

# INTRODUCTION TO IOS

CSCI 4448/5448: OBJECT-ORIENTED ANALYSIS & DESIGN

LECTURE 13 — 02/22/2011

# Goals of the Lecture

- Present an introduction to iOS Program
- Coverage of the language will be INCOMPLETE
  - We'll see the basics... there is a lot more to learn

# History

- ◆ iOS is the name (as of version 4.0) of Apple's platform for mobile applications
  - ◆ The iPhone was released in the summer of 2007
  - ◆ According to Wikipedia, it has been updated 30 times since then, with the 31st update slated to occur when version 4.3 is released soon (currently in beta)
  - ◆ iOS apps can be developed for iPhone, iPod Touch and iPad; iOS is used to run Apple TV but apps are not currently supported for that program

# iOS 4.2

- I'll be covering iOS 4.2 which is the current "official version"
  - Version 4.2 in November 2010 was significant in that it represented the first time that the same OS ran across all three hardware platforms
  - Until that release iPad had been running version 3.2 which had been created specifically for that device

# Acquiring the Software

- ◆ To get the software required to develop in iOS
  - ◆ Go to <http://developer.apple.com/>
  - ◆ Click on iOS Dev Center
  - ◆ Log in (free registration is required)
  - ◆ Download the release called
    - ◆ Xcode 3.2.5 and iOS SDK 4.3
  - ◆ Double click the .dmg and run the installer

# Tools

- ◆ Xcode: Integrated Development Environment
  - ◆ Provides multiple iOS application templates
- ◆ Interface Builder
  - ◆ Provides drag-and-drop creation of user interfaces
- ◆ iPhone Simulator
  - ◆ Provides ability to test your software on iPhone & iPad
- ◆ Instruments: Profile your application at runtime

# iOS Platform (I)

- ◆ The iOS platform is made up of several layers
  - ◆ The bottom layer is the Core OS
    - ◆ OS X Kernel, Mach 3.0, BSD, Sockets, File System, ...
  - ◆ The next layer up is Core Services
    - ◆ Collections, Address Book, SQLite, Networking, Core Location, Threading, Preferences, ...

# iOS Platform (II)

- ◆ The iOS platform is made up of several layers
  - ◆ The third layer is the Media layer
    - ◆ Core Audio, OpenGL and OpenGL ES, Video and Image support, PDF, Quartz, Core Animation
  - ◆ The final layer is Cocoa Touch (Foundation/UIKit)
    - ◆ Views, Controllers, Multi-Touch events and controls, accelerometer, alerts, web views, etc.
- ◆ An app can be written using only layer 4 but advanced apps can touch all four layers

# Introduction to Interface Builder (I)

- ◆ Interface Builder is extremely powerful
  - ◆ It provides a drag and drop interface for constructing the graphical user interface of your apps
  - ◆ The GUIs created by Interface Builder are **actual instances** of the **underlying UIKit classes**
    - ◆ When you save an Interface Builder file, you “freeze dry” the objects and store them on the file system
    - ◆ When your app runs, the objects get reconstituted and linked to your application logic

# Introduction to Interface Builder (II)

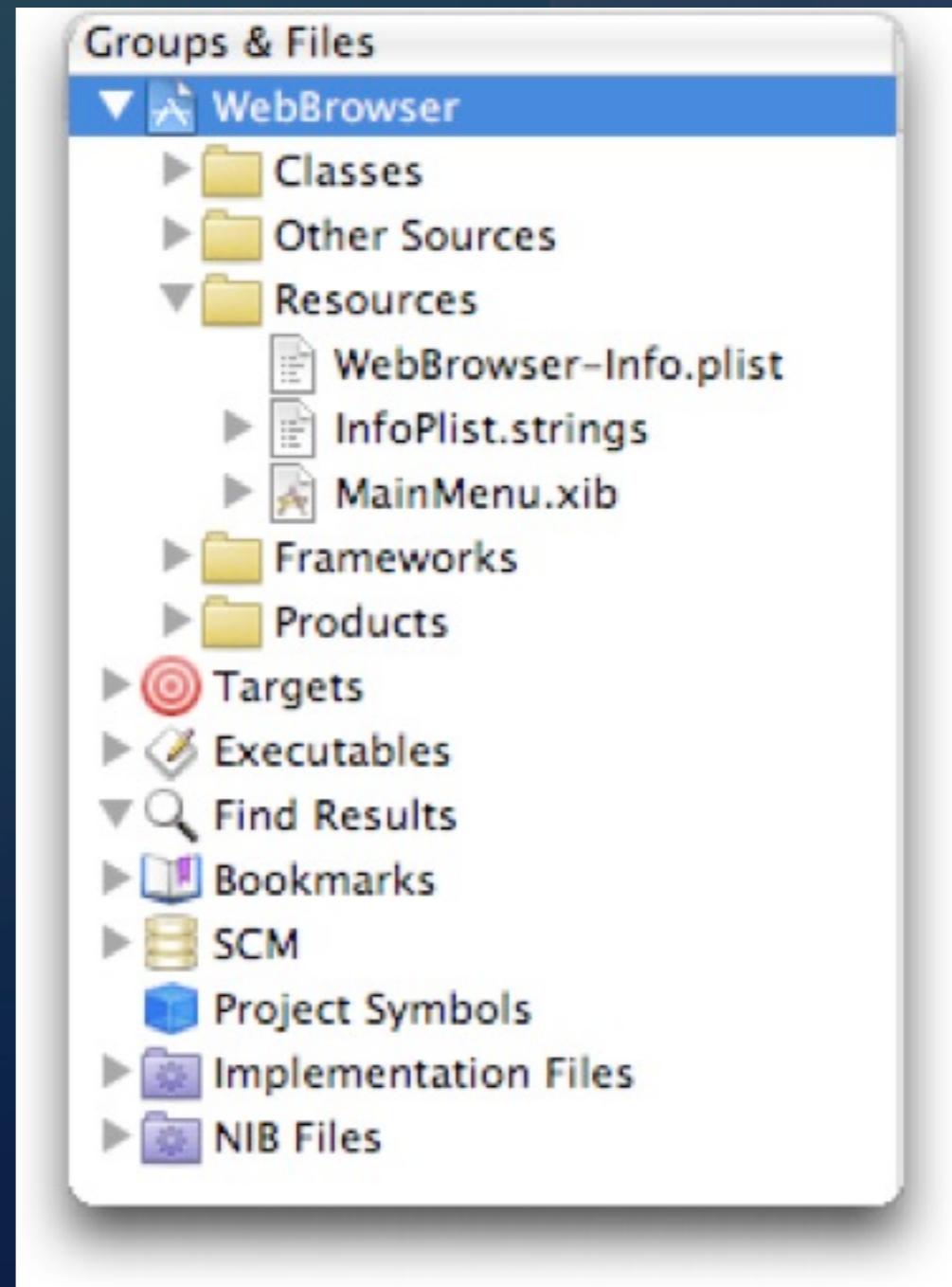
- This object-based approach to UI creation is what provides interface builder its power
  - To demonstrate the power of Interface Builder, let's create a simple web browser without writing a single line of code
  - The fact that we can do this is testament to the power of object-oriented techniques in general

