Designing iPhone Applications

448460-1
Fall 2011
10/20/2011
Kyoung Shin Park
Multimedia Engineering
Dankook University

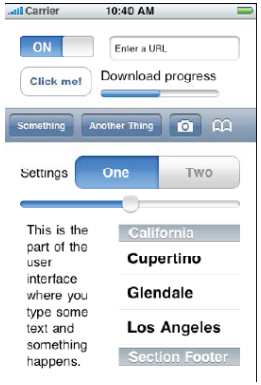
Overview

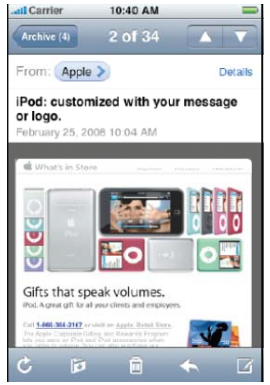
- Designing iPhone Applications
- Model-View-Controller (Why and How?)
- View Controllers
- Navigation Controllers
- Tab Bar Controllers
- Combining Approaches

Designing iPhone Applications

Organizing Content

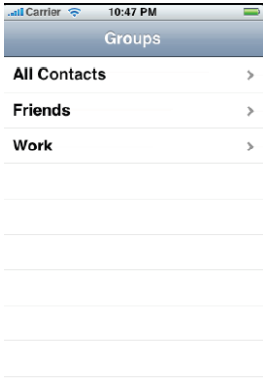
- Focus on your user's data
- One thing at a time
- Screenfuls of content





Patterns for Organizing Content

- Navigation Bar
  - Hierarchy of content
  - Drill down into greater detail



- Tab Bar
  - Self-contained modes

Model-View-Controller (Why and How?)

Why Model-View-Controller?

- Clear responsibilities make things easier to maintain
- Avoid having one monster class that does everything
- Separating responsibilities also leads to **reusability**
- By minimizing dependencies, you can take a model or view class you've already written and use it elsewhere
- Think of ways to write less code

Model

- Not aware of views or controllers
- Typically **the most reusable**
- Communicate generically using
  - **Key-value observing**
  - **Notifications**

View

- Not aware of controllers, may be aware of relevant model objects
- Also **tends to be reusable**
- Communicate with controller using
  - **Target-action**
  - **Delegation**

Controller

- Knows about model and view objects
- The brains of the operation
- Manages relationships and data flow
- **Typically application-specific, so rarely resuable**

View Controllers

Problem: Managing a Screenful

- Controller manages views, data and application logic
- Applications are made of many of these controllers
- Would be nice to have a well-define starting point
  - UIView for views
  - Common language for talking about controllers

Problem: Building Typical Applications

- Some application flows are very common
  - Navigation-based
  - Tab bar-based
  - Combine the two
- Don't reinvent the wheel
- Plug individual screens together to build an application

UIViewController

- Basic building block
- Manages a screenful of content
- Subclass to add your application logic
  - Create **"your" own UIViewController subclass** for each screenful
  - Plug them together using existing **composite view controllers**

Your View Controller Subclass

```
#import <UIKit/UIKit.h>
@interface MyViewController: UIViewController {
    // a view controller will usually manage views and data
    NSMutableArray *myData;
    UILabel *myLabel;
}

// expose some of its contents to clients
@property (readonly) NSArray *myData;

// respond to actions
-(void)doSomeAction: (id)sender;
```

The "View" in "View Controller"

- UIViewController superclass has a view property
  - @property (retain) UIView *view;
- Loads lazily
  - On demand when requested
  - Can be purged on demand as well (low memory)
- Sizing and positioning the view?
  - Depends on where it's being used
  - Don't make assumptions, be flexible

When to call –loadView?

- Don't do it!
- Cocoa tends to embrace a lazy philosophy
  - Call **–release** instead of -dealloc
  - Can **–setNeedsDisplay** instead of –drawRect
- Allows work to be deferred or coalesced
  - Performance!

