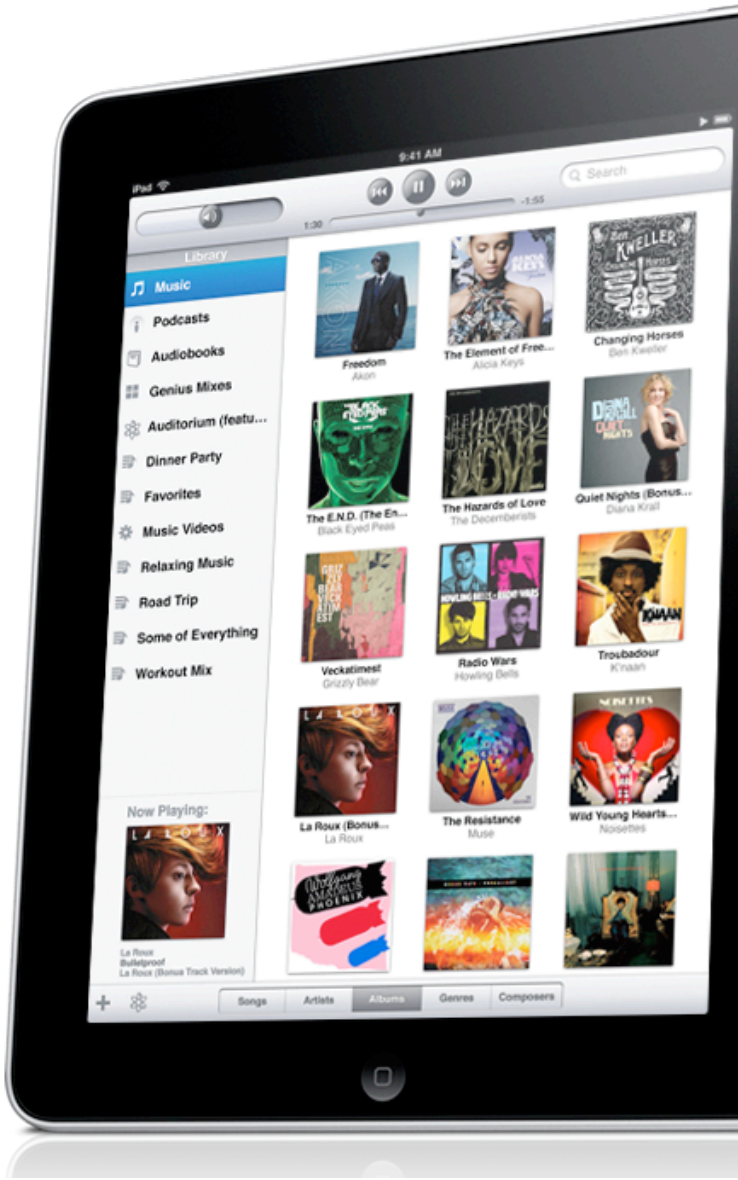

Scott Crouch
Fall 2010, CS50

# Developing Apps for iOS: iPhone, iPad and iPod Touch



# iOS 4

## Requirements:

- Mac running OS X Leopard or greater

- Enrollment in Harvard University's Student Developer Program

- Xcode 3.2.4 and iOS SDK 4.1
  Available for download from developer.apple.com

# What if I don’t have a Mac

No Worries!


CS 50 will get you access to labs with the SDK Installed.

