Objectives

- Show the TA that you know how to use Tab Bars Controllers, Navigation Controllers, and how to modally display a view.

Submission

Lab evaluation form.
Complete a lab feedback form.

Note: If you are unable to complete the lab, please be ready to demonstrate it at the beginning of the next lab. iPods are available for checkout from CSG.

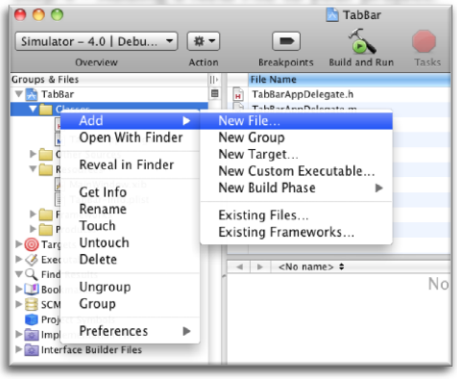
Tab Bar Controller

Tab Bars provide a simple way to easily switch between views.

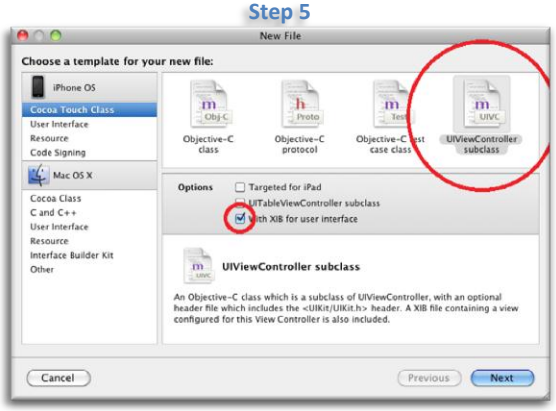
The following steps will help you create an app that switches between three views using a Navigation Controller:

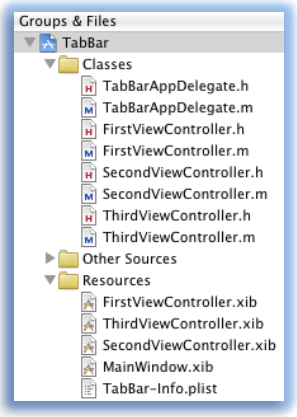
Step 1) Open Xcode.
Step 2) Click *Create a new Xcode project*
Step 3) Select *Window-based Application*. This is the bare bones GUI template. Save the project as *TabBar*.
Step 4) Ctrl-click (right-click) the *Classes* folder and select *Add-> New File.*
Step 5) Select *UIViewController Subclass* and ensure the *With XIB for user interface* option is checked.
Step 6) Name the new class FirstViewController.m and select *Finish*.
Step 7) Repeat steps 4-6 to add two more view controllers (and their .xib files) to your project. Name them SecondViewController and ThirdViewController.
Step 8) Drag-and-drop your .xib files to the *Resources* folder to better organize your code. Your project files should look like this:

Step 4 – Adding a New File to your project.



Step 5

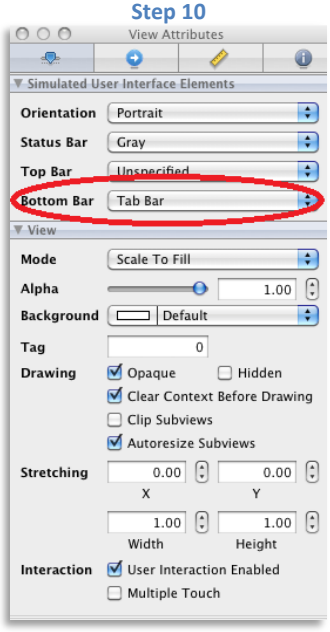




Right now, each View Controller has an identical blank view. Before we create the Tab Bar Controller, you'll want to open up each .xib file for your View Controllers and add a label or something unique so that the three views are different. Follow these steps for using the Interface Builder:

Step 9) Open FirstViewController.xib in the Interface Builder.
Step 10) (Optional) Click on the view and look at the *Attributes Inspector*. Change the *Bottom Bar* from *Unspecified* to *Tab Bar*. While this is not necessary, it will help you place objects on the view so that they are not covered by the Tab Bar.
Step 11) Drag a UILabel from the *Library Window* to the view and change its text to "First View" (or "Second View" or "Third View"). You may optionally do something else, as long as you add something to the view that will distinguish it from the other two.
Step 12) Save and close the .xib.
Step 12) Repeat steps 9-11 for SecondViewController.xib and ThirdViewController.xib.

