Presence - Part 1

Due Date

This assignment is due by **11:59 PM, April 28.**

Assignment

Over the next four weeks, we'll be building an iPhone application for viewing online status updates, also known as "presence", for a list of friends. Just so you know what you're getting yourself into, the evolution of the application will be as follows:

Part 1: Build a basic application, displaying static data, using view controllers. Allow the user to navigate to see additional detail.

Part 2: Use table views to display large, dynamic data sets. Incorporate real data into your application using property lists and web services.

Part 3: Using text input and modal view controllers, allow users to update their status. Improve the performance and responsiveness of your application with caching and threading.

Part 4: Integrate with the system address book. Create additional modes for the application, using UITabBarController to toggle between them. And anything else you can think of! This is your opportunity to try out unfamiliar API, polish your interface and experiment a bit.

Now that you have some idea where we're headed, here's what we're expecting for Part 1:

- Create an application that utilizes UINavigationController. The navigation controller may be instantiated in your MainWindow XIB or programatically in your application delegate.
- **Create a view controller that will manage a list of people** (a good name might be PersonListViewController). Use Interface Builder to lay out a view and make connections between the view controller and user interface elements. The list should display an photo, a text label with their name and a "View" button each person (show at least two).
- **Create another view controller that will manage a single-person detail view** (a good name might be PersonDetailViewController). The PersonDetailViewController should expose properties for setting an image, name and status message to display. Again, use Interface Builder to lay out the view and make connections.
- When your application launches, it should create an instance of the PersonListViewController and push it onto the navigation controller stack. An appropriate title should display in the navigation bar.
- When the user presses one of the "View" buttons in the list, create a PersonDetailViewController instance, set its display properties to reflect the person who's being displayed, and push it onto the navigation stack. Again, an appropriate title should display in the navigation bar.

Testing

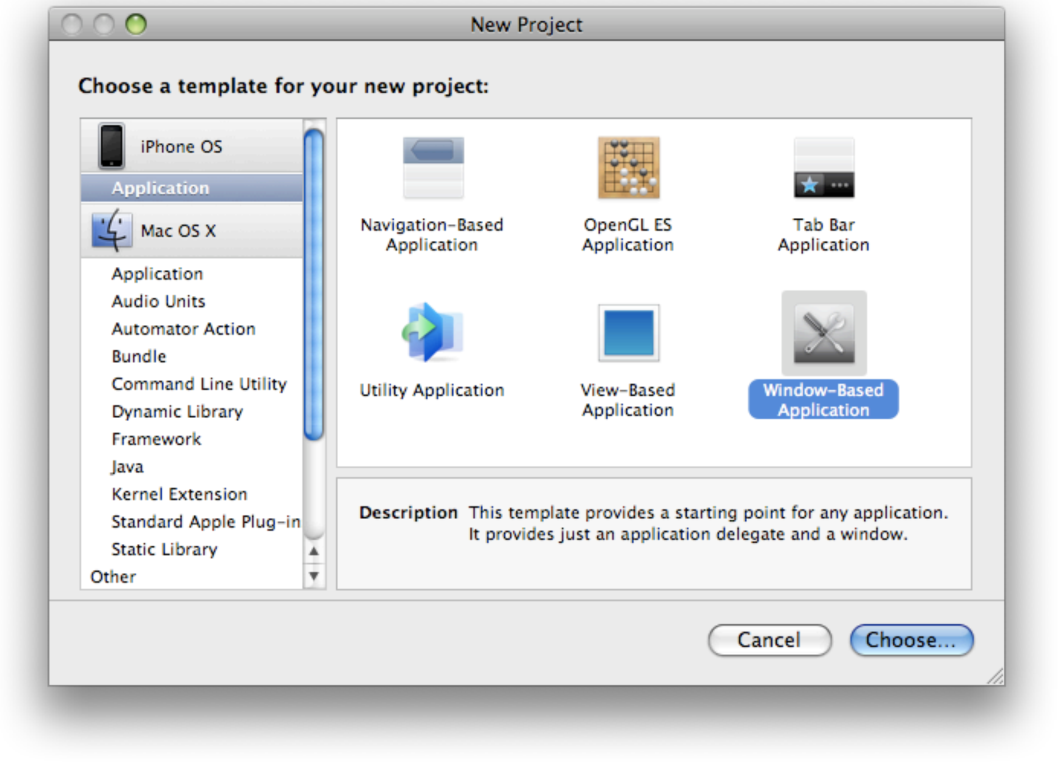
In most assignments testing of the resulting application is the primary objective. In this case, testing/grading will be done both on the behavior of the application, and also on the code.