For Access, email me at:
scrouch@college.harvard.edu

# Development Tools

Xcode 3.2.4 IDE

Interface Builder

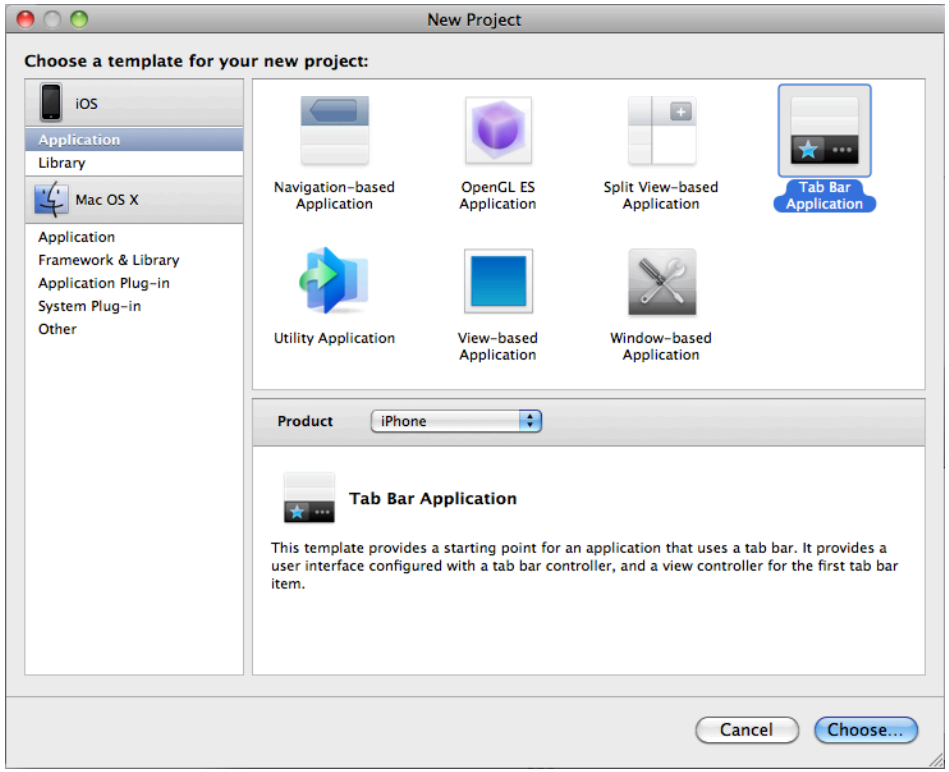
Instruments

# Introduction to Xcode 3.2.4

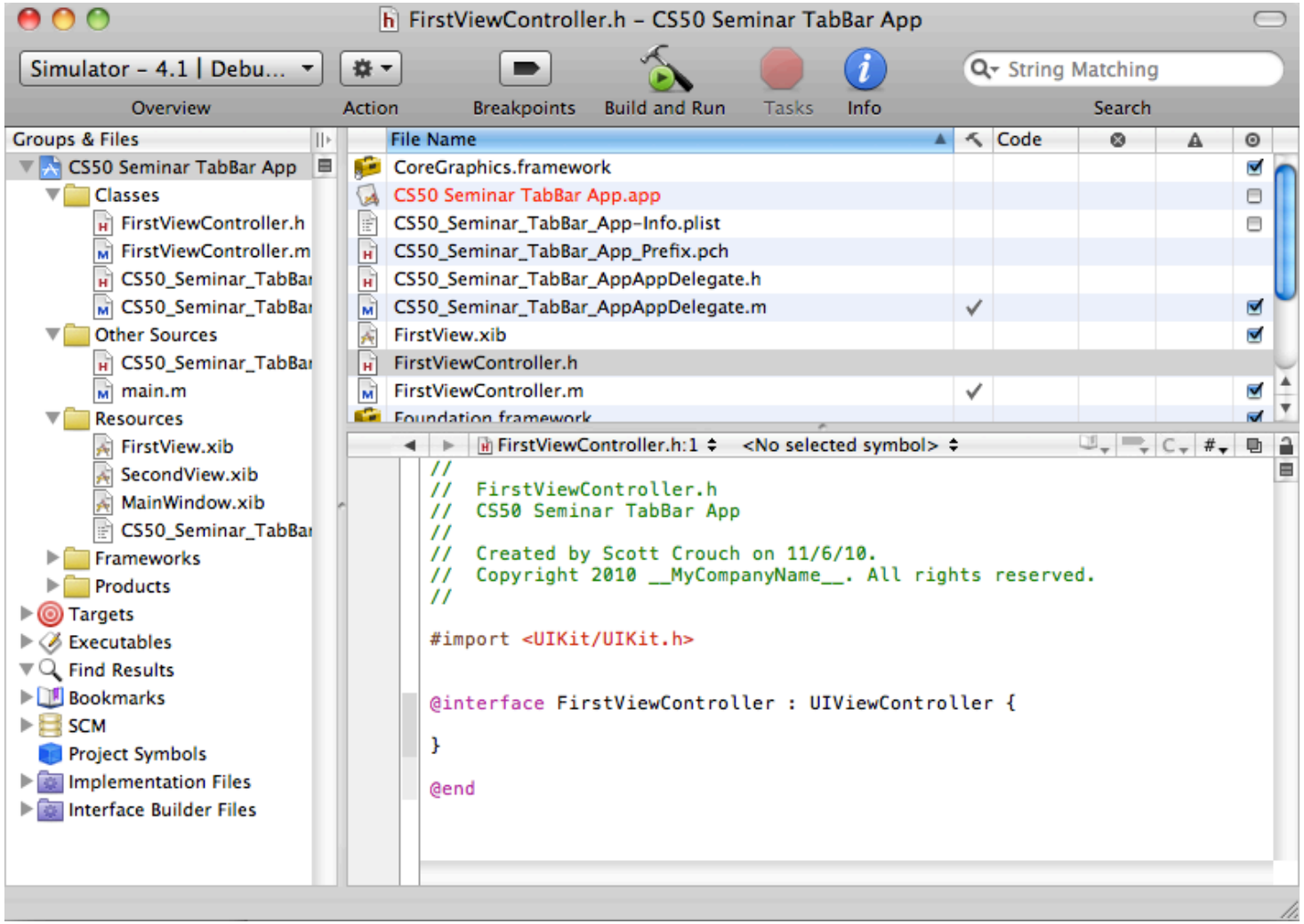
- Select the type of app you want to create