# Step One: Create Project

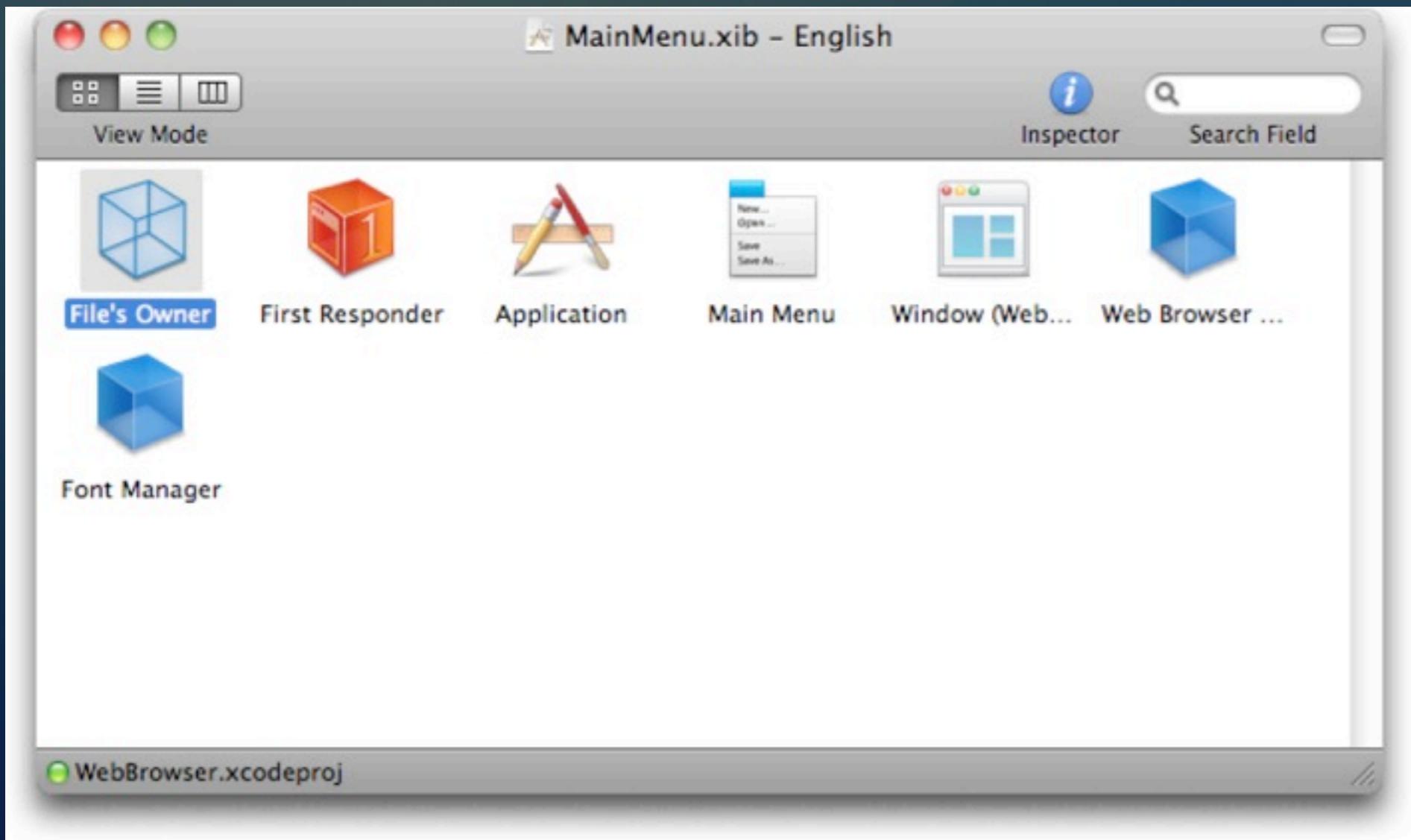
- Launch XCode and create a MacOS X application (we'll get to iOS in a minute)
  - ◆ Unlike last time, select Cocoa Application rather than Command Line Tool
  - ◆ Name the project WebBrowser and save it to disk
    - ◆ Click Build and Run to see that the default template produces a running application
    - ◆ It doesn't do anything but create a blank window

# Step Two: Launch IB

- ◆ Expand Resources
  - ◆ Double click MainMenu.xib
    - ◆ xib stands for “XML Interface Builder”; it is an XML file that stores the freeze dried objects that interface builder creates; You will never edit it directly
  - ◆ Interface Builder will launch