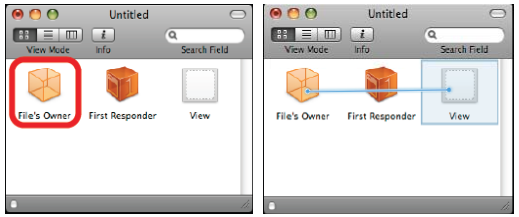
Creating Your View in Code

- **Override –loadView**
  - Never call this directly
- Create your views
- Set the view property
- Create view controller with –init

```
// subclass of UIViewController
-(void)loadView
{
    MyView *myView = [MyView alloc] initWithFrame:frame];
    self.view = myView; // view controller owns the view
    [myView release];
}
```

Creating Your View with Interface Builder

- Layout a view in Interface Builder
- File's owner is view controller class
- Hook up view outlet
- **Create view controller with –initWithNibName:bundle:**



View Controller Lifecycle

-initWithNibName:bundle: & -viewDidLoad

```
// UIViewController's designated initializer
-(id)initWithNibName: (NSString *)nibName
                bundle: (NSBundle *)bundle {
    self = [super init ...];
    if (self) {
        myData = [[NSMutableArray alloc] init];
        self.title = @"Foo";
    }
    return self;
}
-(void) viewDidLoad {
    // your view has been loaded, customize it here if needed
    myLabel.titleLabel.text = @"Test";
}
```

-viewWillAppear: & -viewWillDisappear:

```
// this method gets called every time the view appears on screen
-(void) viewWillAppear: (BOOL)animated {
    [super viewWillAppear: animated];
    // your view is about to show on the screen
    [self beginLoadingDataFromTheWeb];
    [self startShowingLoadingProgress];
}
// this method gets called every time the view disappears on screen
-(void) viewWillDisappear: (BOOL)animated {
    [super viewWillDisappear: animated];
    // your view is about to leave the screen
    [self rememberScrollPosition];
    [self saveDataToDisk];
}
```

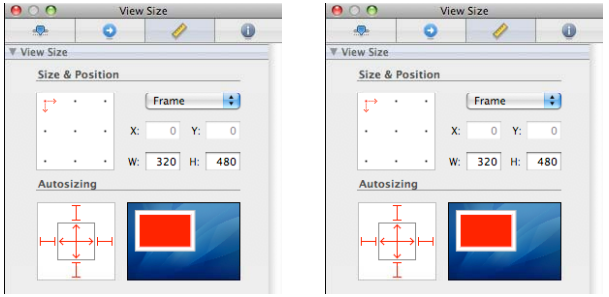
-shouldAutorotateToInterfaceOrientation:

```
// reacting to device rotation
// UIViewController's view is allowed to flip around if the device is
turned upside down
// There is also UIInterfaceOrientationLandscapeLeft and Right
 // This view controller only supports portrait
-(BOOL) shouldAutorotateToInterfaceOrientation:
        (UIInterfaceOrientation)interfaceOrientation {
   return (anOrientation == UIInterfaceOrientationPortrait);
}
// This view controller supports all orientations except for upside-dow
-(BOOL) shouldAutorotateToInterfaceOrientation:
        (UIInterfaceOrientation)interfaceOrientation {
   return (anOrientation != UIInterfaceOrientationPortraitUpsideDown);
}
```

Autoresizing Your Views

```
view.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                        UIViewAutoresizingFlexibleHeight;

view.autoresizingMask = UIViewAutoresizingFlexibleWidth |
                        UIViewAutoresizingFlexibleTopMargin;
```



View Controller Initialization Recap

- 4 places to initialize things in View Controller subclasses
  - **-(id) initWithNibName: (NSString *)nib bundle: (NSBundle *)bundle**
  - **-(void)awakeFromNib;**
  - **-(void)viewDidLoad;**
  - **-(void)viewWillAppear: (BOOL) animated;**

View Controller Initialization Recap

- Designated Initializer
  - **Usually only for things that have to be initialized** for your view controller to even "make sense"
  - Often thought of as the place to initialize things having to do with your model
  - Definitely not for initializing things in your View **(some UI-related things ok like self.title)**
- -awakeFromNib
  - **Same purpose (generally) as your designated initializer**
  - This is called on every object that comes out of a .xib file
  - Usually you want your VC to work when it is alloc/inited or if it comes from a .xib
  - So create a method (e.g. setup) and call it from **initWithNibName:bundle:** and **awakeFromNib**