We will be looking at the following:
1. Your project should build without errors or warnings and run without crashing.
2. Each view controller should be the File's Owner of its own Interface Builder document. **Do not put your entire application into a single Interface Builder document!** It's bad for performance as well as application maintainability.
3. Your program should behave as described above, presenting a navigation hierarchy with a list of people that leads to a person detail view.

Walkthrough

Creating your project in Xcode

To begin with, create a new project using the **"Window-Based Application" template** in Xcode. There is a "Navigation-Based Application" template which may look tempting, but it has a lot of code already written (particular for incorporating a UITableView) which we don't want for this assignment.



Description This template provides a starting point for any application. It provides just an application delegate and a window.

Using this template will create a project that has an application delegate class and a MainWindow.xib Interface Builder document.

Creating your navigation controller

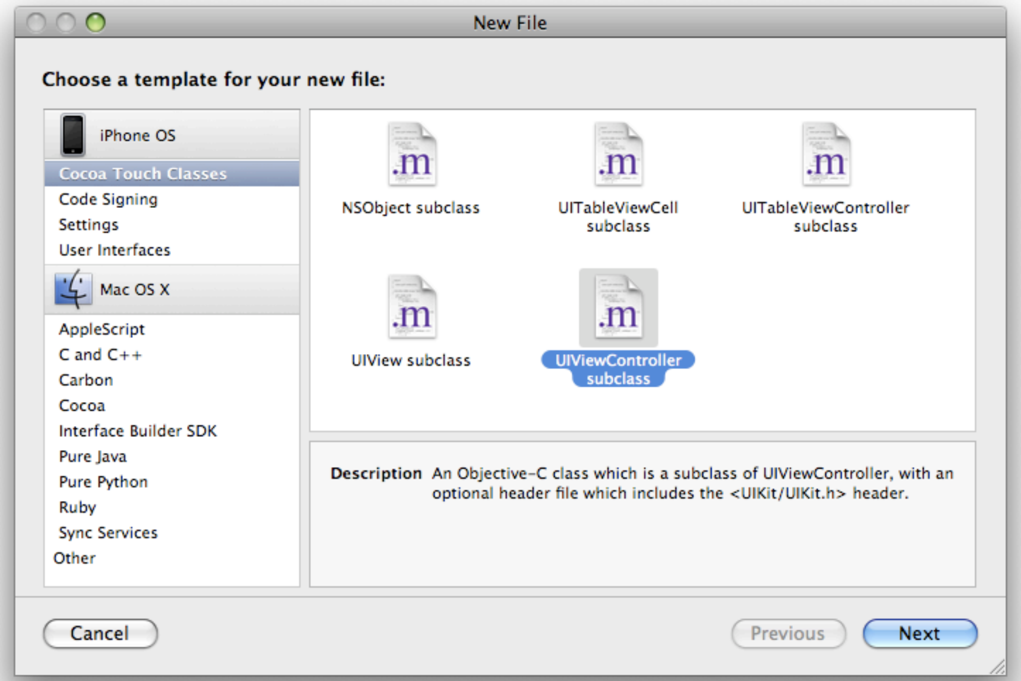
You can create a UINavigationController in your MainWindow NIB or in code using the -init method. Either way, your application delegate should probably have an instance variable referencing it (don't forget to release it in the appropriate place!).