- Keep in mind the data storage you will need.

If you think you'll be using databases, it's a good idea to select the Core Data box.



# Xcode


FirstViewController.h – CS50 Seminar TabBar App
Simulator – 4.1 | Debu...
Overview
Action
Breakpoints
Build and Run
Tasks
Info
String Matching
Search
Groups & Files
CS50 Seminar TabBar App
Classes
FirstViewController.h
FirstViewController.m
CS50_Seminar_TabBar
CS50_Seminar_TabBar
Other Sources
CS50_Seminar_TabBar
main.m
Resources
FirstView.xib
SecondView.xib
MainWindow.xib
CS50_Seminar_TabBar
Frameworks
Products
Targets
Executables
Find Results
Bookmarks
SCM
Project Symbols
Implementation Files
Interface Builder Files
File Name
Code
CoreGraphics.framework
CS50 Seminar TabBar App.app
CS50_Seminar_TabBar_App-Info.plist
CS50_Seminar_TabBar_App_Prefix.pch
CS50_Seminar_TabBar_AppAppDelegate.h
CS50_Seminar_TabBar_AppAppDelegate.m
FirstView.xib
FirstViewController.h
FirstViewController.m
Foundation.framework
FirstViewController.h:1
<No selected symbol>
//
// FirstViewController.h
// CS50 Seminar TabBar App
//
// Created by Scott Crouch on 11/6/10.
// Copyright 2010 __MyCompanyName__. All rights reserved.
//
#import <UIKit/UIKit.h>
@interface FirstViewController : UIViewController {
}
@end

# Xcode Basics

- Breakpoints
- Creating new files
- Changing the project settings
- Changing your base SDK
- Configuration
  - Debug
  - Distribution
  - Release

# Debugging in Xcode

- With break points, we can debug our code!
- Debug Console
- Also, NSLog statements
  - `NSLog("We just reached line 48,259!");`

# Using Instruments to Analyze Your App

- We can use instruments to look at memory usage (RAM) , find memory leaks and analyze our graphics component.

# Objective C - 2.1
# Cocoa Touch

# Keywords

- Springboard
  - The main home screen *App* of iOS which controls the launching and terminating of Apps.

- Cocoa Touch
  - The programming interface that handles the user interaction and multi-touch events

# Model-View-Controller Paradigm

Every Application has views, these views are controlled by view controllers.

# .m and .h files

- .h files are header files are for:
  - Declaration of objects that need to be synthesized
  - @property directives for objects that need to synthesized.
  - Declaration of Function Prototypes
- .m files are for:
  - Synthesis of object variables
  - Implementation of pre-defined methods and custom methods.

