Objectives

- Create an app that demonstrates you know how to use Tab Bars Controllers, Navigation Controllers, and how to modally display and dismiss a view controller.

Submission

Demo your project and give the Lab Evaluation form to your TA.
Complete the online feedback form.

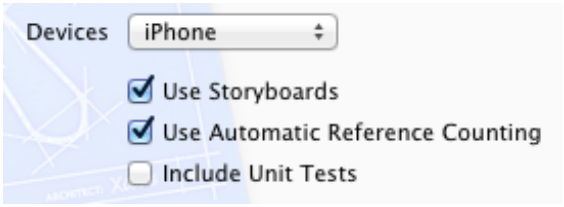
Pre Lab

Read through the entire lab manual. It's not that long; it's mostly pictures.

Notice!

This lab manual is for use with Storyboarding, so older devices are not supported. If your device is running a version of iOS less than iOS 5.0, you will only be able to use the simulator for the lab.

Storyboarding



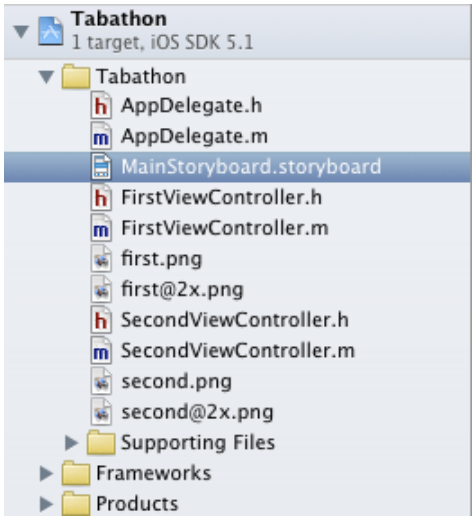
Step 1) Open Xcode. Create a new project.

Step 2) Use the "Tabbed Application" template.

Step 3) Ensure that "Use Storyboards" is selected.

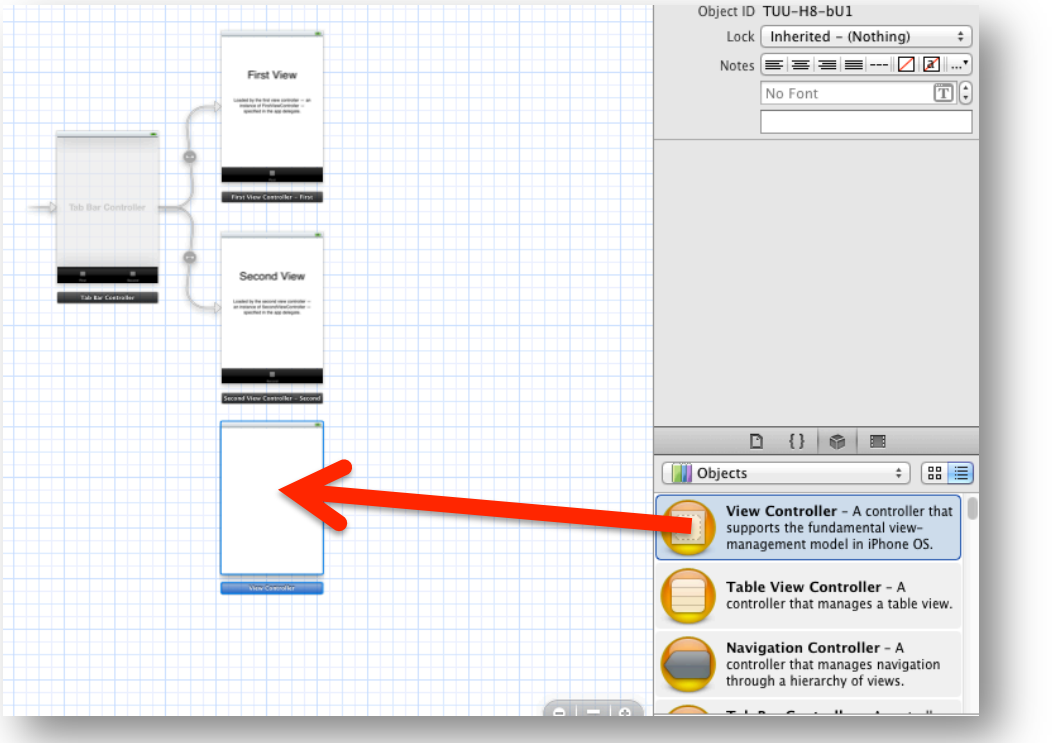
Step 4) Open the MainStoryboard.storyboard file.

