iPhone/iPod Application Development

Views and View Controllers

Class # 4, Fall 2016

Jitin Parikh

parikhj@cod.edu

Resource Files - Files Owner

- On launch of an app iOS starts what's called a skeleton app - NSapp.
- When you designed app, proxy objects such as view controller, first responder and view were created for you.
- On your design canvass you can put object per your design, for e.g. the UITextField. Such objects gets linked to Files' owner for outlets and actions by you per your design needs. This approach allows you to link your objects, code etc to Files Owner as necessary without knowing anything about launch or load logic of apps.
- Objects linked to Files Owner such as UITextField or any custom object can be modified as needed without any linkage to NSapp. Think Delegation pattern.
- At run time as part of application load, nib files will get loaded. Once loaded NSapp and reestablish connections owned by files owner and other proxy objects.

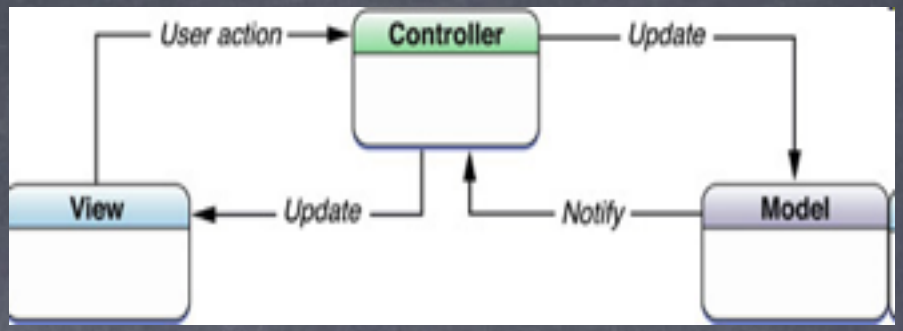
Resource Files - First Responder

- A proxy object that represents the first object in your app's dynamically determined event responder chain. It acts as a stand in target for any action messages (think touch-up inside).
- A responder chain is a sequence through which events will travel. Object responsible for handling the event will take event and process it, others will simply ignore and it. If no objects have signed up to handle the event it will simply be dropped at end of the chain.
- Remember application upon launch starts with skeleton app - NSapp. You have designed objects and have decided if these objects will respond to events using IB. First Responder connects through files owner.
- When storyboard is loaded into memory, the First Responder proxy object gets resolved and there is nothing you have to do to make that happen.
- If you are interested or a you need to know how event handling works in iOS, refer to Event Handling Guide for iOS.

Resource Files - Interface Objects

- Objects that implement app's user interface. New project skeleton will have at least one object called Window.
- At run time Xcode will "compile" storyboard into a nib. When iOS loads NIB all these interface objects will be "instantiated" as part of nib loading.
- Visual objects such as windows, views, fields, controls etc. are interface objects.
- Non-visual objects such as controller objects are also part of the interface objects.
- You may instantiate control objects as well visual objects in your code. However much easier to do that in IB.

MVC Pattern Review



- In MVC Pattern assign objects one of three roles: model, view or controller.
- Each type of object separated from the others by abstract boundaries and communication with objects.
- Model objects encapsulate the data specific to app and contain logic and computation that manipulate and process data. For e.g. a database of historic temperatures for lang/lat on specific time of the year could be data objects if we wanted to enhance our temp. conversion app to give users reference.
    - User action in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes it notifies a controller object which updates the view objects.
- View object is what users can see in an app. View object knows how to draw itself and responds to user actions. Major purpose is to display data from the applications model objects and enable view/edit of that data.
    - View objects learn about changes in model data through the applications controller objects and communicate user-initiated changes through controller objects to an applications model objects.

MVC Pattern Review continued

- Controller Object acts as an intermediary between one or more of an application's view objects and one or more of its model objects. They are conduit through which view objects learn about changes in model objects and vice-e-versa.
    - Controller object interprets user actions made in view objects and communicates new or changed data to the model layer. When model object change, a controller object communicates that new model data to the view objects so they can update display.
- In our proposed Temperature converter there is no model or data object. RootViewController converted (manipulated data) temperature and notified view objects.
- Key take away is to keep data, view and control separate when using MVC pattern.
- Refer to http://developer.apple.com and search for MVC Pattern to read up more.