# Basic Data Types in Objective C

int: standard integer

float: store decimal numbers

double: store decimal numbers > float

char: standard character

BOOL: TRUE/FALSE or YES/NO

id: since objective-c is object oriented, we need a way to reference an object. Each object has an id.

# Declaring Objects in Objective – C in .h files

```
#import <UIKit/UIKit.h>
```

Class Name — Parent Class Name

```
@interface MainViewController : UIViewController {

    UIButton *button1;
    UIImageView *imageView;

    CGFloat tapLocation;
    CGSize originalImageSize;
    CGSize newImageSize;

    Boolean rotated;

}
```

# @property directive

```
@property(nonatomic, retain) UIButton *button1;
@property(nonatomic, retain) UIImageView *imageView;

@property(nonatomic, readwrite) CGFloat tapLocation;
@property(nonatomic, readwrite) CGSize originalImageSize;
@property(nonatomic, readwrite) CGSize newImageSize;

@property(nonatomic) Boolean rotated;


@end
```

Syntax: @property ( <propertyname1>, <propertyname2>, etc.) <Object Type> <Object Name>;

# nonatomic, atomic, retain and readwrite properies

- nonatomic:  most widely used property, this give the application complete access to this this object.
- atomic:  sort of like nonatomic except there are restrictions in place to prevent against simultaneous updates.
- retain:  very similar to malloc
- readwrite:  allows accessing and mutating of an object

# Some Tricks for @property

1. All UI Elements should be under the @property(nonatomic, retain) directive

2. Objects like CGSizes, CGFloats, NSNumbers (which are all numbers) should be under the @property (nonatomic, readwrite) or @property(nonatomic, readonly) directive

3. Booleans should just be under the @property(nonatomic) directive

4. Objects that you want to copy should be created under the @property (nonatomic, copy) directive

# Synthesizing Objects in .m files

```
#import "MainViewController.h"

@implementation MainViewController

@synthesize button1,
            imageView,
            tapLocation,
            originalImageSize,
            newImageSize,
            rotated;
```

# Methods in Objective-C
# One Parameter

Syntax:

- (<return type>) <method name>: (<parameter type>) <parameter name>

Example:

- (CGSize) getSize: (UIImageView *) imageView {}

# Methods in Objective-C
# Two or More Parameters

Syntax:

- (<return type>) <method name 1>: (<parameter 1 type>) <parameter 1 name>  <method name 2>: (<parameter 2 type> <parameter 2 name>

Example:

```
- (NSInteger)tableView:(MGExpandingTableView *)tableView numberOfChildRowsInSection:(NSInteger)section {}
```

# Calling Methods

Syntax:

[<object> <method name>: <parameter>];

Example:

```
[self.navigationController pushViewController:
nextViewController animated:YES];
```

# Built in Methods

-(void) loadView {}:

Create a view, add UI elements without an Interface Builder File (nib)

-(void) viewDidLoad {}:

Add elements to a view after the view has been loaded. Usually used with a nib but not always.

# Adding a UI Element

UIButton FTW!

```
UIButton *cs50Button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[cs50Button setBackgroundImage:[UIImage imageNamed:@"testImage.png"]
                      forState:UIControlStateNormal];
cs50Button.frame = CGRectMake(10, 10, 90, 90);
[cs50Button addTarget: self
               action: @selector(launchCS50ButtonViewController:)
     forControlEvents: UIControlEventTouchUpInside]

[self.view addSubview: cs50Button];
```

# Adding a UI Element

Let's now try adding a scroll view!

```
UIScrollView *cs50ScrollView = [[UIScrollView alloc]
                                initWithFrame: CGRectMake(0,0,320, 480)];

cs50ScrollView.contentSize = CGSizeMake(320, 1000);
cs50ScrollView.scrollsToTop = YES;
cs50ScrollView.bounces = YES;
cs50ScrollView.zooming = YES;
cs50ScrollView.maximumZoomScale = 2.0;
cs50ScrollView.minimumZoomScale = 0.5;

[self.view addSubview:scrollView];
```

# Creating a UI with Interface Builder

# Interface Builder

Allows you to drag and drop UI elements

Very inflexible: Once you have placed the element in IB, there is no way to override it in code.

Hard-coding UI Elements is better style and allows full customization.