With the navigation controller created, you'll need to add its view to the window. Remember, UINavigationController is a UIViewController subclass, so it has a view property that you can access.

Try building & running your application now & see what happens.

Creating your first view controller subclass

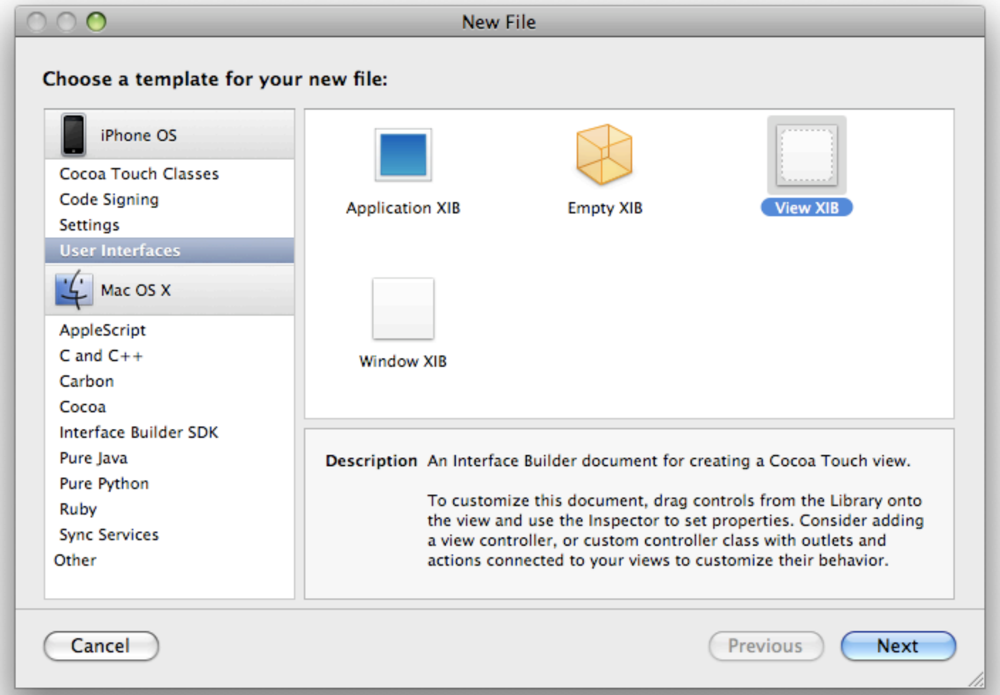
First, we need to create the header and implementation files for our new UIViewController subclass. Select "New File…" and use the **UIViewController subclass** file template.



Description An Objective-C class which is a subclass of UIViewController, with an optional header file which includes the <UIKit/UIKit.h> header.

Using this template will create a UIViewController subclass with quite a few methods already filled in. You can peruse this if you're curious, but then **delete all of the implemented methods** from the .m file. We don't need any of that right now.

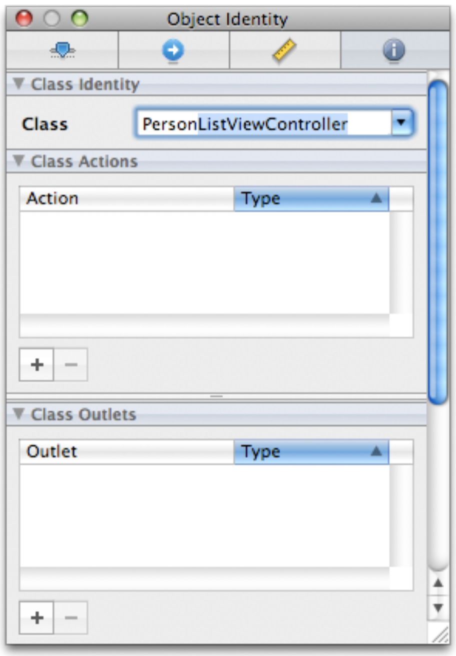
Now we need to create an Interface Builder document for the view controller to manage. An IB document may have a .nib or .xib file extension, but is commonly referred to as a "NIB" either way. Select "New File…" again and **create a View XIB** from User Interfaces under iPhone OS.