View Controller and their Role in Applications

- They are controller objects in MVC Paradigm. View Controllers:
  - Provide the logic needed to manage basic application behaviors.
  - Manage the presentation and removal of content from the screen.
  - Manage the reorientation of view in response to device orientation changes.
- Limited screen real estate leads to creative ways of presenting information to the users. Information generally spreads over multiple screens and show and hide screen at appropriate time with right gestures.
- View controllers provide standard APIs and behavior such as back buttons to navigate pages on the stack

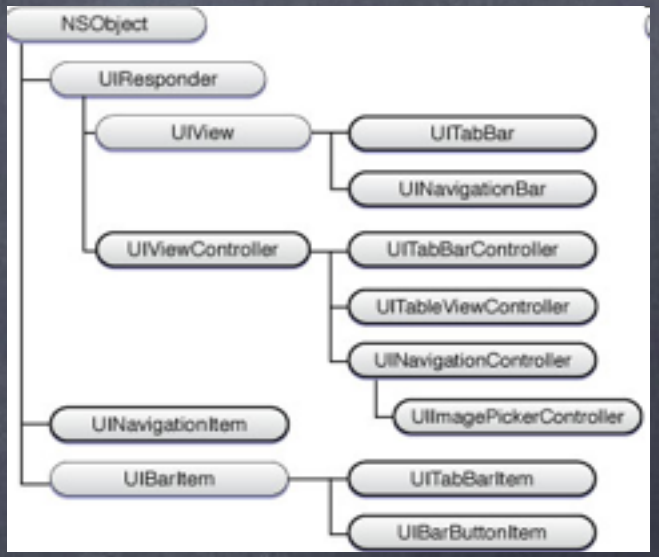
Types of View Controllers

- Custom View Controller is a controller object that you define for the express purpose of presenting some content on the screen. Most apps display data on several different screens. For e.g. one screen may present a list of items in a table, another screen may display details of a row and third screen may display content of a cell. The corresponding architecture for such an application would involve the creation of separate view controllers to manage the marshalling and display of data for each distinct type of screen.

- Container View Controller is a VC object that manages other VC and defines the navigational relationship among them. Navigation and tab bar controllers are examples of Container VCs. Typically you use the Container VC provided by the system as is. Typically you do not define Container VC because each Container VC (say Navigation) provides complete implementation of a specific type of interface.

- Modal View Controller is a VC that is presented in a specific way by another VC. The most common reason to present a VC modally is to prompt user for data or action.

View Controller Classes in the UI Kit



Custom View Controllers

- Primary coordinating objects for app contents. Nearly all apps has at least one custom VC, contains the logic and glue code needed to facilitate interaction between app's data and it's views.
- As a rule one screen's worth of content to be managed by one custom VC. The one to one correspondence between a VC and screen is very important consideration. Do not use multiple VC to manage different portion of the same screen nor use a single custom VC to manage multiple screens.
- If you need to split single screen into multiple areas and manage each one separately, use generic controller objects from NSObject instead of UIview controllers. Navigation and other behavior will not work properly. Let the custom VC coordinate the overall screen interactions and forward messages to the generic controller objects it manages.
- For the Custom VC:
    - Subclass UIViewController to create custom VC
    - Member variables point to the objects containing data to be displayed
    - Member variables or outlets pointing to key view objects with which custom VC must interact
    - Action methods that perform tasks associated with buttons and other controls in the view hierarchy
    - Any additional methods needed to implement your VC's customer behavior.