# IB Outlets

When you drag and drop an item in interface builder, you need to connect it to an outlet in one of your view controllers.

In the .h file:

```
IBOutlet UIButton *button;
```

```
@property (nonatomic, retain) IBOutlet UIButton *button;
```

# IBActions

Sort of like IBOutlets except for methods. These need to be connected as well.

In the .h file:

```
- (IBAction) hitTheButton;
```

In the .m file:

```
    your method implementation
```

HelloWorldXIB.xcodeproj

# Memory Management

- You have to do this yourself
- If you alloc or retain an object like a button or an image view, you must release it in the dealloc function.
  - [<object name> release];

  iOS has a flaw! The counter retainCount which is built in is unreliable and will not report correct values.

# Data Structures and Core Data

# Data Storage

Just like in C, you have Arrays (well an NSArray)

You also have NSSet, NSDictionary and NSMutableArray.

```
NSArray *myArray = [[NSArray alloc]
  initWithObjects: @”Hello”, @”My”,
  @”Name”, @“Is”, @”Scott”];
```

# Core Data

- If you're using a data base (most likely SQLite), you'll need the Core Data wrap around.

- Revolves around three concepts
  - Managed Object Model
  - Managed Object Context
  - Persistent Store Coordinator

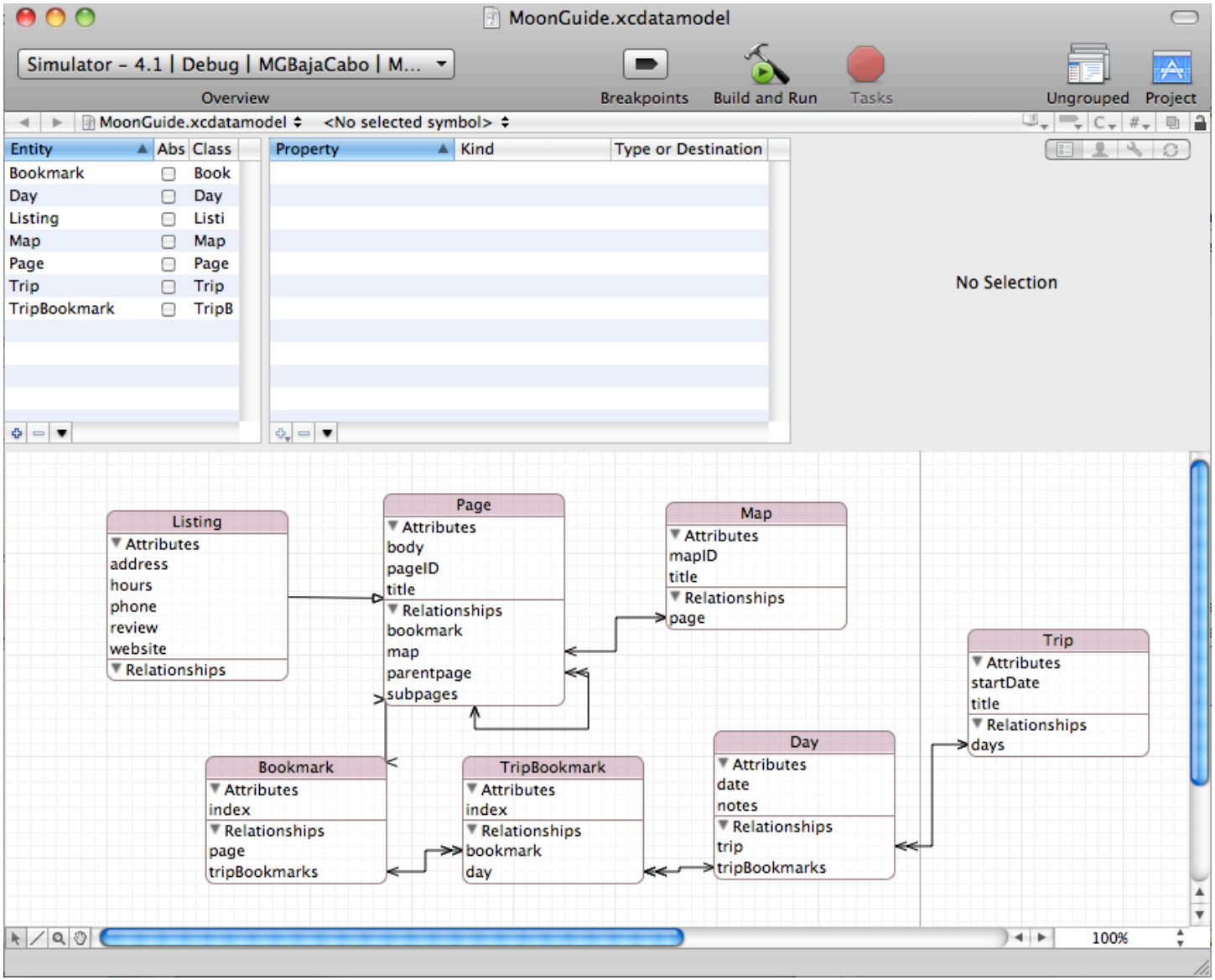
# Managed Object Model

Think of this as a database schema:

Your objects are known as *Entities* and these entities have information called *Attributes*

To set up your Managed Object Model, you will create a .xcdatamodelfile

# .xcdatamodel



# Managed Object Context

- Most Important of the three!
- We will call methods on the Managed Object Context to access information from our database

```
-(NSManagedObjectContext *)managedObjectContext {

    if (managedObjectContext != nil) {
        return managedObjectContext;
    }

    MGAppDelegate *appDelegate = (MGAppDelegate *)[[UIApplication sharedApplication] delegate];