# Interface Builder UI

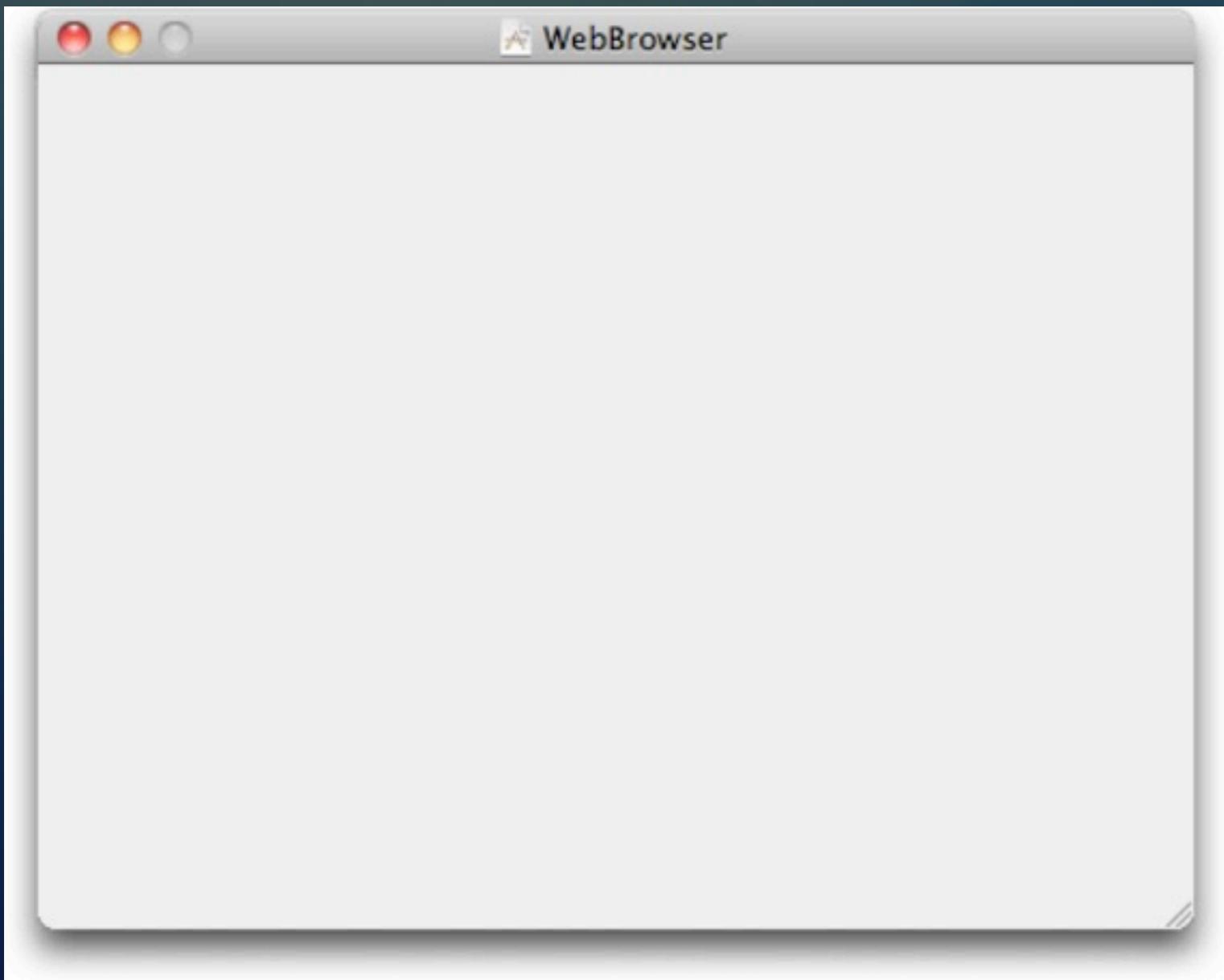


**Shows the objects instantiated for our default project**

**Window and Menu represent actual instances of `NSWindow` and `NSMenu`**

**You can establish connections between the objects in Interface Builder**

# Our default Window



**This is the window you saw when you ran the application back in Step One;**

**Exciting, isn't it**

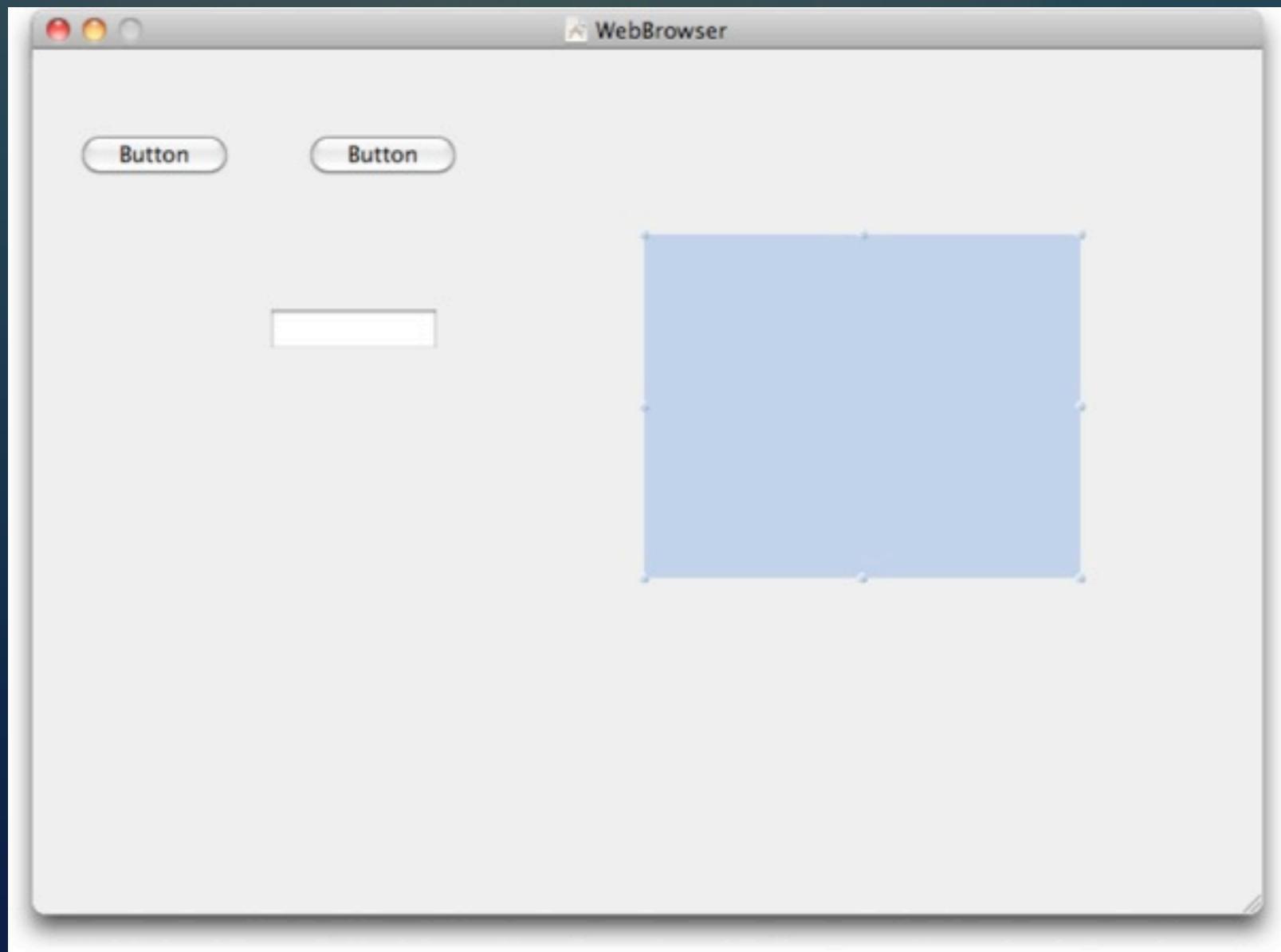
**Try changing its size, save the document, switch back to Xcode and run the program**

**You'll see your changes reflected**

# Step 3: Acquire Widgets

- ◆ Invoke Tools ⇒ Library to bring up the widgets that can be dragged and dropped onto our window
- ◆ Type button in the search field and then drag two “push buttons” on to the window
  - ◆ It doesn’t matter where you drag them just yet
- ◆ Type text field in the search field and then drag a “text field” on to the window (ignore “text field cell”)
- ◆ Type “web” and drag a “web view” to the window

# Results of Step 3



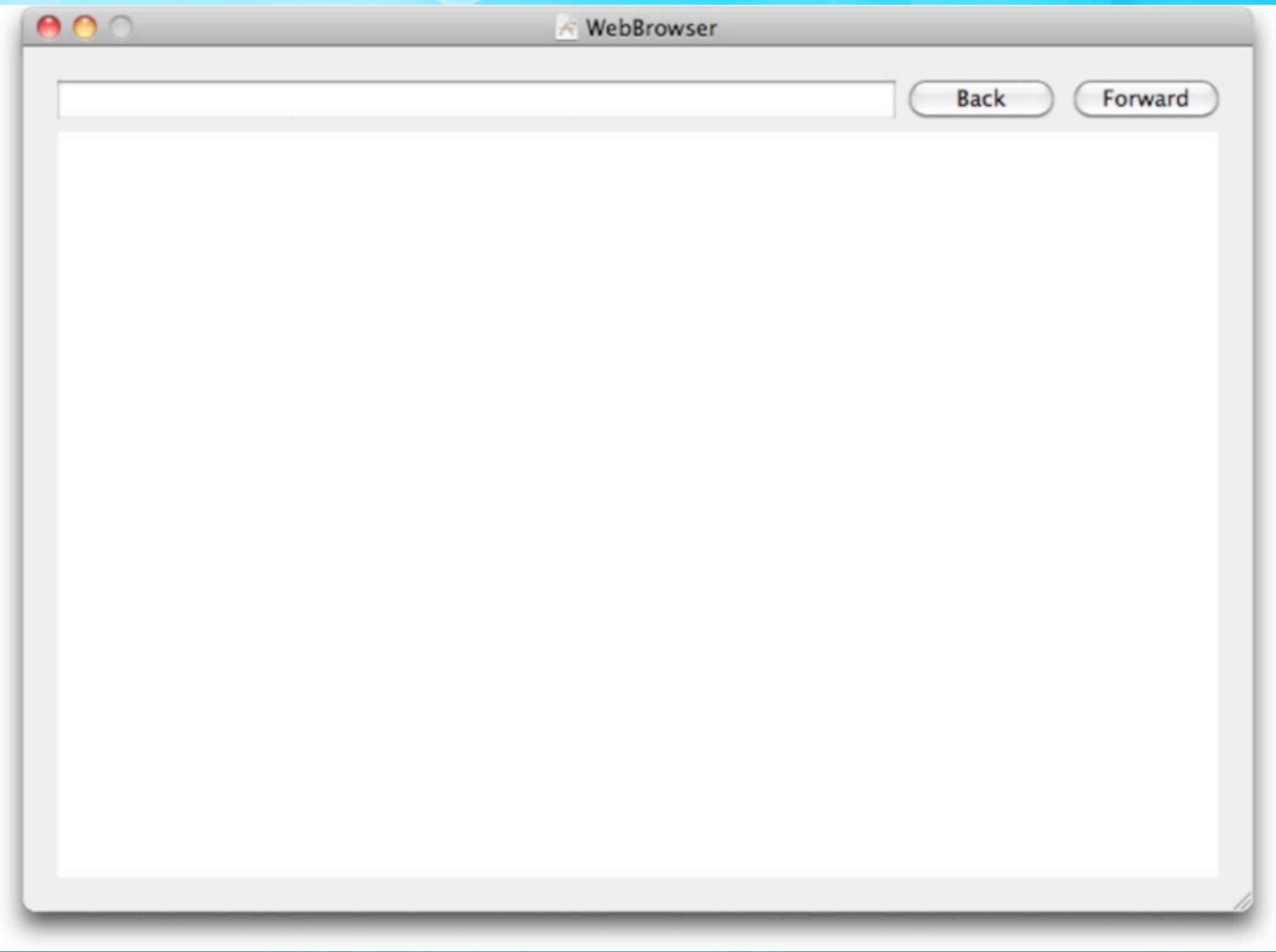
**Window now has four widgets but they are not yet placed where we want them**

