iPhone Programming
CMSC 498i – Spring 2010

View Controllers
Lecture #9 – Chuck Pisula
Today’s Topics

• Using “View Controllers”

• View Controller Feature overview

• Application flow – Chaining view controllers together

• Semester Project – Team Assignments
View Controllers

• View Controllers are a specific type of MVC controller
• Provide more than just MVC coordination
  ▪ View management – manages screenful of content
  ▪ Application flow – makes navigation easy
  ▪ User experience – Implement standard behaviors
  ▪ Override points – For handling system events
  ▪ Performance – Designed with mobile device performance in mind
Class Hierarchy

UIViewController
Supporting Classes

- NSObject
  - UIResponder
  - UIView
    - UITabBar
    - UINavigationBar
  - UINavigationItem
    - UIBarButtonItem
  - UIBarButtonItem
    - UITabBarItem
    - UIBarButtonItem
Supporting Classes

- NSObject
  - UIResponder
  - UIView
    - UITabBar
  
  - UINavigationItem
    - UITabBarItem
  
  - UIBarButtonItem
    - UIBarButtonItem
Normal Usage

- Custom View Controllers
  - Will have one for each screen
  - You Will Subclass
    - UIViewController
    - UITableViewController

- Use and compose with others
  - UITabBarController
  - UINavigationViewController
Overview

Tour of View Controllers
UIViewController

- Manages one view
- Subclass to work with your own UI
**UIViewController**

- Multiple views contained within a top level view
- Top level view managed by a view controller
UIViewController

- Multiple views contained within a top level view
- Top level view managed by a view controller
UIViewController

- Lazy, on-demand view creation
  - Creates view as needed
  - Deallocates view when possible

- Rotation control and events

- Memory management
  - Low memory warnings

- Editing behavior and override points

- Display related events

- Container controller (tab, navigation) related properties
UITableViewController

• Subclass to work with your data
UITableViewController

- UITableView managed by a view controller
UITableViewController

- **UITableView** managed by a view controller
- View controller contained in chain of other controllers
UITableViewController

- “view” property expected to be a `UITableView`
- Acts as data source, and delegate of the table
  - Override by setting the data source, delegate property yourself
- Very simple class, provides a couple things
  - Standard user experiences
    - Display “flash” when table first becomes visible
    - Automatically deselect during back navigation
  - Coordinates Editing UI
  - Configures view autosizing mask to allow resizing
Mini Summary…

- UIViewController and UITableViewController
- Meant for subclassing
- Nodes within an application’s flow
  - Concerned with data display, not application flow
UITabBarController

- Provides view switching functionality
- Each child view controller has its own view
UITabBarController

- Provides view switching functionality
- Each child view controller has its own view
UITabBarController

- Provides view switching functionality
- Each child view controller has its own view
UITabBarController

- Provides view switching functionality
- Each child view controller has its own view
UITabBarController

- View switching
- Automatic “More” list creation for overflow
- Allow user customization of order
- Delegate API to control certain behaviors
  - Selection
  - Customization
- Typically the top level controller in an application
  - Each tab containing either a UIViewController, or UINavigationController
UITabBarItem

• Information needed to display buttons in tab bar

• Creation
  ▪ System Items – (UITabBarItemSystemItemFavorites, Contacts, ...)

  ![System Items Icon]

  ▪ Using any UIImage
    ▪ UITabBar automatically creates gray, “shiny” blue versions

• Badges
  ![Badges Icon]
UIApplication

UINavigationController

- Primary application flow – push / pop

“nav stack”

Navigation Controller → View Controller → ... → View Controller

[Image of iPhone with a contact list]
UINavigationController

- Primary application flow – push / pop
UINavigationController

- Primary application flow – push / pop
Navigation Parts
Navigation Parts

Navigation Bar
Navigation Parts

Navigation Bar

Toolbar
Navigation Parts

Navigation Bar

Top Controller’s view

Toolbar
**UIViewController**

- Manages hierarchal application flow
  - Usually contained within a `UITabBarController`
- Navigation Stack
  - **Root Controller** - bottom of the stack
  - **Top Controller** - currently show view controller
  - Operations – push, pop, set stack
- Customizable components in navigation / toolbar
- Dynamic per-node toolbar
  - TabBar is fixed and does not change with each selection
**UINavigationItem**

- Encapsulates navigation bar display information
  - UINavigationController consults to configure its display
- Each UIViewController has a navigationItem
- By default you don’t need to do anything
  - Title automatically configured to match view controller’s title
    - Main Title
    - Back Button Title
- Use if you want to customize display
  - Title, left / right buttons
UIBarButtonItem

- Used in UIToolbar and UINavigationBars

- Creation
  - With any custom view
  - Buttons
    - System Items – (e.g. UIBarButtonItemSystemItemEdit, Compose, ...)
    - Using a UIImage
    - With an NSString title

- Configurable target, action for buttons
Navigation Bar

- Views provided by UIBarButtonItem's view property
- Title provided as string or custom view
Navigation Bar