Step 5) Read through the rest of the lab manual to understand the process of creating new tabs, adding navigation controllers, modally displaying a view controller, and changing the class of your view controllers to work with your custom subclasses of UIViewController.



Add a new tab to the Tab Bar Controller

Step 1) Drag a new View Controller onto the storyboard.



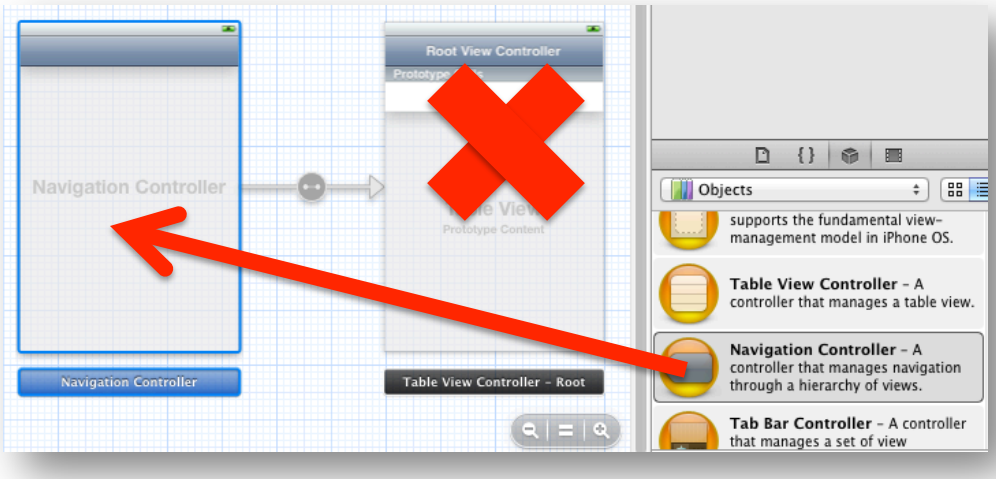
Step 2) Right click on the new View Controller and drag a blue connector from "Relationship" to the Tab Bar Controller. You've added a new tab! You should see a third button appear in the Tab Bar Controller and your View Controller will have a Tab Bar Item on the bottom of the view.

Step 3) Test your program in the simulator to ensure your app has three tabs.

Creating Navigation Controllers

Navigation Controllers handle one or more view controllers that can be thought of as a stack. Initially, the "root view controller" is displayed. The user can "push" a new view controller onto the navigation controller. Pushing a new view controller will display a back button on the left side of the navigation item; this button can be used to "pop" a view controller off the stack and go back.

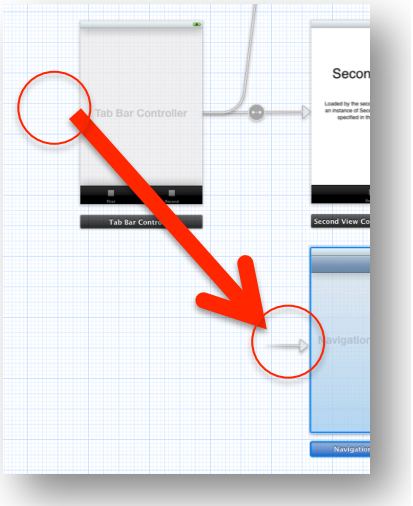
Step 1) Add a Navigation Controller to your storyboard.