    managedObjectContext = [appDelegate managedObjectContext];

    return managedObjectContext;
}
```

# Saving with Managed Object Context

Presuming you have modified/added an object and it corresponds to your database, you just have to enter these lines to save the state of that object.

```
NSError *error = nil;
if (![self.managedObjectContext save:&error]) {
    NSLog(@"Error saving after editing Trip: %@, %@", error, [error userInfo]);
}
```

# Persistent Store Coordinator

Think of it as the guy who opens the apple store every morning, he doesn't sell the iPhones, but the store has to be open for them to be sold.

This sets up the names and locations of the databases. Any Managed Object Context that saves an object goes through the Persistent Store Coordinator.

# The App Delegate

- The App Delegate is a class in EVERY iPhone App.  It handles the launching of the app.

  The Managed Object Model, Context and Persistent Store Coordinator need to be set up here.


# Building a Great App

# The Basics of an App

- Most every app (though not all) has the following:

  - Tab Bar
  - Navigation Bar
  - Status Bar

# The Tab Bar

- Adds Stability and Organization
  - Width: 360 pts Height: 49 pts
- Classed under the
  - UITabBarController Class

  Properties:
  viewControllers (NSArray)
  customizabeViewControllers (NSArray)
  delegate
  moreNavigationController
  selectedViewController

# Creating a Tab Bar Application

You can do this two ways, with by selecting a Tab Bar App at the start of Xcode. Or by hard-coding it in yourself. Should be done in App Delegate!

```
UITabBarController *tabBar = [[UITabBarController alloc] init];
tabBar.frame = CGRectMake(320, 0, 320, 49);

NSArray *viewControllers = [NSArray arrayWithObjects: vC1, vC2, nil];
[tabBar setViewControllers:viewControllers];

[window addSubview: [tabBar view]];
```