View Controller Initialization Recap

- -viewDidLoad
  - **This is the best place to put non-geometry-related initialization code which pertains to your View**
  - You might even add some views to your hierarchy in this method (stuff you couldn't do in IB)
  - Wouldn't be out of the question to put some Model initialization code here, but theoretically this method could be called multiple times, so don't re-initialize something already initialized
  - It's not totally unheard of to have Views which are loaded, but then never appear on screen
  - So consider viewWillAppear: for some things, but maybe check to avoid multiple initialization
- -viewWillAppear
  - **If you have initialization that depends on your View's bounds being set, you must do it here** (and not in viewDidLoad)

Controller of Controllers

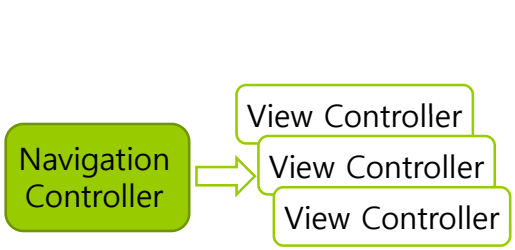
Controller of Controllers

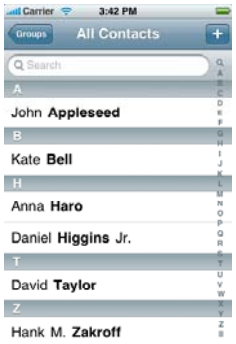
- Special View Controllers that manage a collection of other MVCs
- **UINavigationController**
  - Manages a hierarchical flow of MVCs and presents them like a "stack of cards"
  - Commonly used on the iPhone
- **UITabBarController**
  - Manages a group of independent MVCs selected using tabs on the bottom of the screen
- **UISplitViewControllers**
  - Side-by-side, master-detail arrangement of two MVCs
  - iPad only

Navigation Controller

UINavigationController

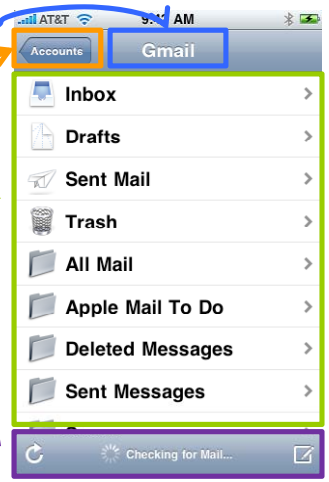
- **Stack of view controllers**
- Navigation bar





How It Fits Together

- Top view controller's **title**
  - NSString
- Previous view controller's **title**
  - NSString
- Top view controller's **view**
  - UIView
- Top view controller's **toolbar items** (iPhone OS 3.0)
  - NSArray of UIBarButtonItems



Modifying the Navigation Stack

- **Push** to add a view controller

```
-(void) pushViewController: (UIViewController *)viewController
                  animated: (BOOL) animated;
```

- **Pop** to remove a view controller

```
-(UIViewController *)popViewControllerAnimated: (BOOL)animated
```

- **Set** to change the entire stack of view controller

```
-(void)setViewControllers: (NSArray *)viewControllers
                 animated: (BOOL)animated;
```

Pushing Your First View Controller

```
-(BOOL)application: (UIApplication *)application
        didFinishLaunchingWithOptions: (NSDictionary *)launchOptions
{
    // create a navigation controller
    navController = [[UINavigationController alloc] init];
    // push the first view controller on the stack
    [navController pushViewController:firstViewController
                             animated:NO];
    // add the navigation controller's view to the window
    [window addSubview:navController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

In Response to User Actions

- **Push** from within a view controller on the stack

```
-(void) someAction: (id)sender
{
    // potentially create another view controller
    UIViewController *viewController = ...;
    [self.navController pushViewController:viewController
                                  animated:YES];
}
```

- Almost never call **pop** directly!
  - It automatically invoked by the back button.
  - But it can happen programmatically as well using