- Views provided by UIBarButtonItem’s view property
- Title provided as string or custom view

```objective-c
UINavigationItem *navItem = someViewController.navigationItem;

navItem.leftBarButtonItem = ...;
navItem.rightBarButtonItem = ...;

navItem.titleView = // some view
navItem.title = // some string
```
Navigation Bar

- Current View Controller Provides
  - Left / Right Button
    - Any UIBarButtonItem (e.g. edit, save, ...)
    - Avoid custom Left Button where “back” buttons may be needed
  - Title Area
    - Typically just a string
    - Can provide a custom view instead (e.g. segmented control...)
Navigation Bar

• Current View Controller Provides
  ▪ Left / Right Button
    ▪ Any UIBarButtonItem (e.g. edit, save, ...)
    ▪ Avoid custom Left Button where “back” buttons may be needed
  ▪ Title Area
    ▪ Typically just a string
    ▪ Can provide a custom view instead (e.g. segmented control...)

• Previous View Controller Provides
  ▪ Usually a “back button” – title from previous view controller
Toolbar

- Views provided using `UIBarButtonItemAtItem`'s view property
Mini Summary...

- **UITabBarController** and **UINavigationController**
- Not meant for subclassing
  - Presence of `delegate` API is often a good indicator of this...
    ```
    // UITabBarController
    @property(nonatomic,assign) id<UITabBarControllerDelegate> delegate;
    ```
    ```
    // UINavigationController
    @property(nonatomic, nonatomic, assign) id<UINavigationControllerDelegate> delegate;
    ```
- Handle most of your application flow
  - View switching
  - Hierarchal navigation
View Autoresizing

• Reasons a view’s height can change
  ▪ Incoming phone call – double height status bar
  ▪ Rotation

• Other reasons to make view size flexible
  ▪ Can install view controller in navigation, or tab bar controller
  ▪ Area available to view controller based on where it lives
    ▪ Navigation bar takes up space
      ▪ Can specify a title prompt and take up even more…
    ▪ Tab bar takes up space
Combinations

Some Standard Configurations
Tab & Navigation

• **UITabBarController** top level item
  - Enough items for “more” button bar item to automatically be created

• Each tab has a **UINavigationController**

• Chain of **UITableViewControllers**
Tab & Navigation

- UINavigationController top level item
- Bottom Bar is a UIToolbar
  - No UITabBarController
- Chain of UITableViewControllers
- Leaf node is UIViewController managing a custom view that displays email text
Subclassing

Creating View Controllers
Subclassing

- Choose view controller to subclass

```objective-c
@interface PersonViewController : UIViewController {
    @property (retain, readwrite) Person *person;
}

- (id)initWithPerson:(Person *)person;
```

- Provide custom methods for data passing

- IBOutlets and IBActions
  - Hook up outlets to UI, and action to your controller code
Subclassing - XIB Based

- Provide XIB filename when you call

```swift
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil;
```

- Or... If a nil filename is passed, `-nibName` is used

```swift
@property(nonatomic, readonly, copy) NSString *nibName;
```

- Set the title property in Interface Builder

- **DO NOT** implement `-loadView`
```swift
-(IBAction)someAction:(id)sender {
    controller = [[[PersonViewController alloc] initWithPerson:...]];
}

@implementation PersonViewController

- (NSString *)nibName {
    return @"PersonViewController";  // note, no "\.xib"
}

- (id)initWithPerson:(Person *)p {
    // Since nil is specified as nibName, -nibName property is used
    if ((self = [self initWithNibName:nil bundle:nil])) {
    }
    return self;
}

- (id)initWithNibName:(NSString *)n bundle:(NSString *)b {
    if ((self = [super initWithNibName:n bundle:b])) {
    }
    return self;
}

@end
```
Nib Loading Details

• First, some review…
Review

Application Delegate

Manage application lifecycle and events

ToDoListViewController

Provide cells for table
Manage array of events
@implementation iNeedToDoAppDelegate

- (void)applicationDidFinishLaunching:(UIApplication *)app {
    [window addSubview:[navigationController view]];
    [window makeKeyAndVisible];
}
....
@end

Application Delegate

Manage application lifecycle and events

ToDoListViewController

Provide cells for table
Manage array of events

Review
`@implementation UINavigationController`

- (UIView *)view {
  if (!_view) {
    [self loadView];
  }
  return _view;
}

`@end`

`@implementation iNeedToDoAppDelegate`

- (void)applicationDidFinishLaunching:(UIApplication *)app {
    [window addSubview:[navigationController view]];
    [window makeKeyAndVisible];
}

`@end`

`@implementation UINavigationController`

- (UIView *)view {
  if (!_view) {
    [self loadView];
  }
  return _view;
}

- (void)loadView {
  // Create top level container view with nav bar
  rootController = // set to ToDoListViewController in NIB
  [topView addSubview:[rootController view]];
}

@end
Review

rootController = // set to ToDoListViewController in NIB
[topView addSubview:[rootController view]];
}
@end