Step 10



Tab Bar Controllers (continued)

On this page, we'll add code to the App Delegate in preparation for a Tab Bar Controller that we'll add onto the main window. The Tab Bar will allow the user to switch between the three views. First, create an IBOutlet:

Step 13) In TabBarAppDelegate.h, add an IBOutlet for the Tab Bar Controller:

```
IBOutlet UITabBarController *tabBarController;
```

Step 14) In TabBarAppDelegate.m, add the following line of code at the top of the *application:didFinishLaunchingWithOptions:* method to add your Tab Bar onto the main window:

```
[window addSubview:tabBarController.view];
```

Step 15) Open MainWindow.xib in the Interface Builder.
Step 16) From the *Library*, drag-and-drop a UITabBarController into the Document Window (MainWindow.xib).
Step 17) By default, the Tab Bar will have two tabs. To add a third tab, select the Tab Bar Controller from the Document Window and click the + to add another View Controller in the *Attributes Inspector*.

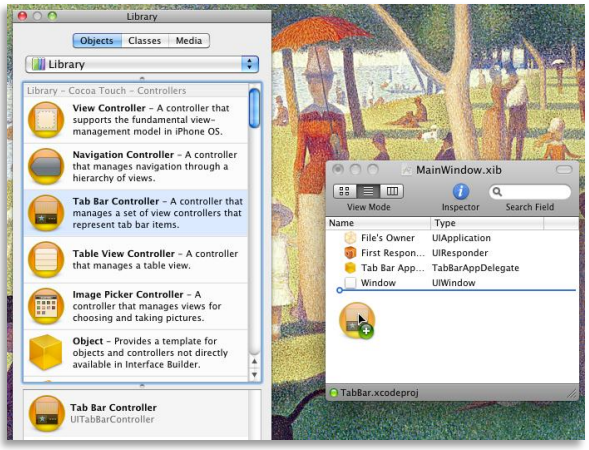
Next, you'll want to connect each tab with one of your three ViewControllers:

Step 18) Expand the Tab Bar Controller in the Document Window.
Step 19) Select the first tab (or the first UIViewController in the document window) and change its Class Identity to FirstViewController in the Identity Inspector.
Step 20) Repeat step 17 for the other two tabs, changing their class identity to SecondViewController and ThirdViewController.

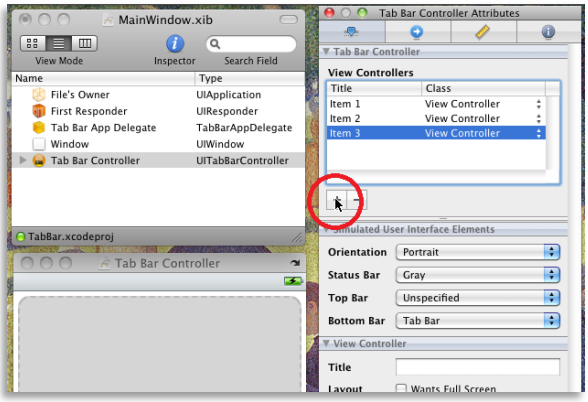
Finally, complete the application:

Step 21) Ctrl-click (right click) on the *Tab Bar Controller* in the Document Window. Drag a blue connection from *New Referencing Outlet* to the *Tab Bar App Delegate* and select *tabBarController*.
Step 22) Run your project in the simulator and show the TA that you can switch between your three views using tabs.