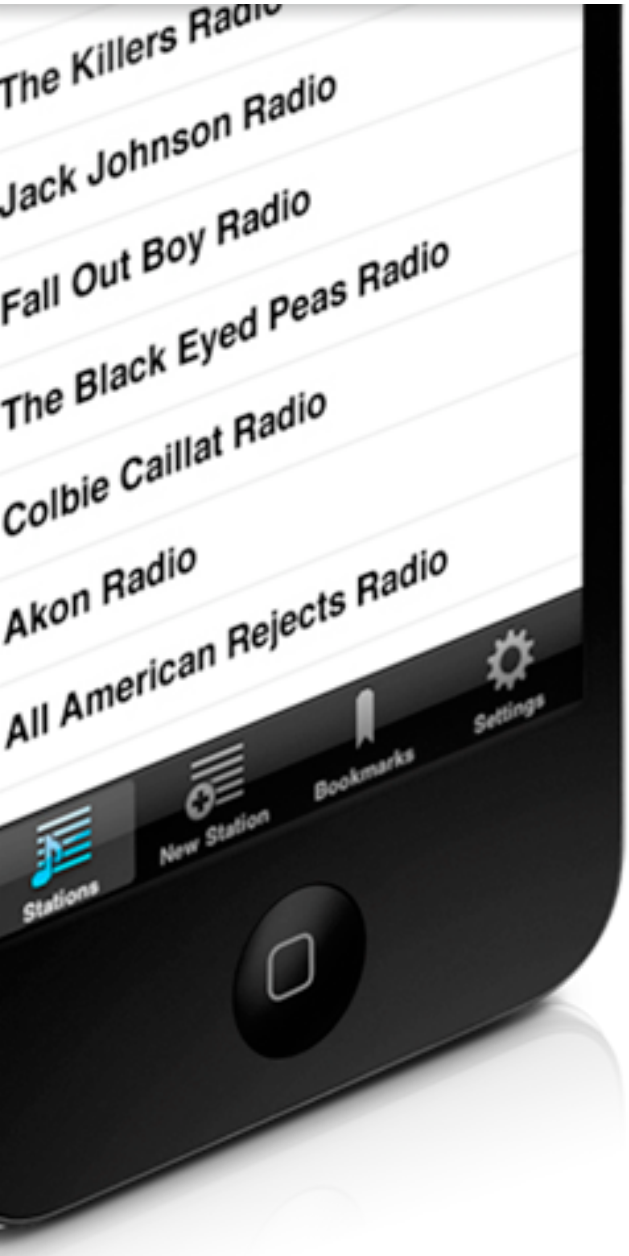

# The Navigation Bar

• Adds hierarchical navigation to your app.
  - Width:  360 pts  Height: 44 pts

•Stores View Controllers on the stack

•Classed under the:
  -UINavigationController

  Properties:
    navigationBar
    viewControllers (NSArray)
    topViewController
    visibleViewController




# Table View Apps

Belongs to the UITableViewController class

Can be customized for height, width, images and multitudes of text.

Can be created by clicking *Table View App* when Xcode launches or by hard coding a table.

# Hard-Coding a Table View App

Let's Start in the .h file!

```
@interface CS50TableViewController : UITableViewController
        <UITableViewDelegate,
        UItableViewDataSource>  {

            UITableView *tableView;

}

@property (nonatomic, retain) UITableView *tableView;
```

# Table View in the .m file

A few methods need to be implemented here, and I'll leave it up to you guys to figure out how to implement them

# Table View Methods to be Implemented

1. -(NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
2. -(NSInteger)tableView (UITableView *)tableViewnumberOfChildRowsInSection:(NSInteger)section
3. - (UITableViewCell *)tableView:(UITableView*)aTableView childCellForRowAtIndexPath:(NSIndexPath *)indexPath
4. – (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath

# ModalViewControllers

Basically popup controllers! They can have anything in them (tableview, etc.)

```
[self presentModalViewController: <view controller name> animated: <boolean>];
```

```
[self dismissModalViewController: <view controller name> animated: <boolean>];
```

# UIGestureRecognizers

This is an awesome class! Allows for all sorts of cool multi-touch events.

Example: Swipe-Left

```
- (IBAction)handleSwipeLeft:(UIGestureRecognizer
*)sender{}
```

# Handling Input from Sensors

Accelerometer:

```
- (void) accelerometer: (UIAccelerometer
  *)accelerometer didAccelerate: (UIAcceleration
  *)acceleration {}
```

Camera:

Try creating a UIImagePickerController in a modalViewController.

# Developing for iPad

Awesome!  iPad development is a lot of fun.  All of the same code, view controllers and schemes apply.  Although we have a new type of application

The Split View Controller App:

# Developing Games

Open GL ES


Core Animation

Core Audio

Quartz Composer

# Finally, Some tips

Make it intuitive!

- No one wants to see a thousand buttons or a very convoluted hierarchy

Have one clear goal for your app!

- Don’t get bogged down in making it do too many different things.

# Thanks for Coming!


Images credit of apple.com, hopefully they don't mind I'm using them