```
-popViewControllerAnimated: (BOOL)animated;
```

Application Data Flow

A Controller for Each Screen

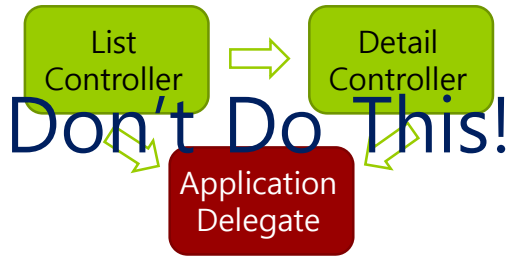
List Controller ⇨ List Controller ⇨ Detail Controller

Connecting View Controllers

- Multiple view controllers may need to share data
- One may need to know about what another is doing
  - Watch for added, removed or edited data
  - Other interesting events

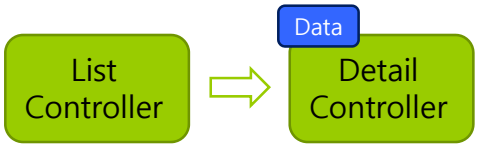
How Not To Share Data

- Global variables or singletons
  - **This includes your application delegate!**
- Direct dependencies make your code less reusable
  - And more difficult to debug & test



Best Practices for Data Flow

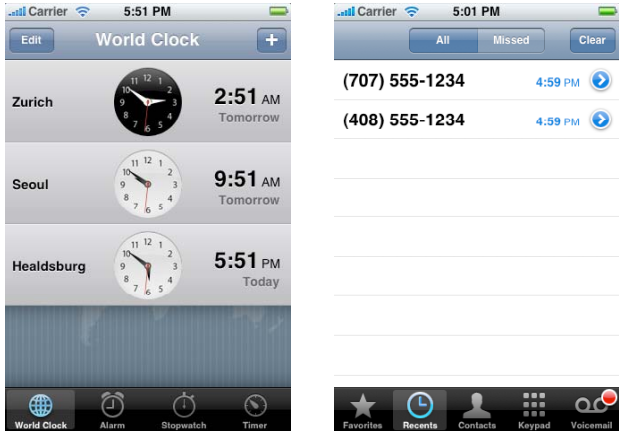
- Figure our exactly what needs to be communicated
- Define **input parameters** for your view controller
- For communicating back up the hierarchy, **use loose coupling**
  - **Define a generic interface for observers (like delegation)**



Customizing Navigation

Customizing Navigation

- Buttons or custom controls
- Interact with the entire screen



UINavigationItem

- Describes appearance of the navigation bar
  - Title string or custom title view
  - Left & right bar buttons
  - More properties defined in UINavigationBar.h
- **Every view controller has a navigation item** for customizing
  - Displayed when view controller is **on top of the stack**

Navigation Item Ownership

Displaying a Title

- UIViewController already has a **title** property
  - @property(nonatomic, copy) NSString * title
- Navigation item inherits automatically
  - Previous view controller's title is displayed in back button


```
viewController.title = @"Detail";
```

Left & Right Buttons

- **UIBarButtonItem**
  - Special object, defines appearance & behavior for items in navigation bars and toolbars
- Display a string, image or predefined system item
- **Target + action** (like a regular button)

Text/System Bar Button Item

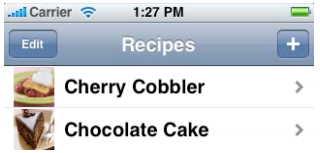

```
-(void) viewDidLoad {
    UIBarButtonItem *fooBtn = [[UIBarButtonItem alloc]
            initWithTitle: @"Foo"
            style:UIBarButtonItemStyleBordered
            target:self
            action:@selector(foo:)];
     self.navigationItem.leftBarButtonItem = fooBtn;
     UIBarButtonItem *addBtn = [[UIBarButtonItem alloc]
            initWithBarButtonSystemItem: UIBarButtonSystemItemAdd
            style:UIBarButtonItemStyleBordered
            target:self
            action:@selector(add:)];
     self.navigationItem.rightBarButtonItem = addBtn;
     [fooBtn release];      [addBtn release];
}
```

Edit/Done Button

- Very common pattern
- Every view controller has one available
  - Target/action already set up

```
self.navigationItem.leftBarButtonItem = self.editButtonItem;

//called when the user toggles the edit/done button
-(void) setEditing: (BOOL)editing animated: (BOOL)animated
{
    // update appearance of views
}
```



Custom Title View

- Arbitrary view in place of the title

```
UISegmentedControl *segmentedControl = ……
self.navigationItem.titleView = segmentedControl;
[segmentedControl release];
```

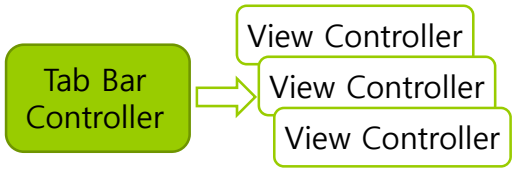
Back Button