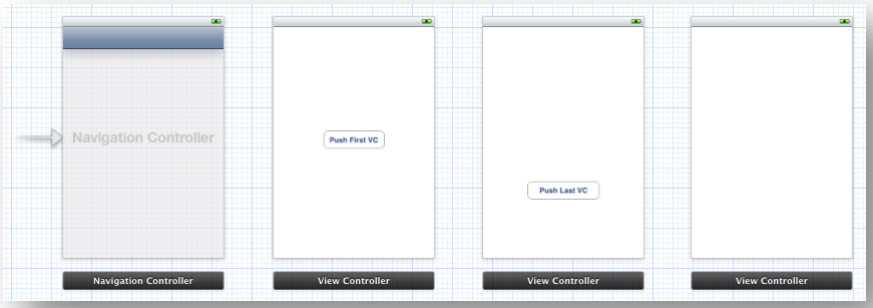
Step 2) We will learn about Table View Controllers in a later lab; delete the Table View Controller.

Step 3) Add your Navigation Controller as a new tab if you want your Navigation Controller embedded in a tab (not pictured).

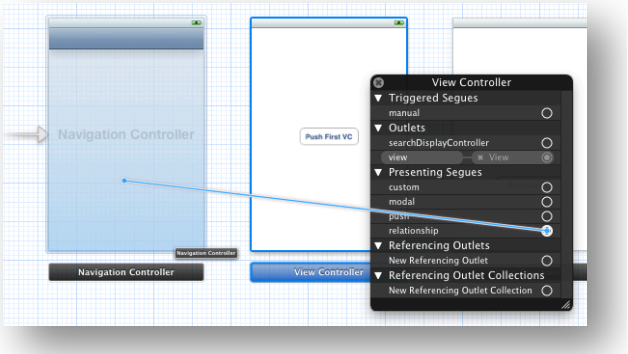
Alternatively: Drag the starting arrow from the Tab Bar Controller to the Navigation Controller as pictured to the right. This will load the Navigation Controller as the starting point of your app.



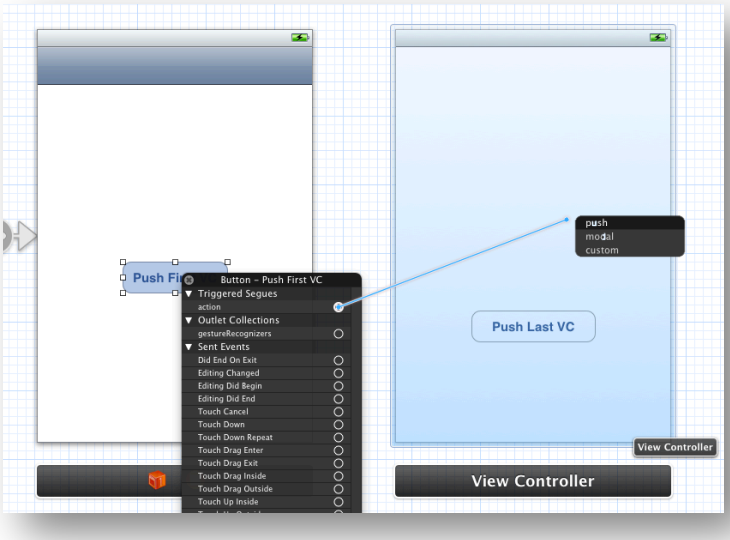
Step 4) Add three new View Controllers on to the storyboard. Place a button on the first two (you must be "zoomed in" in order to place items on the views).