Description An Interface Builder document for creating a Cocoa Touch view.

To customize this document, drag controls from the Library onto the view and use the Inspector to set properties. Consider adding a view controller, or custom controller class with outlets and actions connected to your views to customize their behavior.

You're now in the familiar confines of Interface Builder, with an empty view as your blank canvas. Lay out your static list of people using labels, buttons and image views. In addition, though, **we need our view controller to manage the NIB**. There are two steps to make this happen.



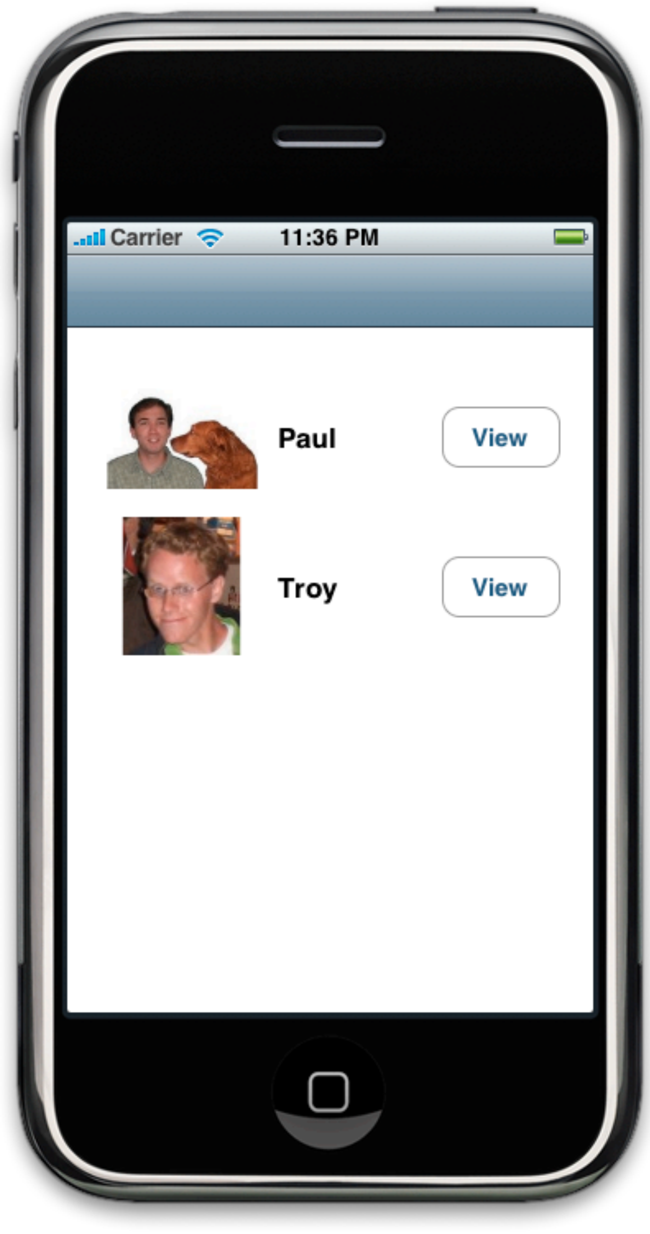
Class PersonListViewController

First, we need to **set the view controller as the custom class for the File's Owner proxy object**. Select the File's Owner object in your nib, then navigate to the Object Identity Inspector in the Tools menu. Type in the name of your view controller subclass in the text field.

Now that the NIB knows that the File's Owner is an instance of your view controller subclass, we can **make the connection from the "view" outlet to the view**. You already know how to do this.

