Showing locations in a list
- Use table view