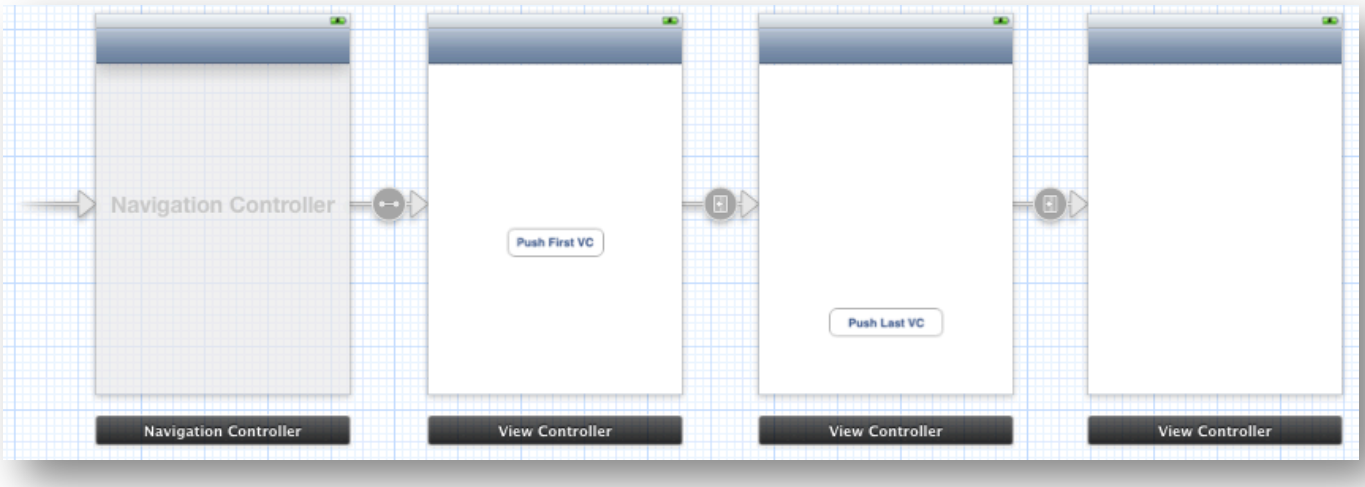
Step 5) Right click on the first view controller and drag a blue connector from “relationship” to the Navigation Controller in order to make it the root view controller. You only need to do this for the root view controller.



Step 6) Right click on the button in the root view controller and drag a blue connector from “action” under the Triggered Segues category to the next view controller. Select the “push” option.



Step 7) Repeat step 6, but from the button in the 2nd view controller to the 3rd view controller. Your storyboard should now look like this (notice the “Segues”, or arrows, connecting the view controllers):

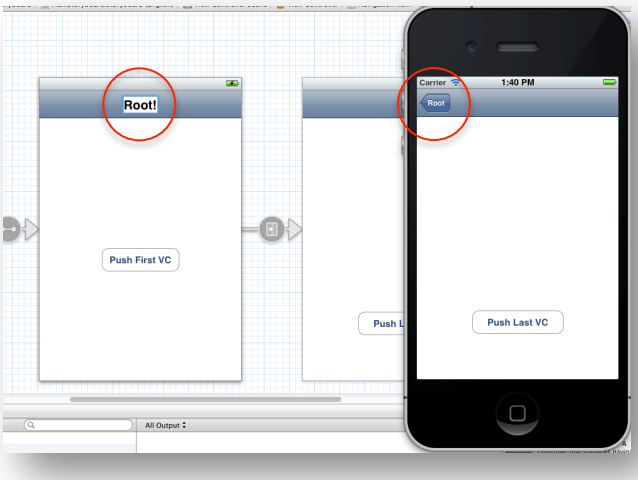


Step 8) Test your application in the simulator.