# Step 4: Layout Widgets (I)

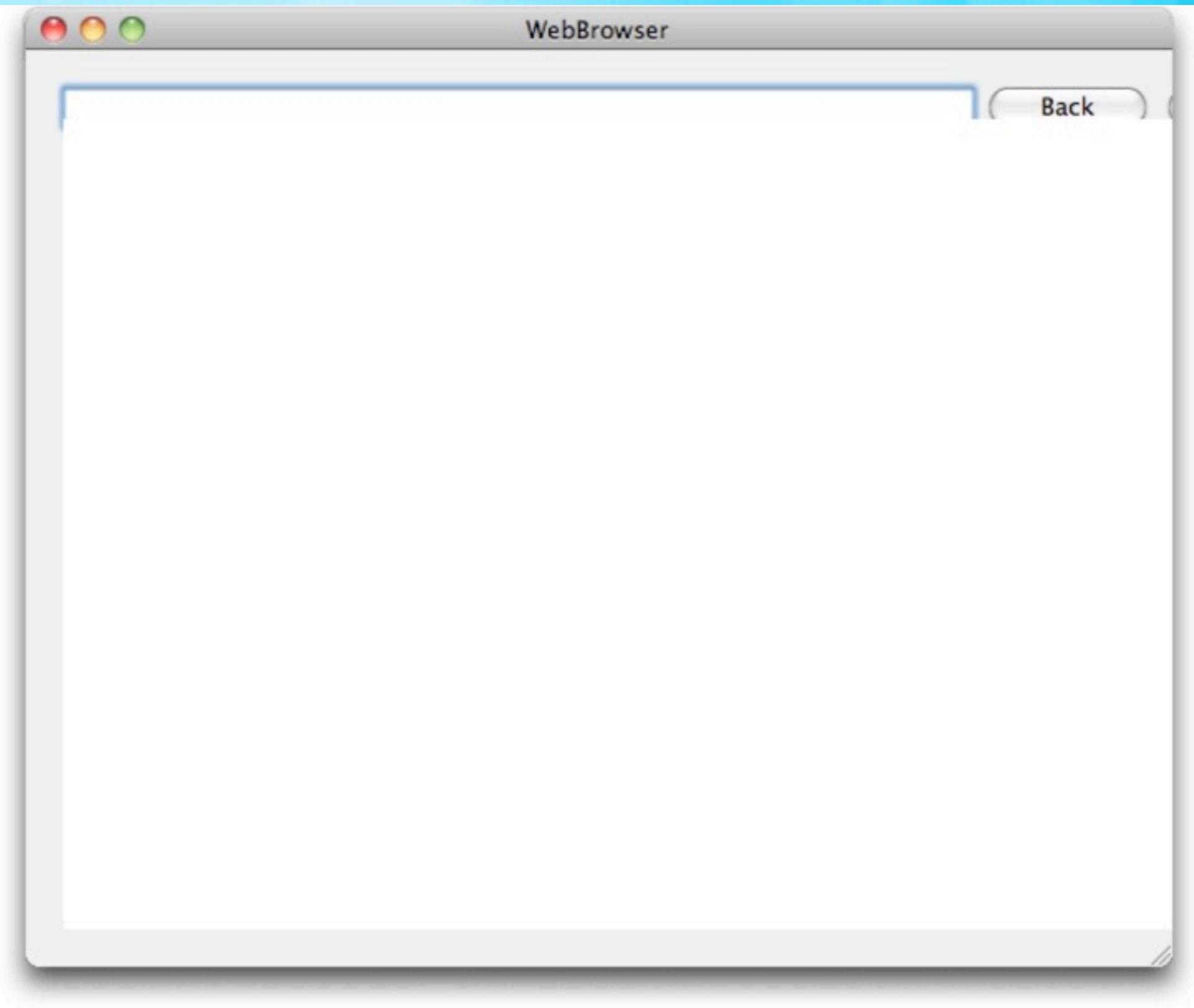
- ◆ Put the buttons in the upper right corner
  - ◆ Use the guides to space them correctly
  - ◆ Double click on them and name one “Back” and one “Forward”
- ◆ Put the text field in the upper left corner and stretch it out so it ends up next to the buttons
  - ◆ Again use the guides to get the spacing right
  - ◆ These guides help you follow Apple’s human interface guidelines

# Step 4: Layout Widgets (II)

- ◆ Expand the Web view so that it now fills the rest of the window, following the guides to leave the appropriate amount of space
  - ◆ Your window now looks like the image on the next slide