@implementation UIViewController

// ToDoListViewController is a UIViewController!
-(void)loadView {
    [NSBundle loadNibNamed: [self nibName]
        owner: self
        options:nil];
}
@end

Application Delegate

Manage application lifecycle and events

ToDoListViewController

Provide cells for table
Manage array of events
Provide cells for table

Manage array of events

Manage application lifecycle and events

Review
File’s Owner

- Interface Builder proxy object
  - Already exists! Other objects are instantiated by NIB loading
  - File’s Owner is an external reference passed to IB that is dynamically connected later
Nib Loading Details

- Object initialized by NIB loading are sent -awakeFromNib
- “Proxy” objects like file’s owner are NOT sent -awakeFromNib
  - Need to figure out your own place...
  - UIViewController subclasses should use -viewDidLoad

@implementation PersonViewController

- (void)initWithNibName:(NSString *)n bundle:(NSString *)b {
  if ((self = [super initWithNibName:n bundle:b])) {
    // IBOutlets will be *nil* here! NIBs are loaded lazily!
    // -loadView will call -viewDidLoad when done...
    return self;
  }
}

- (void)viewDidLoad {
  // Yay, my IBOutlets are all connected!
}
@end
Subclassing – No XIB

• Don’t
  ▪ Pass a XIB filename to `-initWithNibName`:
  ▪ Return a value from `-nibName`
    ▪ You could, but should return nil…

• Do
  ▪ Create your view in `-(void)loadView`
  ▪ Set the `title` property
    ▪ Call setter or override `-title`
@implementation PersonViewController

- (id)initWithPerson:(Person *)p {
    // Since nil is specified as nibName, -nibName property is used
    if ((self = [self initWithNibName:nil bundle:nil])) {
        return self;
    }
}
@implementation PersonViewController

- (id)initWithPerson:(Person *)p {
    // Since nil is specified as nibName, -nibName property is used
    if ((self = [self initWithNibName:nil bundle:nil])) {
        return self;
    }
}

- (void)initWithNibName:(NSString *)n bundle:(NSString *)b {
    // nibName better be *nil* !!!
    if ((self = [super initWithNibName:n bundle:b])) {
        self.title = @"Info";
        return self;
    }
}

- (void)loadView {
    // Create the view, and the set the view controller’s view property
    SomeAwesomeView *sav = [[SomeAwesomeView alloc] initWithFrame];
    self.view = sav;
}

@end
Subclassing

Standard Overrides
Subclassing - Overrides

- **viewWillAppear:**
  - Perform tasks associated with presenting your view
  - E.g. update a display value, coordinate status bar look, etc.

- **viewDidAppear:**
  - Called when the view has been fully transitioned onto the screen
  - E.g. UITableView flashes its scroll bars
Subclassing - Overrides

• -viewDidDisappear:
  - Your view is dismissed, covered or hidden
  - Should stop timers, display updates, etc...

• -viewWillDisappear:
  - May want to stop timers and updates at this point
  - May want to save state to disk
Subclassing Overrides

- **-viewDidLoad**
  - Called after a view is loaded from XIB, or by `-loadView`

- **-viewDidUnload**
  - Called whenever UIKit deems it necessary to release the view without deallocating the controller
  - You can release any data easily recreated
  - If your data is expensive to recreate (often the case with caches), maybe wait until `-didReceiveMemoryWarning` to release
Laziness...

- Notice, view creation is very “lazy”
  - Requires some coordination because of laziness
  - Benefit is memory saving

- (UIView *)view {
  if (!view_) {
    // load the “view_
    [self loadView];
  }
  return view_;
}
Laziness...

- Notice, view creation is very “lazy”
  - Requires some coordination because of laziness
  - Benefit is memory saving

```c
-(NS MUTABLE DICTIONARY *) mapTable {
    if (!mapTable_) {
        mapTable_ = [[NS MUTABLE DICTIONARY alloc] init];
    }
    return mapTable_;
}
```
Lazy Loading

- Example –UITabViewController
  - tab view created and set up with list of view controllers
  - however, only one view exists
  - view is created on demand when switching to selected tab

- Example –UINavigationController
  - Only views on screen must be kept around
  - For good performance, visited items views are kept around
  - …Unless a “memory notification” comes in…
Demo

View Controllers
Next Lecture

• View Controllers In Depth
  ▪ Application Flow
    ▪ Implementing common use cases
    ▪ Code showing how to setup a chain of view controller
  ▪ UINavigationController bells and whistles
  ▪ “Modal” view controllers
  ▪ Other advanced features
Reading

• “View Controller Programming Guide For iPhone OS”
  ▪ ViewControllerPGforiPhoneOS.pdf
  ▪ Read “About View Controllers” – p.11 - 20
  ▪ Read “Custom View Controllers” – p.21 - 52

• iPhone OS Reference Library
  ▪ UIViewController Class Reference
Term Project

Team Assignments
Term Projects

- 10 minutes to meet now
- Get organized and plan times to meet
  - Begin brainstorming ideas
  - **Proposal Due**: March 12th, 11:59 pm