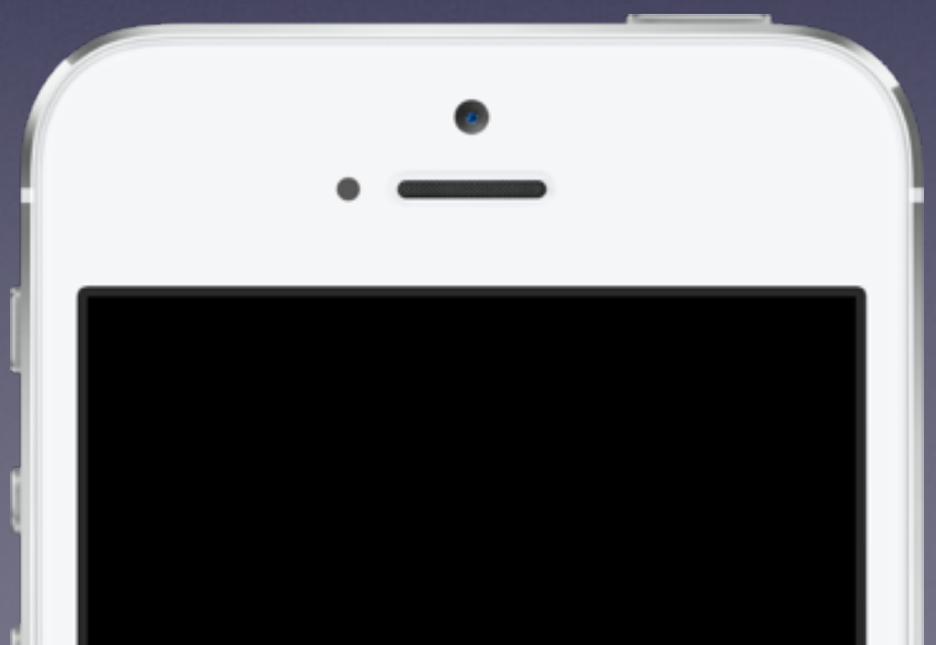
COMS W3101: Programming for iOS

Michael Vitrano



View Controllers, part 2

- Presenting and Dismissing View Controllers
- Using delegation for communication between View Controllers
- UINavigationController
- Displaying and editing text
- Demo!

Presenting and Dismissing View Controllers

- Each of our View Controllers manages the display of information for one "screen"
- We don't want to cram all of an apps functionality into one screen
- The transition between different views helps inform the user that the app is moving between different sets of functionality

Presenting and Dismissing View Controllers

- View controllers are arranged in a hierarchy that starts with the **rootViewController** of your app's window
  - As each VC presents another, it is pushed onto the top of the stack
  - Each VC has a **presentingViewController** and **presentedViewController** property that stores its neighbors in the VC stack
- Presenting a VC on this stack is known as presenting it "modally"

Presenting a VC

- To present a new VC on the stack, we invoke the following method on the topmost VC:

```
- (void)presentViewController:(UIViewController *)vcToPresent
                     animated:(BOOL)animated
                   completion:(void (^)(void))completion
```

- When animated is set to **YES**, the new VC will slide up from the bottom of the screen
- **completion** is a *block* that specifies code to run when the VC is finished being presented
  - We will cover blocks in depth in a later class, you can pass **nil** for this parameter to specify no completion action

Presenting a VC

```
- (void)presentInputViewController
{
    MSVInputViewController *vcToPresent = [[MSVInputViewController alloc] init];
    // Perform additional configuration of vcToPresent here

    [self presentViewController:vcToPresent animated:YES completion:nil];
}
```

Storyboard Segues

- Storyboards are the technology that we have been using thus far to design our user interfaces
- They are capable of managing the interaction between multiple view controllers, which we will see in the demo later today
- We set up the relationships between view controllers in storyboards with **segues**, which are named transitions between two view controllers

Storyboard Segues

- Storyboard segues are represented in code with instances of the **UIStoryboardSegue** and have three important properties:
  - **identifier**, **sourceViewController**, and **destinationViewController**
- When the action which triggers the segue is performed, we are given an opportunity to customize the segue with the following method:

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue
                 sender:(id)sender
```

  - This method is called on the source view controller in the segue

Dismissing a VC

- Dismissing a VC is done with the following method:

```
- (void)dismissViewControllerAnimated:(BOOL)animated
                           completion:(void (^)(void))completion;
```

- Calling this method on the *presenting* VC will dismiss the VC it presented
  - If called on a VC whose presented VC is not on the top of the VC hierarchy, all VCs above the VC on which the method was called will be dismissed
- Calling this method on a VC that does not have a presented VC will result in the message being forwarded to its own presenting VC

Communicating between VCs

- Presenting a new VC to collect input from the user is a common design pattern
  - We often need to get the collected information from the presented VC to the presenting VC
    - The presented VC should not know nor care about who is receiving the information it collects
- The **delegation** design pattern is used to solve this problem

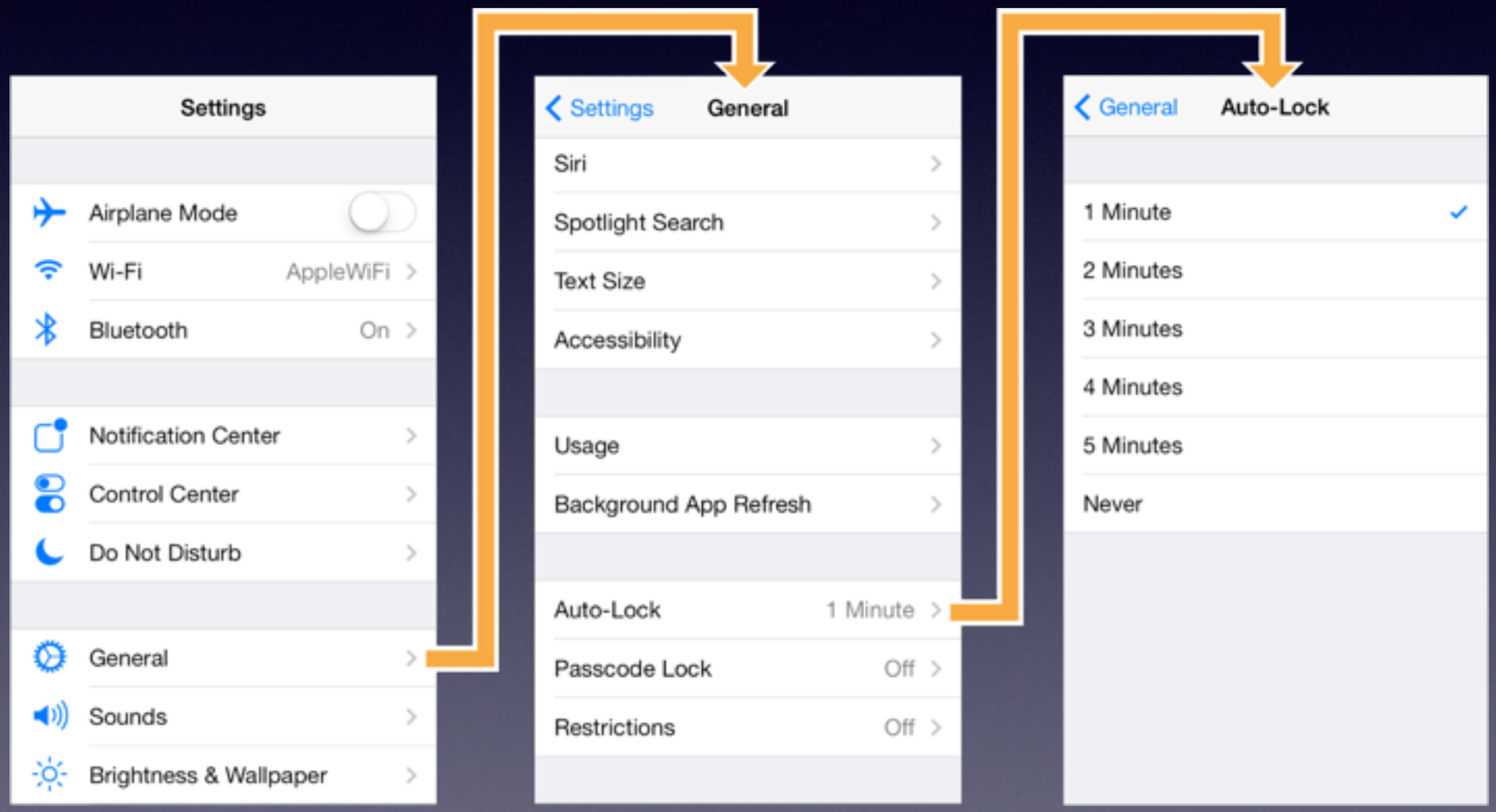
Delegation

- A **delegate** is an object that works on behalf of or in coordination with another object
- To specify the responsibilities of a delegate, we define a **protocol** which enumerates the methods that the delegate is expected to implement
- An object must always maintain a **weak** reference to its delegate to prevent a retain-cycle

Delegation

```
#import <UIKit/UIKit.h>

@class MSVInputViewController;

@protocol MSVInputViewControllerDelegate <NSObject>

- (void)inputController:(MSVInputViewController *)inputController
     didFinishWithText:(NSString *)text;

- (void)inputControllerDidCancel:(MSVInputViewController *)inController;

@end

@interface MSVInputViewController : UIViewController

@property (nonatomic, weak) id<MSVInputViewControllerDelegate> delegate;

@end
```

UINavigationController



UINavigationController

- Instances of the **UINavigationController** class manage the navigation between a hierarchical set of view controllers
- Navigation controllers are commonly the rootViewController of your apps window and help manage the navigation between your app's different VCs
- The set of VCs managed by a navigation controller is known as the **navigation stack**
  - To push a VC onto the stack, you call **pushViewController:animated:** on your navigation controller
  - To remove the pop the top VC of the stack, you call **popViewControllerAnimated:**
- Navigation controllers are container view controllers meaning that they embed the the VCs in the navigation stack within their view

UINavigationController

- Each navigation controller has a navigation bar that is used to display the current VC's title and navigation controls
  - Navigation bars are instances of **UINavigationBar**
  - When a VC is not the only member of a navigation stack, there will be a back button on the left side of the navigation bar which will pop the topmost VC off the stack when pressed
- The contents of the navigation bar are provided by the current VC's **navigationItem** property
  - Navigation items are instances of **UINavigationItem**
  - Setting the **title** property on a VC will change the title displayed in its navigation item
  - You can set instances of **UIBarButtonItem** to the **leftBarButtonItem** or **rightBarButtonItem** properties of a navigation item to add custom buttons

Displaying and Editing Text

- **UILabel** is used to display non-editable text
- Instances of **UITextView** are used to display and optionally edit multiline text blocks in a scrollable view
- There is also **UITextField**, which can be used to get a single line's worth of text from the user