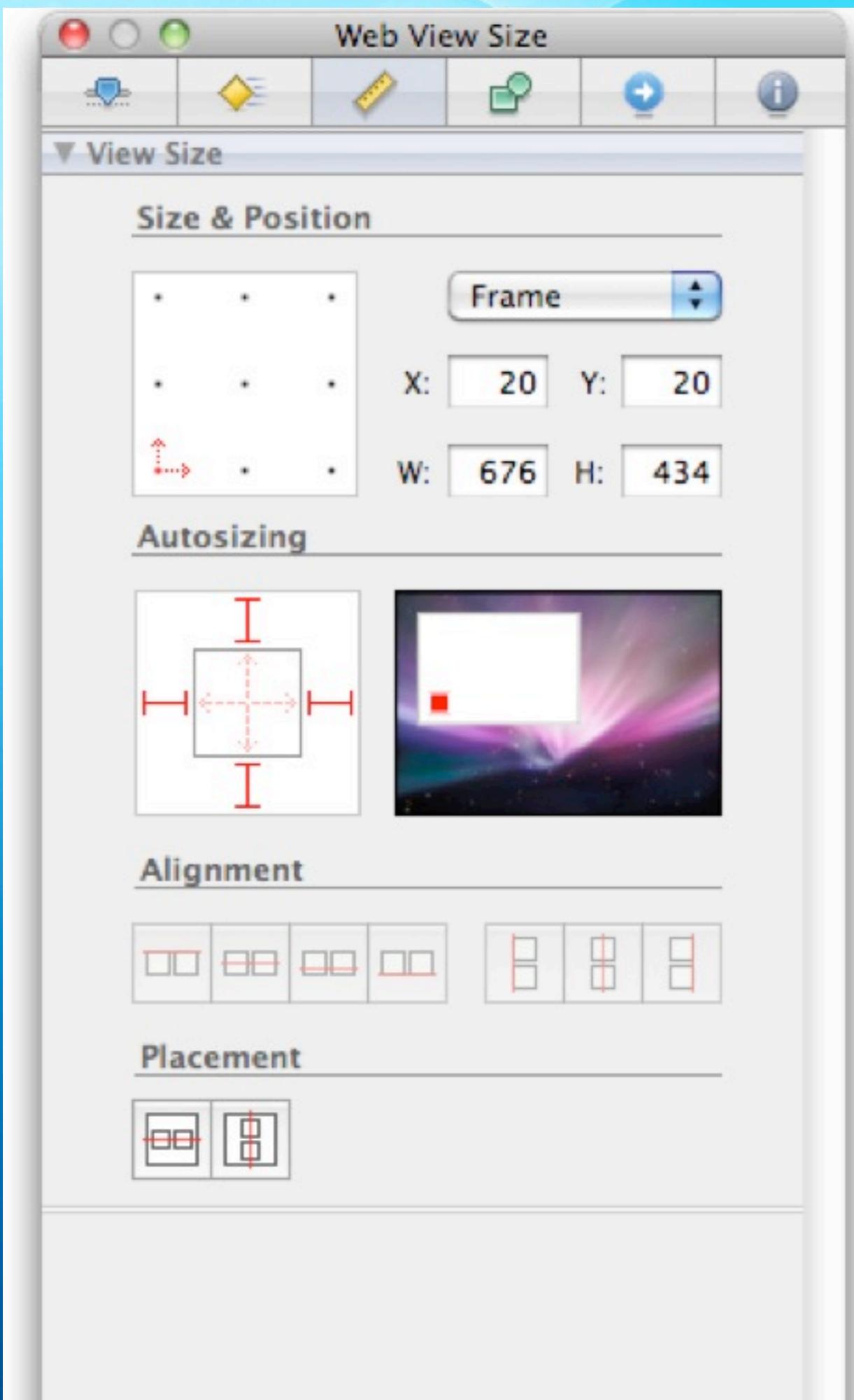


**Everything is fine until you try to resize the window. Click ⌘-R and try it out**



**Whoops!**

**Fortunately,  
Interface Builder  
makes it easy to  
specify constraints  
on how widgets  
should behave  
when resize events  
occur**



**With the Web View selected, click ⌘-3 or select Tools ⇒ Size Inspector**

**The Autosizing section provides the ability to specify resizing constraints**

**The outside brackets indicate whether a widget should try to remain relative to a particular side of the window during a resize event**

**The internal arrows (currently deselected) indicate whether a widget should grow horizontally or vertically during a resize event**

**For web view, we want all four brackets on (stay locked in place) and both arrows on (grow to fill all available space)**

**Select the window and be sure to set your window's minimum size too**

# Finish specifying autosizing behavior

- ◆ The two buttons need to be anchored on top and on the right hand side; they should not resize themselves during a window resize event
- ◆ The text field should be anchored on top, left and right; it should resize horizontally during a window resize event
- ◆ With these changes, your window should behave as expected during a resize event
- ◆ Without writing a single line of code!

# Step 5: Make Connections (I)

- We want to establish connections between the various widgets we've created
  - With Interface Builder, you do this via “Control Drags”
    - You hold down the control key, click on a widget and hold, and then finally drag to another widget
    - A menu will pop-up allowing you to specify a connection

# Step 5: Make Connections (II)

- ◆ Establish the following connections
  - ◆ From Text Field to Web View: **takeStringURLFrom:**
  - ◆ From Back Button to Web View: **goBack:**
  - ◆ From Forward Button to Web View: **goForward:**
  - ◆ Note the colon symbol at the end of these names: ":"
    - ◆ these are Objective-C method names!
    - ◆ they are methods defined by Web View
    - ◆ they will be invoked when the source widget is triggered

# Step 6: Link the Framework

- ➊ Save your Interface Builder project
- ➋ Switch back to XCode
- ➌ Expand the Targets folder and right click on WebBrowser
  - ➍ Select “Get Info” in the pop-up menu
- ➎ Under the Linked Libraries pane, click +
  - ➏ In the resulting window, click WebKit.framework and click Add

# Step 7: Run the App; Browse the Web

- ➊ Type a URL and click Return
  - ➋ Watch the page load
- ➋ Load another page
- ➌ Click the Back button
- ➍ Click the Forward button
- ➎ A simple web browser without writing a single line of code

# Discussion

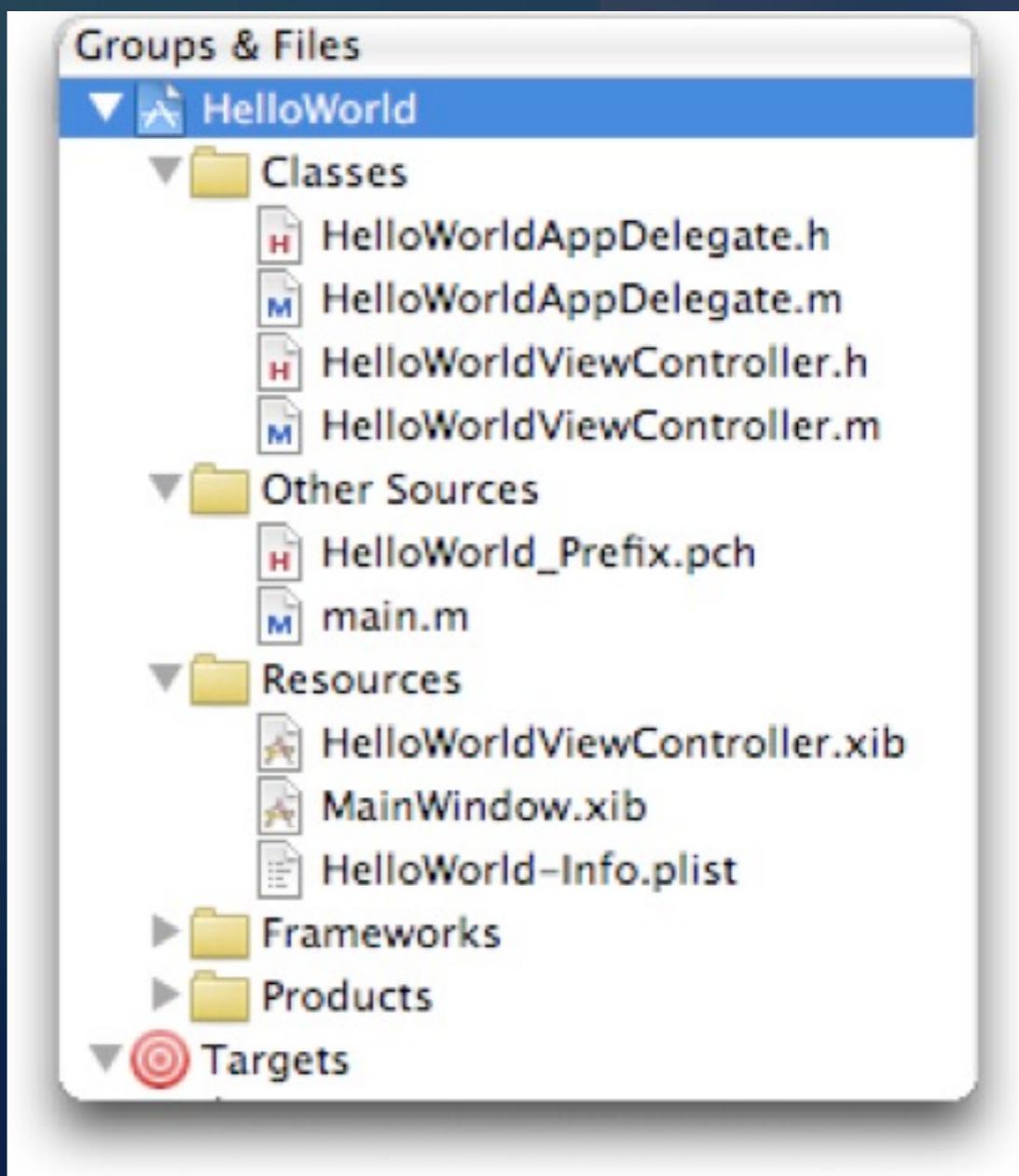
- This example is relevant to iOS programming because it shows all of the major mechanics of Interface Builder
  - We'll see a few more things Interface Builder can do when we link up code in XCode to widgets in Interface Builder
- Example demonstrates the power of objects; WebView is simply an instance of a very powerful object that makes use of Apple's open source WebKit framework
  - We can establish connections to it and invoke methods