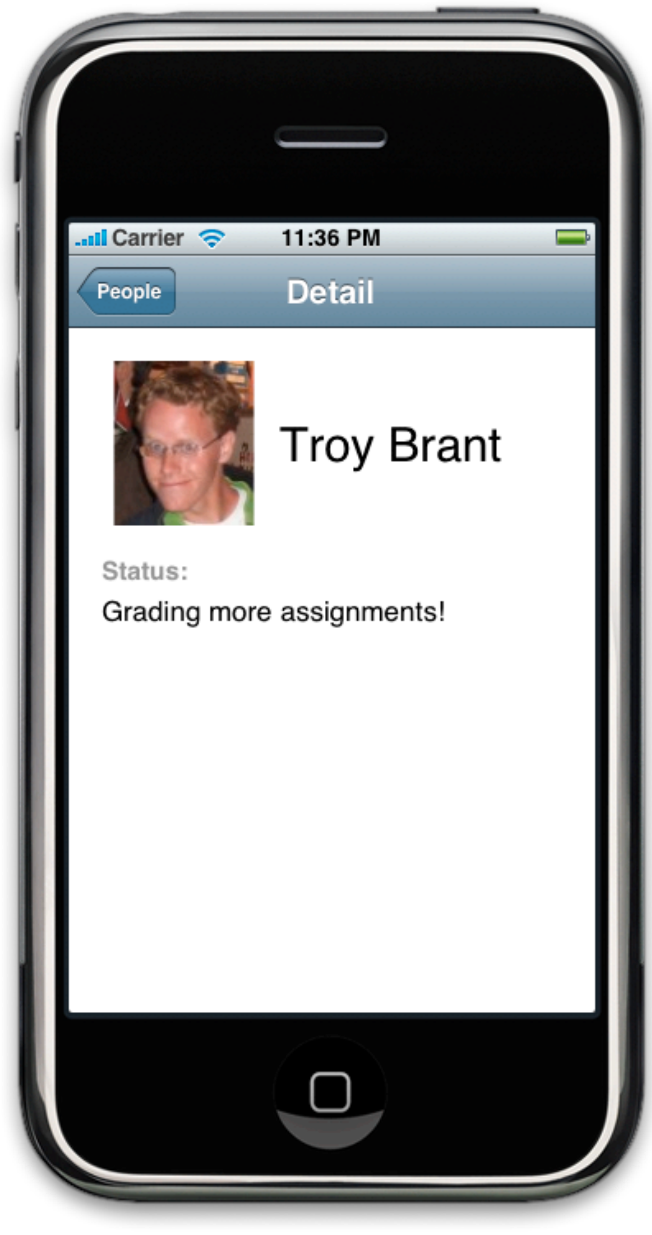
Finally, we want to **create a PersonListViewController instance** and display it in our navigation stack. This code should go in your application delegate in the same place you create your navigation controller. Using the -initWithNibName:bundle: method, pass the name of your Interface Builder document as the first argument (you can leave out the file extension), and [NSBundle mainBundle] as the second argument.

Finally, we need to **push the view controller onto the navigation controller's stack** so that it's displayed. Now, you should be able to build & run from Xcode and see your list. At this point, it should look something like this:



The rest of the assignment, in brief...

Create a second view controller subclass & NIB using the same steps as above. **When a button in your list is touched, push an instance of this second view controller**. You'll want to expose some properties so that it knows what to display. Make sure not to leak memory! The view should look something like this:



In order to display a helpful title in the navigation bar when each view controller is being shown, you'll want to **set the title property for each view controller**. See the lecture slides for a hint on where to do this.

Extra Credit

Here are some suggestions for enhancing this first iteration of Presence.

- Add custom buttons on the left or right side of the navigation bar for the person list view controller. When pressed the button should provide some interesting feedback, like popping up an alert or animating your UI momentarily.
- Our Presence screenshots are pretty dull. Spice yours up with some better colors and artwork.
- Think of a reason for a third view controller subclass, and push it onto the navigation stack in response to some user action. Make it load some data when it's about to appear & save the data when it's about to disappear.