Table View Controllers

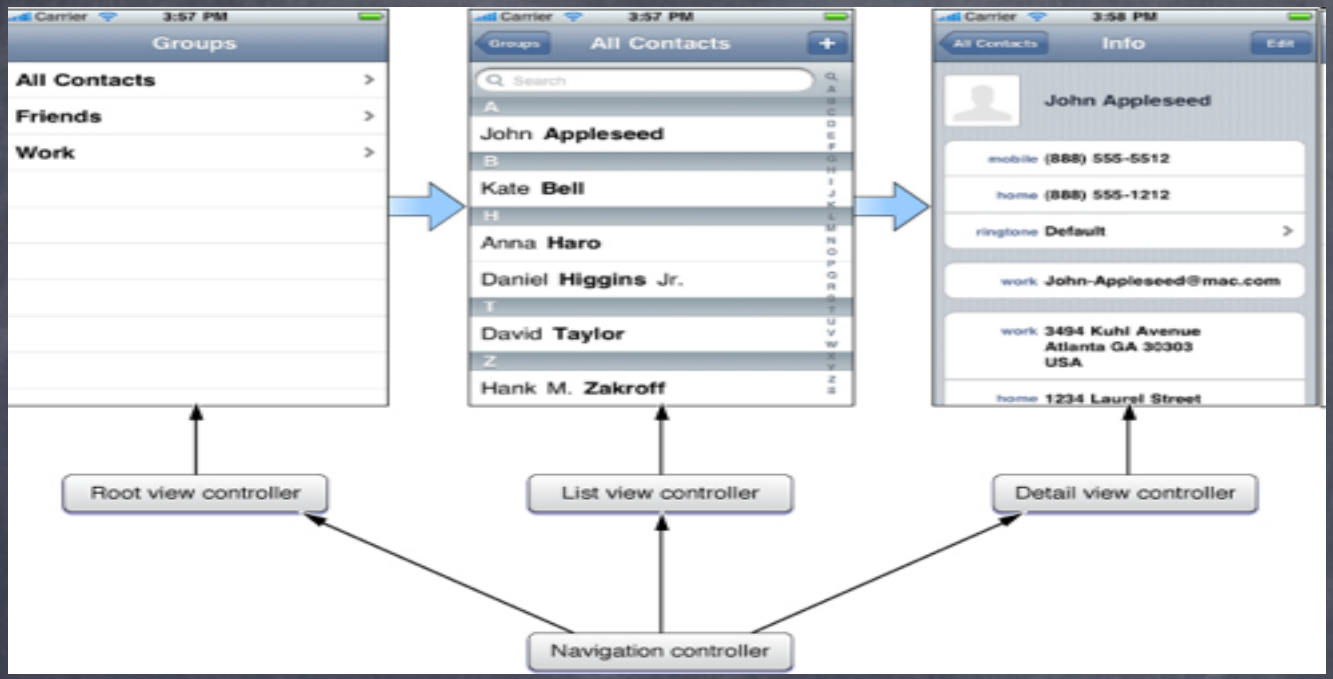
Table View Controllers

- UITableViewController is custom view controller designed for managing tabular data. TableViewController provides automatic support for many standard table-related behaviors such as selection management, row editing, table configuration, and others.
- In this visual example of a TableViewController has a pointer to the root view of the UI interface through "view" property and also a separate pointer to the table view displayed in that interface.

Navigation Controllers

- Navigation Controller is a container view controller that can be used to present data that is organized hierarchically.
- Navigation Controller is an instance of the UINavigationController class, which you use as is and do not subclass. The method of this class provide support for managing a stack based collection of customer view controllers. Stack represents the path taken by the user through the hierarchical data.
- Primarily navigation controller manages other view controllers. However it can also manage a few views by itself. For e.g. it manages a navigation bar that displays info about the user's current location in data hierarchy, b back button to navigate to previous screen and any customer control the current view controller needs. It also manages tool bar.
- In most cases views are not directly modified but configured through UIViewController (for view changes on toolbar).