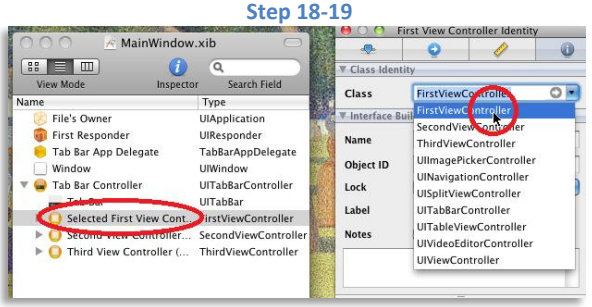
Step 16



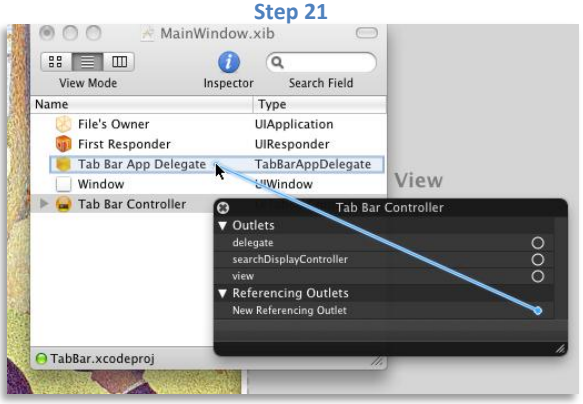
Step 17



Step 18-19



Step 21



Navigation Controller

Navigation Controllers allow you to create a stack of Views that the user can jump back and forth. The user starts at the root view and subsequent views are pushed onto the stack. Only the root view needs a Navigation Controller; subsequent views only require a Navigation Item. The following steps will help you create an app that switches between three views using a Navigation Controller:

Step 1) Follow steps 1-7 from the Tab Bar Controller section but do not generate .xib files with the user interface (Alternatively, you can reuse the old project and simply delete the three .xib files and undo any actions taken in the App Delegate and MainWindow.xib)

Steps from previous section (do not create XIB files):

Step 1) Open Xcode.
Step 2) Click *Create a new Xcode project*
Step 3) Select *Window-based Application*. This is the bare bones GUI template. Save the project as ***Nav***.
Step 4) Ctrl-click (right-click) the *Classes* folder and select *Add-> New File.*
Step 5) Select *UIViewController Subclass* and ensure the *With XIB for user interface* option is **NOT** checked.
Step 6) Name the new class FirstViewController.m and select *Finish*.
Step 7) Repeat steps 4-6 to add two more view controllers (and their .xib files) to your project. Name them SecondViewController and ThirdViewController.

Step 2) In NavAppDelegate.h, import the header files for your view controllers.
Step 3) In NavAppDelegate.h, add IBOutlets for your three view controllers and a UINavigationController, and add two IBAction methods for two buttons that you will create in later steps. Suggested instance names for IBOutlets and method names for IBActions are:

```
IBOutlet UINavigationController *navigationController;
IBOutlet FirstViewController *firstVC;
IBOutlet SecondViewController *secondVC;
IBOutlet ThirdViewController *thirdVC;

- (IBAction)next;
- (IBAction)next2;
```

Step 4) In NavAppDelegate.m, add the view of your Navigation Controller to the main window by inserting the following line of code at the top of the *application:didFinishLaunchingWithOptions:* method:

```
    [window addSubview:navigationController.view];
```

*Make sure *navigationController* matches the instance name you defined for the IBOutlet in step 3.
Step 5) In your two IBAction methods, you need to call the pushViewController:animated: method on the navigation controller in order to push a view onto the stack:

```
- (IBAction)next {
    [navigationController pushViewController:secondVC animated:YES];
}
- (IBAction)next2 {
    [navigationController pushViewController:thirdVC animated:YES];
}
```

Navigation Controllers (continued)

Step 6) Build your project so the IBOutlets and IBActions will appear in the Interface Builder.

Next, we'll develop our interface using Interface Builder:

Step 7) Open MainWindow.xib.
Step 8) From the *Library*, drag-and-drop a UINavigationController into the Document Window (MainWindow.xib).
Step 9) From the *Library*, drag-and-drop **two** UIViewControllers into the Document Window (MainWindow.xib).

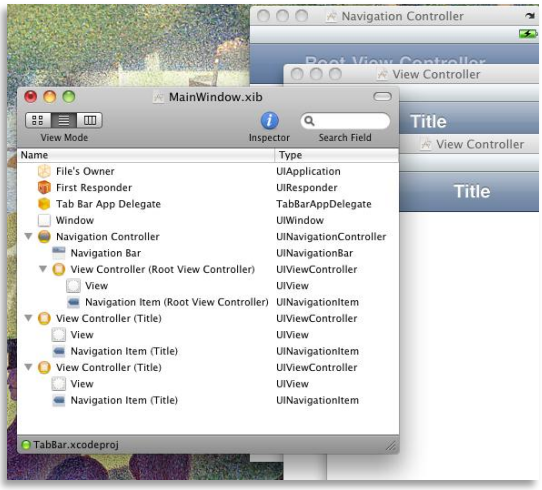
The root view controller becomes associated with the navigation controller, and view controllers that are pushed onto the stack must be associated with a navigation item.