# Let's create a Hello World iOS App

- ◆ Select New Project from the File Menu
  - ◆ Click Application under iOS
  - ◆ Click View-based Application
  - ◆ Select iPhone
  - ◆ Click Choose and Name the App HelloWorld
  - ◆ Save the App
  - ◆ The project window opens

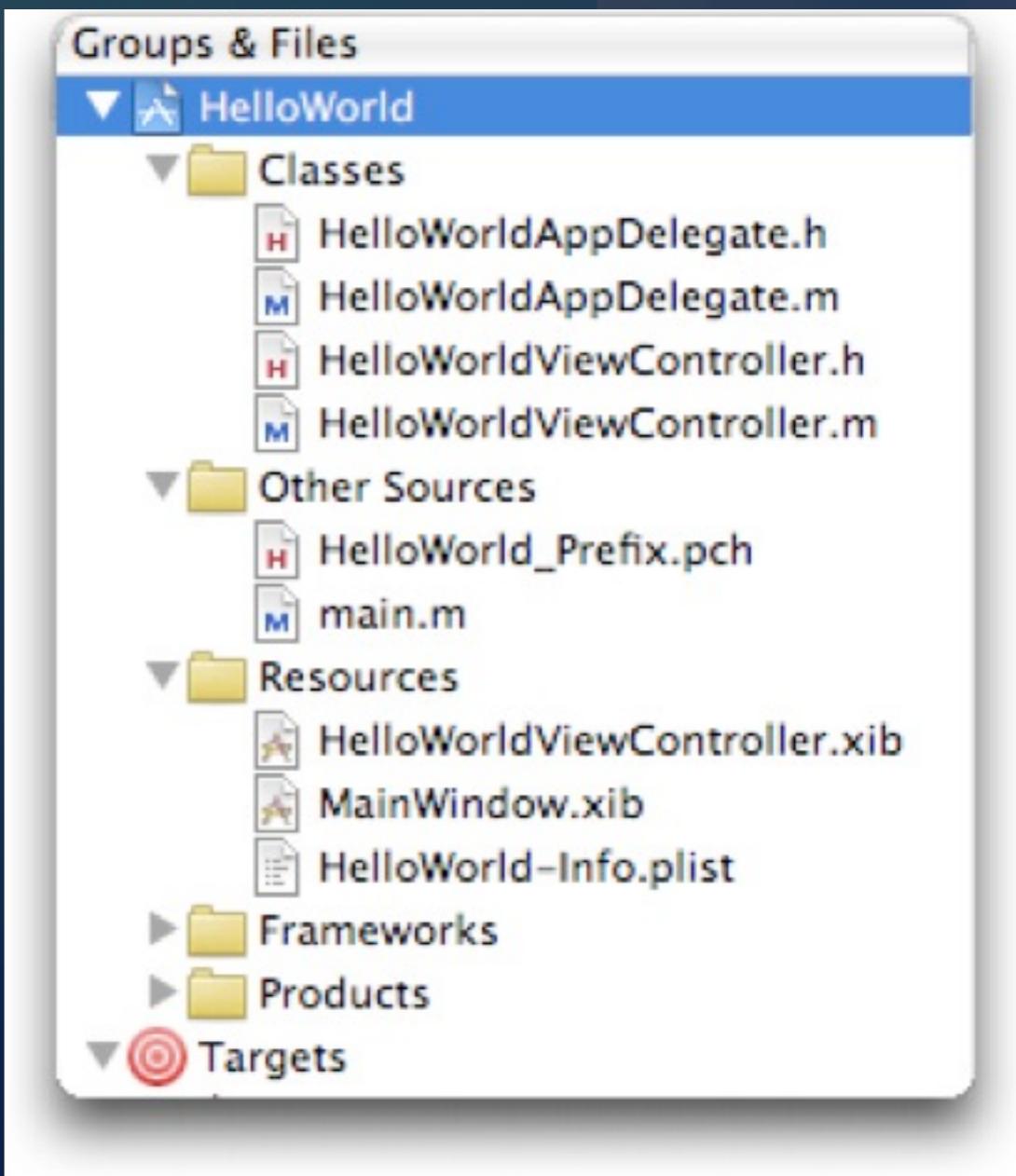
# iOS Project Structure

- ◆ All iOS apps are instances of a class called UIApplication
  - ◆ You will never create an instance of that class
  - ◆ Instead, your application has an “AppDelegate” that is associated with UIApplication
    - ◆ The former will call your AppDelegate at various points in the application life cycle



# iOS Project Structure

- Since this is a view-based application, an instance of a class called UIViewController was also created for you
- The application has a .xib file called MainWindow; the view controller has one too, it's called HelloWorldViewController.xib
- How is this all connected?