Navigation Controller Example

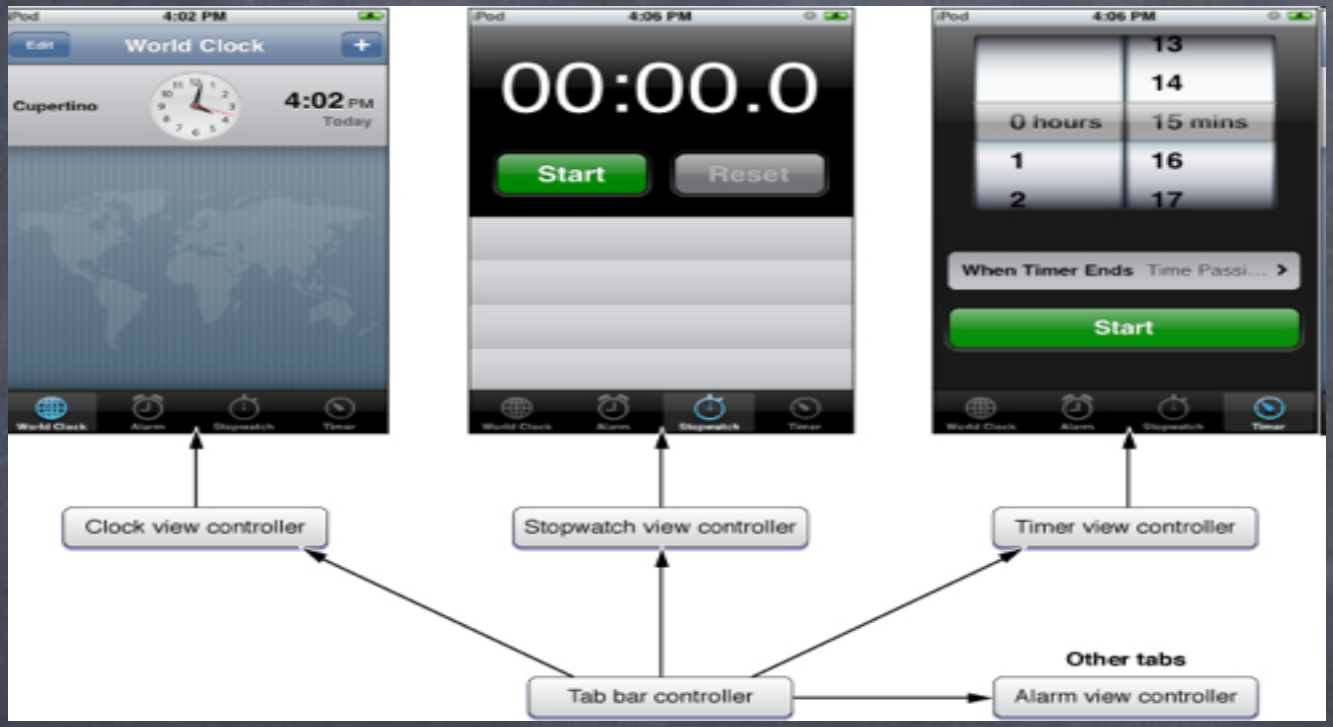


Navigation controller used to present contact information to the user.
Each screen is managed by a customer view controller object that maintains info at that specific level of the data hierarchy.
As the user interacts with controls in the interface, those controls tell the navigation controller to display the next view controller in the sequence or dismiss the current view controller.
Navigation view controller is container type of controller – just repeating.

Tab Bar Controller

- A tab bar controller is a container view controller that you use to divide your app into two or more distinct modes of operations. Like Navigation controller, tab bar controller is an instance of the UITabBarController class which you use as is and do not subclass.
- The modes of the tab bar controller are presented using tab bar view. Selecting a tab causes an associated view controller to present its interface on the screen.
- Use tab bar controllers in situations where your app presents different types of data or presents the same data in significantly different ways.
- The tab bar controller facilitates the automatic switching of modes in response to user taps on the tab bar view.
- If there are more modes than there is space for tabs, the tab bar controller also manages the selection of tabs that are not normally visible and the customization of the tabs that are visible.

Tab Bar Controller Example



Each mode has a root view controller to manage the main content area.

The clock and alarm view controllers both display a navigation-style interface to accommodate some additional controls along the top of the screen.

The other modes use customer view controllers to present a single screen.

Tab bar controller is container type controller.

Modal View Controllers

Modal View Controllers

- A modal view controller is not a specific type of view controller class but is a way of presenting a view controller to the user.
- Most of the time you present view controllers modally in order to gather information from the user or capture the user's attention for some specific purpose. Once that purpose is completed, you dismiss the modal view controller and allow the user to continue navigating through your app.
- When user clicks the plus button to add a new contact, the Contact view controller presents the New Contact view controller modally.
- This creates parent/child relationship between the two view controllers.
- New Contact screen remains visible until the user cancels the operation or provides enough information about the contact that it can be saved to the contact database, at which point the contacts view controller dismisses its child.

UIPickerView

- Hands On Exercise to use UIPickerView
- UIPickerView is one of the many examples of a class delegating some of the tasks to a delegate object.
- Notice use of delegate and methods supported by delegate and data source delegate.
- Review Class Reference

Class Exercise on Slider and Picker

- Use of Slider and Label for additional Hands on
- Demo of Picker Control as basis of your next home work

Homework #2

- Build an app that converts one unit of measure to another. For e.g. if user enters KG, provide conversion to Lbs. Other examples are currency exchange, temperature, nautical speed to land speed etc.
- Provide a picker control to choose between several conversion factors. You need to put at a minimum three conversions. For e.g. units of mass, distance and temperature. Feel free to add as many as you would like.
- Use slider control or textfield input box to receive input from users on unit of conversion. You are welcome to use one or two sliders. Analyze why one slider may be enough.
- Use backgrounds pics, launch images and app icons. Web is a wonderful source of .png. Surf around and pick images that are appropriate.
- Past HW Examples
- Homework 2 is due next week on Class #6 - on 9/28.

Pointers to Reading Material

- Material in previous section was from developer.apple.com Apple keeps it's documentation up to date and is a great reference, better than most if not all of the book I have read.
- Search "View Controller Programming Guide" at developer.apple.com and read up their guide. In the latest version they have simplified their description
- Read up on MVC Patterns - wikipedia is great source - http://en.wikipedia.org/wiki/MVC_Pattern

Summary

- MVC - key to user interaction. Important to keep Model, View and Control separate from each other. Connivence of kludging is not worth the mess that app can evolve into.
- Slider, text boxes, picker, and label APIs. Use of keyboard to enter data and use of "Return" in the keyboard to trigger an event.
- HW #2 due on 9/28.