Step 10) From the *Library*, drag-and-drop **two** UINavigationItems into the Document Window (MainWindow.xib), one under each ViewController. Note that the View Controller inside the Navigation Controller already has a Navigation Item.
Step 11) Drag and drop three UIViews into each ViewController.
Step 12) Add two UIBarButtonItems, one onto the Root View Controller, and another on the second View Controller. These will act as our next buttons.
Step 13) Rename the Titles in the Navigation Items to *First*, *Second*, and *Third*.
Step 14) For each ViewController, change its *Class Identity* in the Identity Inspector to match with its title (either FirstViewController, SecondViewController, ThirdViewController).

Now it's time to connect our Interface Builder objects with the IBOutlets and IBActions we created earlier in the App Delegate.

Step 15) In the Document Window, Ctrl-click (right-click) the Nav App Delegate. Connect the IBOutlets to their respective ViewControllers or Navigation Controller. Connect the IBAction next to the selector event of the first Bar Button, and connect next2 to the second bar button.
Step 16) Save and run your project in the simulator. Show the TA that you are able to move back and forth between the three views.

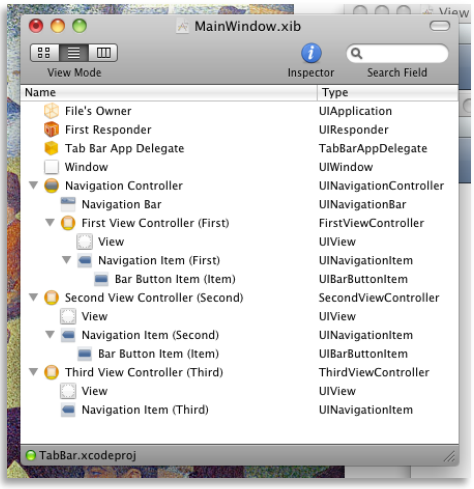
Step 11 – Document Window after Step 11.



Step 12

Step 13

Step 14



Modal View Controller

Reference:
http://developer.apple.com/iphone/library/featuredarticles/ViewControllerPGforiPhoneOS/ModalViewControllers/ModalViewControllers.html

Skim the Apple documentation for a background on Modal View Controllers. Alternatively, you can start a new project and use the Utility App template. The template for a Utility App has code for modally displaying another ViewController's view.

Step 1) Reuse one of the previous projects and add functionality so that the user may modally display a view and return. You will need to create two buttons and their corresponding IBAction methods: one on a view to show the new view and one on the modal view to dismiss the modally displayed view.

Step 2) Show your app to the TA when you have completed it.

Delegates: Working with TextFields (Optional)

A common beginner's problem is figuring out how to hide the keyboard after typing text into a UITextField. The answer is to make YourViewController a delegate for the UITextField object. While handling an event, then you tell the UITextField object to resign its FirstResponder status, which hides the keyboard.

Step 1) Edit your viewController.h and viewController.m files by adding with the following:

YourViewController.h:

```
//...
@interface YourViewController : UIViewController <UITextFieldDelegate> {
      IBOutlet UITextField *myText; //optional
}

@end
```

YourViewController.m:

```
//...
@implementation YourViewController

- (BOOL) textFieldShouldReturn(UITextField *)textField {
      [textField resignFirstResponder];
      // or [myText resignFirstResponder];
}
//...
```

Note: A delegate is an assigned object that another object delegates work. In this case, the object doing the work adds <UITextFieldDelegate> to the class definition to do the work for a UITextField object. When you add a delegate definition, your class is given a list of optional and mandatory methods to implement. For UITextField, all methods are optional to implement. Using delegates is a way to ensure that an object to which you delegate work has a standardized set of methods that the delegating object can send messages.

A class can be the delegated object for multiple objects. For such cases, the delegate definitions are separated by commas. For example: @interface YourViewController : UIViewController <UITextFieldDelegate, UIAlertViewDelegate>

If you right-click on <UITextFieldDelegate>, you can *Jump to Definition* and view all the methods defined in the delegate protocol (or you may use Apple's documentation).

Step 2) Open up YourViewController.xib in the Interface Builder.

Step 3) Add a UITextField, resize, and position it on the view. You must right click and drag a connection from Delegate to File's Owner so that the instance of YourViewController acts as the UITextField's delegate. If you defined an IBOutlet, you can connect the referencing outlet at this time as well.

Step 4) Save and test your project in the simulator. Show the TA that you can successfully hide the keyboard.