# iOS Project Structure

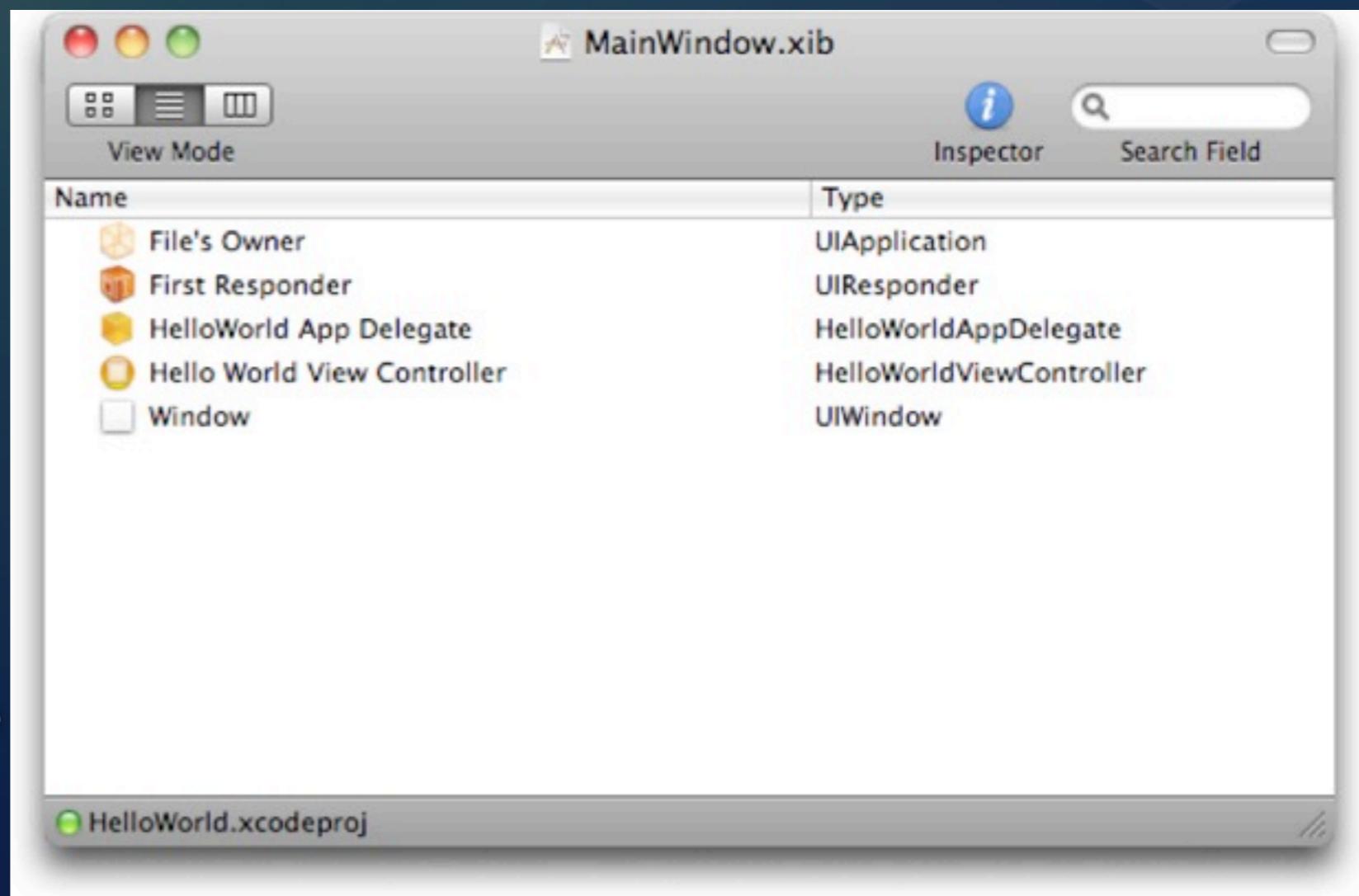
- ◆ The main.m file is straightforward
  - ◆ It creates an autorelease pool
  - ◆ Invokes UIApplicationMain()
    - ◆ This program reads in the .xib files, links up all the objects, loads the view controller, and starts the event loop, where it will remain until the application is told to shutdown
    - ◆ It then deallocates the pool and returns

# iOS Project Structure

- ◆ Double click **MainWindow.xib**

**Your app starts with three objects created by the MainWindow.xib file: Hello World App Delegate, Hello World View Controller and Window**

**No need to create them in code, they will be created automatically when this file is read by UIApplication**



# iOS Project Structure

- ◆ Bring up the Connections Inspector ⌘-2

This picture shows that not only does the .xib file create the Hello World App Delegate, it also connects it to the Hello World View Controller object via the “viewController” attribute and the Window via the “window” attribute



Take a look at the code to see these attributes defined.

We also see that UIApplication (File's Owner) is wired to the App Delegate

# iOS Project Structure

- ➊ Close the MainWindow.xib file without modifying it
- ➋ Back in XCode, double click the HelloWorldViewController.xib file
  - ➌ Here we see that this file creates an instance of UIView
- ➍ When MainWindow.xib creates the Hello World View Controller object, that object will, in turn, load its .xib file causing this view object to be created
  - ➎ The connection settings view shows that this view will then be connected to the view controller

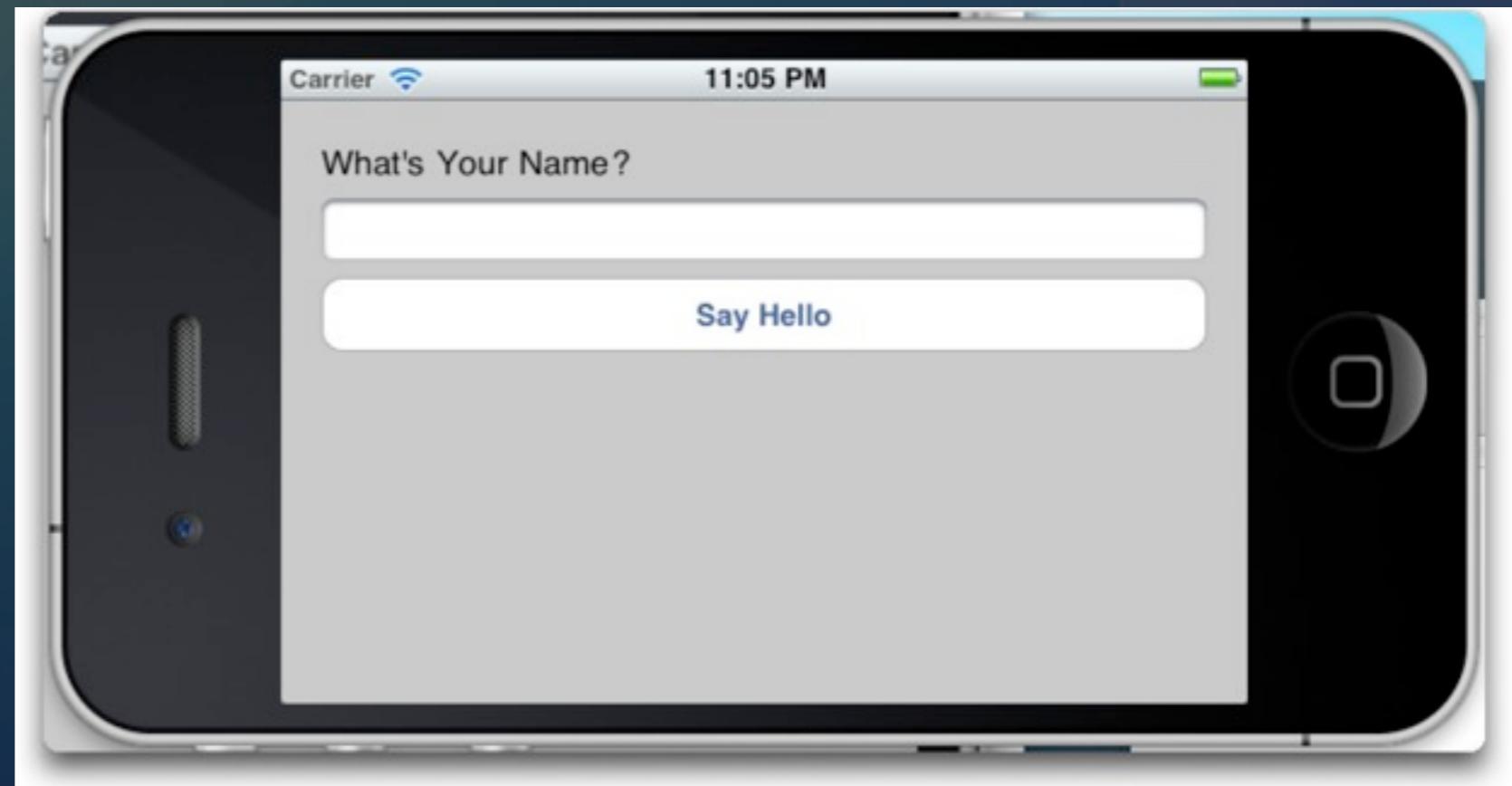
# The Power of Interface Builder

- ➊ All of this, once again, shows the power of Interface Builder
  - ➊ By creating objects in .xib files and specifying their connections, we completely eliminate the init code that would otherwise need to be written
  - ➋ UIApplication will load MainWindow.xib automatically creating the window, delegate, and view controller
    - ➌ The view controller will then load its .xib file, creating the view; the app will then display to the user
  - ➋ Click Build and Run to confirm; Quit the iPhone Simulator