Additional Notes on Navigation Controllers:

Double click on the Navigation Item inside a view controller to edit the "title" of the view controller. When a new view is pushed onto the stack, the back button will take the name of the previous view controller! You can also edit this programmatically through the "title" property of the UIViewController class.

```
self.title = @"I haz cheezebarger";
```

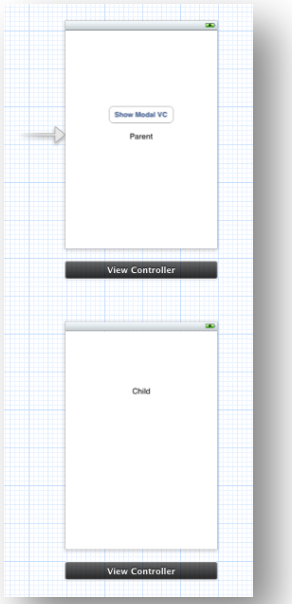


You can programmatically push and pop view controllers on and off the navigation controller's stack. To do so, access the navigationController property of the UIViewController class and call the `pushViewController: animated:` or `popViewControllerAnimated:` methods.

```
UIViewController *vc = [[UIViewController alloc] init];
[self.navigationController pushViewController:vc animated:YES];

[self.navigationController popViewControllerAnimated:YES];
```

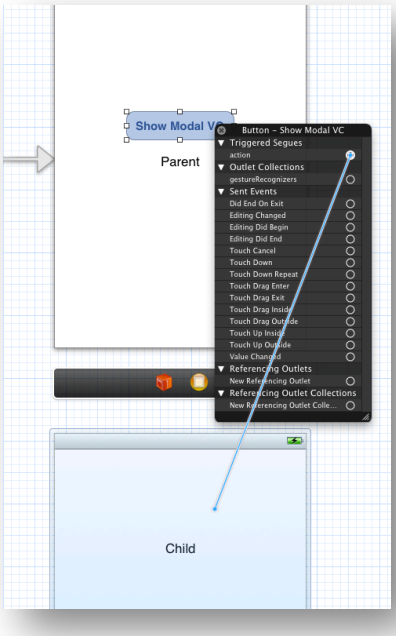
Modal Displayed View Controllers



A Modal View Controller is a child view controller that requires the user to interact with it before they can return to the parent view controller. An example is an alert using the UIAlert class. Another example would be in the calendar app when adding a new event to the calendar.

Step 1) Choose any view controller to act as the parent; this view controller will be doing the "presenting". Add a button to the view controller.

Step 2) Add a new view controller to your storyboard to act as the child view controller (or use an existing one!); this view controller will be the "presented" view controller. Add a Label or some other component to identify the child from the parent.



Step 3) Right click on the button in the parent (the presenting view controller) and drag a blue connector from "action" under the Triggered Segues category to the child view controller. Select the "modal" option.

Step 4) Unfortunately, the only ways to dismiss a modal view controller are through code or through a custom segue. Parents "dismiss" the child modal view controller.

The UIViewController class has two properties to help the parent (the "presenting" view controller) and child (the "presented" view controller) communicate. Use the *presentingViewController* property in the child to communicate to the parent. Use the *presentedViewController* property of the parent to send messages to the child.

The easiest way to dismiss the modal view controller would be for you to add a button to the child view and have it trigger an IBAction in the child view controller:

```
- (IBAction) dismissModalVC {
    [self.presentingViewController dismissModalViewControllerAnimated:YES];
}
```

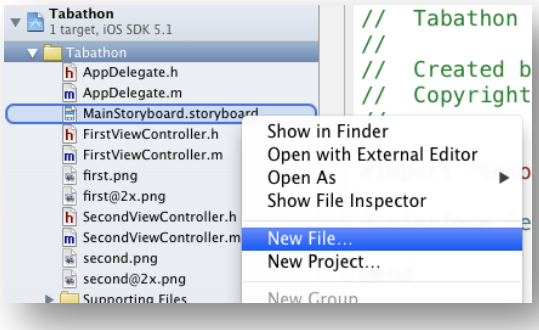
Step 5) Continue to the next part of the lab for a walk-through on how to dismiss the modal view controller.