```
self.title = @"Hello there";
UIBarButtonItem *heyBtn = [[UIBarButtonItem alloc]
                                    initWithTitle:@"Hey!"
                                    …];
self.navigationItem.backButtonItem = heyBtn;
[heyBtn release];
```

Tab Bar Controllers

UITabBarController

- Array of view controllers
- Tab bar



How It Fits Together

- Selected view controller's view
- All view controller's titles

Setting Up a Tab Bar Controller

```
-(BOOL) application: (UIApplication *)
          didFinishLaunchingWithOptions: (NSDictionary *) options {
    UIViewController *vc1 = ....;
    UIViewController *vc2 = ....;
    // create a tab bar controller
    UITabBarController tabController = [[UITabBarController alloc] init];
    // set the array of view controllers
    tabController.viewController = [NSArray arrayWithObjects:
                                                    vc1, vc2, ...., nil];
    [vc1 release]; [vc2 release];
    // add the tab bar controller's view to the window
    [window addSubview:tabController.view];
    [window makeKeyAndVisible];
    return YES;
}
```

Tab Bar Appearance

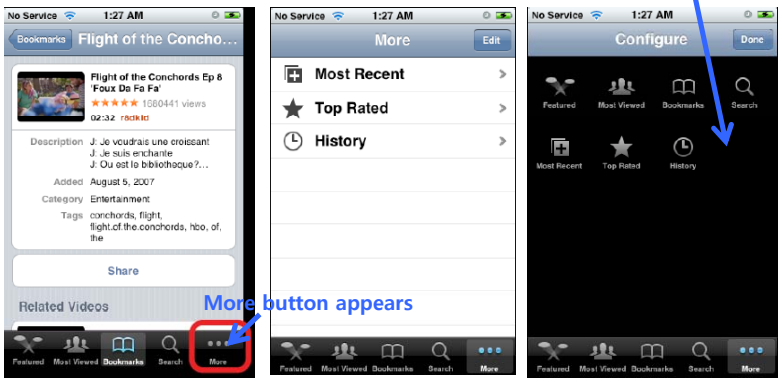
- View controllers can define their appearance in the tab bar
- Each view controller comes with a tab bar item for customizing
- **UITabBarItem**
  - **Title + image** or **system item**

Tab Bar Item


```
-(void) setup // call from initiWit                    awakeFromNib
{
    UITabBarItem *titleAndImageItem = [[UITabBarItem alloc]
          initWithTitle:@"Playlists"
          image:[UIImage imagedNamed:@"music.png]
          tag:0];
    self.tabBarItem = titleAndImageItem;
    UITabBarItem *systemItem = [[UITabBarItem alloc]
          initWithTabBarSystemItem: UITabBarSystemItemBookmarks
          tag:0];
    self.tabBarItem = systemItem;
    [titleAndImageItem release];     [systemItem release];
}
```

More View Controllers

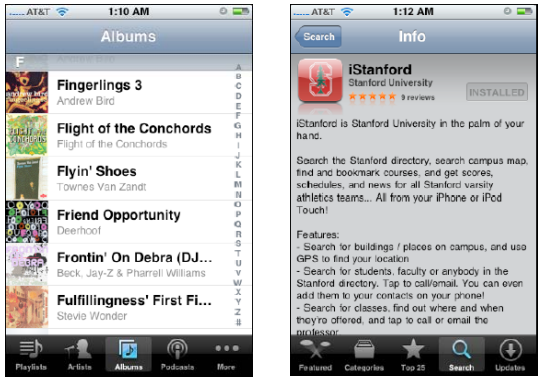
- What happens when a tab bar controller has too many view controllers to display at once?
  - **More tab bar item displayed automatically**
  - Use can navigate to remaining view controllers
  - Customize order



Combining Approaches

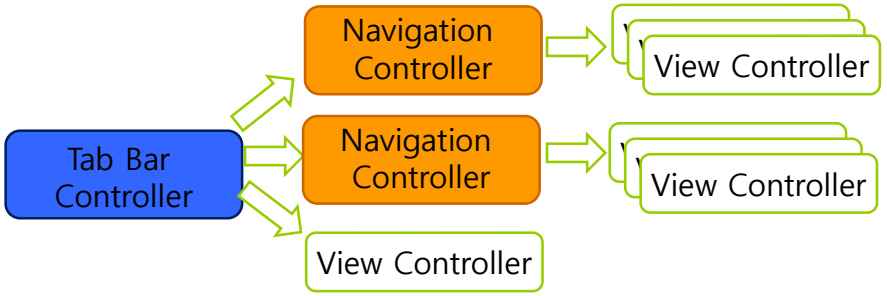
Tab Bar + Navigation Controllers

- Combine UINavigationController & UITabBarController?
  - **Quite common**
  - Multiple parallel hierarchies



Tab Bar + Navigation Controllers

- UINavigationController goes "inside" the UITabBarController
  - Never the other way around



Nesting Navigation Controllers

- Create a tab bar controller