# Add an interface

- ◆ Back in Interface Builder
  - ◆ Add a label that says “What’s your name?”
  - ◆ Add a text field
  - ◆ Add a button that says “Say Hello”
- ◆ Position them vertically and specify resize behavior
  - ◆ Test out the UI via ⌘-R and test switching the phone from portrait to landscape until the UI does what you want (See next slide)

# Our Simple UI



© Kenneth M. Anderson, 2011

# Configure the Code

- We now need to modify the code of our view controller
  - ◆ We need an attribute that points at the text field
    - ◆ because we are going to read its contents to get a name
  - ◆ We need a method that will get invoked when the button is pressed
  - ◆ But first, we need to learn about properties...

# Getters and Setters and Properties

- In the last lecture, I showed a simple Objective-C program that had a string called `greetingText` and I manually wrote the getter and setter method for this attribute
- In Objective-C 2.0, we can declare attributes to be properties and then the compiler can automatically generate the getter and setter methods for us
- Conventions are followed, for an attribute `greetingText`
  - a property will generate methods `greetingText` and `setGreetingText:`

# The syntax of properties in .h files

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

@interface Greeter : NSObject {
    NSString *greetingText;
}

@property (nonatomic, retain) NSString *greetingText;

@end
```

**We define the attribute, then declare it a property**

# The syntax of properties in .m files

```
#import "Greeter.h"  
  
@implementation Greeter  
  
@synthesize greetingText;  
  
@end
```

**In the .m file, we ask the compiler to “synthesize” a.k.a. generate the methods greetingText and setGreetingText; Even though we don’t see them, those methods exist and can be called by Greeter and other classes**

# Back to Configuring the Code (I)

- ◆ In HelloWorldViewController.h add an attribute
  - ◆ IBOutlet UITextField \*name;
- ◆ And a property definition
  - ◆ @property (nonatomic, retain) UITextField \*name;
- ◆ Add this method signature
  - ◆ - (IBAction) sayHello: (id) sender;
- ◆ IBOutlet and IBAction are clues to Interface Builder

# Back to Configuring the Code (II)

- ◆ IBOutlet tells Interface Builder that we will be linking a widget in the interface to this attribute in the class
  - ◆ In this case, our attribute name to our text field
- ◆ IBAction tells Interface Builder that this method will be invoked by one of the widgets in the interface
  - ◆ In this case, it will be invoked when we click our button

# Back to Configuring the Code (III)

- ◆ In the HelloWorldViewController.m file, synthesize the property

- ◆ @synthesize name;

- ◆ Add the following method body

```
- (IBAction) sayHello: (id) sender {  
    NSString *greeting = [[NSString alloc]  
                           initWithFormat:@"Hello, %@", [[self name] text]];  
  
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Hello World!"  
message:greeting delegate:self cancelButtonTitle:@"Ok!" otherButtonTitles:nil];  
    [alert show];  
  
    [greeting release];  
  
    [alert release];  
}
```

# Connect Code and Interface

- ◆ In Interface Builder
  - ◆ Select the text field, ⌘-I, set text to “Frodo”
  - ◆ Control drag from File’s Owner to Text Field, select “name”
  - ◆ Control drag from button to File’s Owner, select “sayHello:”
  - ◆ Save the file; Return to XCode

# Build and Run

- ➊ You're done
  - ➌ Build and Run the App
  - ➌ Click the Say Hello Button; Dismiss the Dialog
  - ➌ Type a name and click Say Hello again
  - ➌ Quit the simulator

# Polish

- ◆ The keyboard would not go away; let's fix this
- ◆ Add to .h file
  - ◆ - (IBAction) doneEditing: (id) sender;
- ◆ Add to .m file
  - ◆ - (IBAction) doneEditing: (id) sender {
    - ◆ [sender resignFirstResponder];
    - ◆ }
- ◆ Connect text field's Did End on Exit event to doneEditing:

# Make it Universal

- ◆ A Universal app is one that runs on both iPhone and iPad
  - ◆ You essentially ask XCode to upgrade your iPhone program to a “universal” program
  - ◆ You add a new .xib file that specifies the UI for the iPad
  - ◆ You can then create a second view controller or hook up the existing view controller to the new .xib file
  - ◆ Regardless, the application will now autodetect which platform it is on and load the appropriate classes/resources

# Step One; There is no Step 2

- ◆ Select the HelloWorld Target
  - ◆ Invoke Project ⇒ Upgrade Current Target for iPad
  - ◆ In the resulting dialog, select “One Universal application”; click “Ok”
- ◆ XCode creates a new .xib file that automatically hooks up to the existing HelloWorldViewController
  - ◆ Since we configured HelloWorldViewController.xib to autoresize its widgets, we are done!
- ◆ Select iPad Simulator 4.2 from XCode’s pop-up and run the app

# That was too easy!

- ◆ Let's create a custom interface for the iPad version
  - ◆ Select Resources-iPad and then File ⇒ New File
  - ◆ Select Cocoa Touch Class
  - ◆ Select UIViewController subclass
  - ◆ Make sure “Targeted for iPad” and “With XIB for user interface” are clicked
  - ◆ Click Next and name file iPadHelloWorldVC

# Configure new user interface

- Double click on iPadHelloWorldVC.xib and create a new user interface: label, text field, button
  - But sized (larger fonts) and formatted for an iPad
  - Set autosizing constraints to deal with portrait and landscape
    - Since the view is so much bigger, we won't let the button and text field resize to fill horizontal space

# Steal from the Best

- ➊ We need to borrow code from our other view controller
  - ➋ It's going to need the same properties and methods to provide the same behavior
  - ➋ We'll ignore this duplication of code for the purposes of this example

# Finally, ensure new VC gets invoked

- We now modify MainWindow-iPad.xib to make sure that it invokes our new view controller
  - It is currently configured to invoke HelloWorldViewController
  - We need it to invoke iPadHelloWorldVC
- Double Click MainWindow-iPad.xib
  - Select HelloWorldViewController and select the Identity Inspector ( $\mathfrak{⌘}-4$ ); Change the Class pop-up to point to iPadHelloWorldVC
  - Now, switch to the attributes inspector and update its NIB file; Click save, return to Xcode and run the program on both platforms

# Wrapping Up

- ◆ Introduction to Interface Builder
  - ◆ Powerful, object-based GUI creation
- ◆ Basic introduction to iOS programming
  - ◆ iPhone application template
  - ◆ views and view controllers
  - ◆ hooking up code and widgets
  - ◆ making a universal application

# Coming Up Next

- Lecture 14: Review for Midterm
- Homework 4 Due on Friday
- Homework 4 solution released on Saturday
- Lecture 15: Midterm
- Lecture 16: Review of Midterm