Creating and Using Custom Subclasses of UIViewController

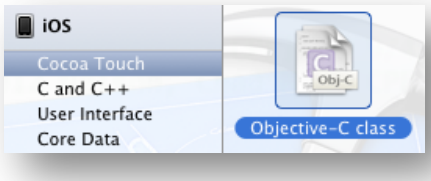
If you need to perform any code or link the visual objects in Interface Builder (i.e. IBActions, IBOutlets), you'll need to have a custom subclass of UIViewController. So far in lab, we've used the custom subclass templates that Xcode generates for us when we start a project. It's time to learn how to create your own.

Let's reuse the modal view controller tutorial. We'll create a new subclass of UIViewController named "ModalViewController" and use it to dismiss the modal view.

Step 1) Right click in the Project Navigator and select "New File" from the context menu.



Step 2) Select the *Objective-C class* and click *Next*.



Step 3) Name the *Class* "ModalViewController" and ensure the *Subclass of* text field says UIViewController. Click *Next* and *Create*. Do not create a new .xib file. Do not target for iPad. Two new files have been added to your project.

Step 4) Open ModalViewController.h and ModalViewController.m. Add the following IBAction in the appropriate places:

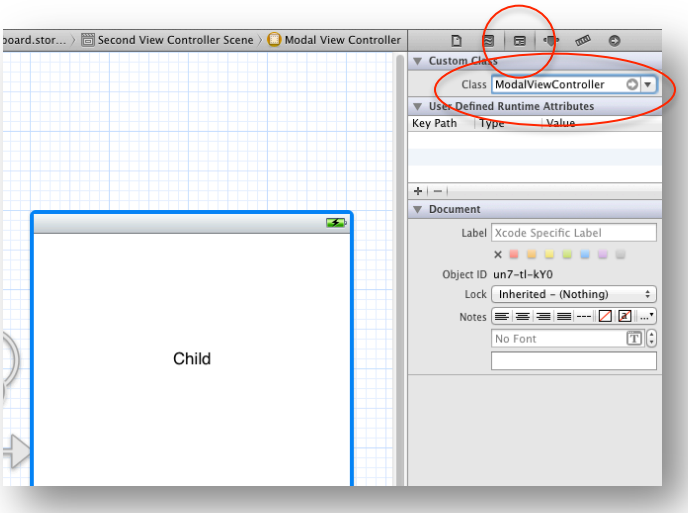
In the interface block in ModalViewController.h add:

```
- (IBAction) dismissModalVC;
```

In the implementation block in ModalViewController.m add:

```
- (IBAction) dismissModalVC {
    [self.presentingViewController dismissModalViewControllerAnimated:YES];
}
```

Step 5) Open MainStoryboard.storyboard. Select the child view controller that is modally presented. Open the class inspector and change the class to ModalViewController.



Step 6) You can now add a button to the child view controller and connect it with your IBAction. Do so now.

Step 7) Test your application in the simulator.

Delegates: Working with TextFields

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 the delegate object for the UITextField object. While handling the textFieldShouldReturn: event, then you tell the UITextField object to resign its FirstResponder status, which hides the keyboard.

Step 1) Edit your custom view controllers .h and .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 object is an object that signs up to received messages from another object. In this case, the view controller signs up by adding <UITextFieldDelegate> to its class definition, then sets the delegate property of the UITextField object. The text field then sends messages to the view controller based on the UITextFieldDelegate protocol. When you add a protocol definition to your class in <> brackets, 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 delegate object for multiple objects. For such cases, the protocols 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 the .xib or .storyboard associated with your custom view controller.

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 (.xib) or View Controller (.storyboard) so that your custom view controller 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. You should be able to successfully show and hide the keyboard.

Create an App

The requirements for the app are simply that it incorporates what you have learned in lab.

Requirements:

- App uses a Tab Bar Controller
- App uses a Navigation Controller
- App displays at least on modal view (a view controller, alert, etc)
- Creativity & Completeness – Is the app creative and complete?