```
tabBarController = [[UITabBarController alloc] init];
```

- Create each navigation controller

```
navController = [[UINavigationController alloc] init];
[navController pushViewController:firstViewController
                         animated:NO];
```

- Add them to the tab bar controller

```
tabBarController.viewControllers = [NSArray arrayWithObject:
                                    navController,
                                    anotherNavController,
                                    someViewController,
                                    nil];
```

Setting Up TabBar+Navigation Controller

```
-(BOOL) application: (UIApplication *)
          didFinishLaunchingWithOptions: (NSDictionary *) options {
  UINavigationController *nav1 = [[UINavigationController alloc] init];
  UINavigationController *nav2 = [[UINavigationController alloc] init];
  [nav1 pushViewController: ...];    [nav2 pushViewController: ...];
  // here, release the view controllers pushed onto the nav1 & nav2
  // create a tab bar controller
  UITabBarController tabController = [[UITabBarController alloc] init];
  // set the array of view controllers
  tabController.viewController = [NSArray arrayWithObjects:
                                  nav1, nav2, nil];
  // add the tab bar controller's view to the window
  [window addSubview:tabController.view];
  [window makeKeyAndVisible];
  return YES;   }
```

Modal View Controllers

- Making a view controller's view appear temporarily
  - An blocking all other "navigation" in application until the user has dealt with this view
- One view controller presents another view controller **modally**
  - Putting up a modal view that asks the user to find an address

```
-(void) lookupAddress { // this might be a target/action method
    AddressLookupViewController *vc =
                        [[AddressLookupViewController alloc] init];
    [self.presentModalViewController:vc animated:YES];
    [vc release];
}
```

  - This method will fill the entire screen with vc's view and **immediately return**. The user will then not be able to do anything except interact with vc's view.

Modal View Controllers

- So when does it all end?
  - It stays this way until someone sends this message to the view controller which put vc up

```
-(void) dismissModalViewControllerAnimated: (BOOL)animated;
```

  - You do NOT send this to vc! You send it to the view controller that presented vc (i.e., the one that implements the method lookupAddress above).
  - Not only that, but vc should NOT send it (since it should not have a "back" pointer to that VC).

Modal View Controllers

- So how is this conundrum resolved? **Delegation**

```
-(void) lookupAddress {
    AddressLookupViewController *vc =
                        [[AddressLookupViewController alloc] init];
    vc.delegate = self;
    [self.presentModalViewController:vc animated:YES];
    [vc release];
}
// (one of) AddressLoopupViewController's delegate method(s)
-(void) addressLookupViewController:
                        (AddressLookupViewController *) sender
                        didSelectAddress: (Address *)anAddress {
    // do something with the address the user selected & dismiss
    [self.dismissModalViewControllerAnimated:YES];
}
```

Modal View Controllers

- How is the modal view controller animated onto the screen?
  - Depends on this property in the view controller that is being put up modally

```
@property UIModalTransitionStyle modalTransitionStyle;
UIModalTransitionStyleConverVertical // slides up and down
UIModalTransitionStyleFlipHorizontal // flips the current vc view
UIModalTransitionStyleCrossDissolve // old fades out
UIModalTransitionStylePartialCurl
```

Modal View Controllers

- What about iPad?
  - Sometimes it might not look good for a presented view to take up the entire screen

```
@property UIModalPresentationStyle modalPresentationStyle;
UIModalPresentationFullScreen
UIModalPresentaitonPageSheet
UIModalPresentationFormSheet
UIModalPresentationCurrentContext
```

References

- Lecture 6 & 7 Slide from iPhone Application Development (Winter 2010) @Stanford University