Stanford CS193p

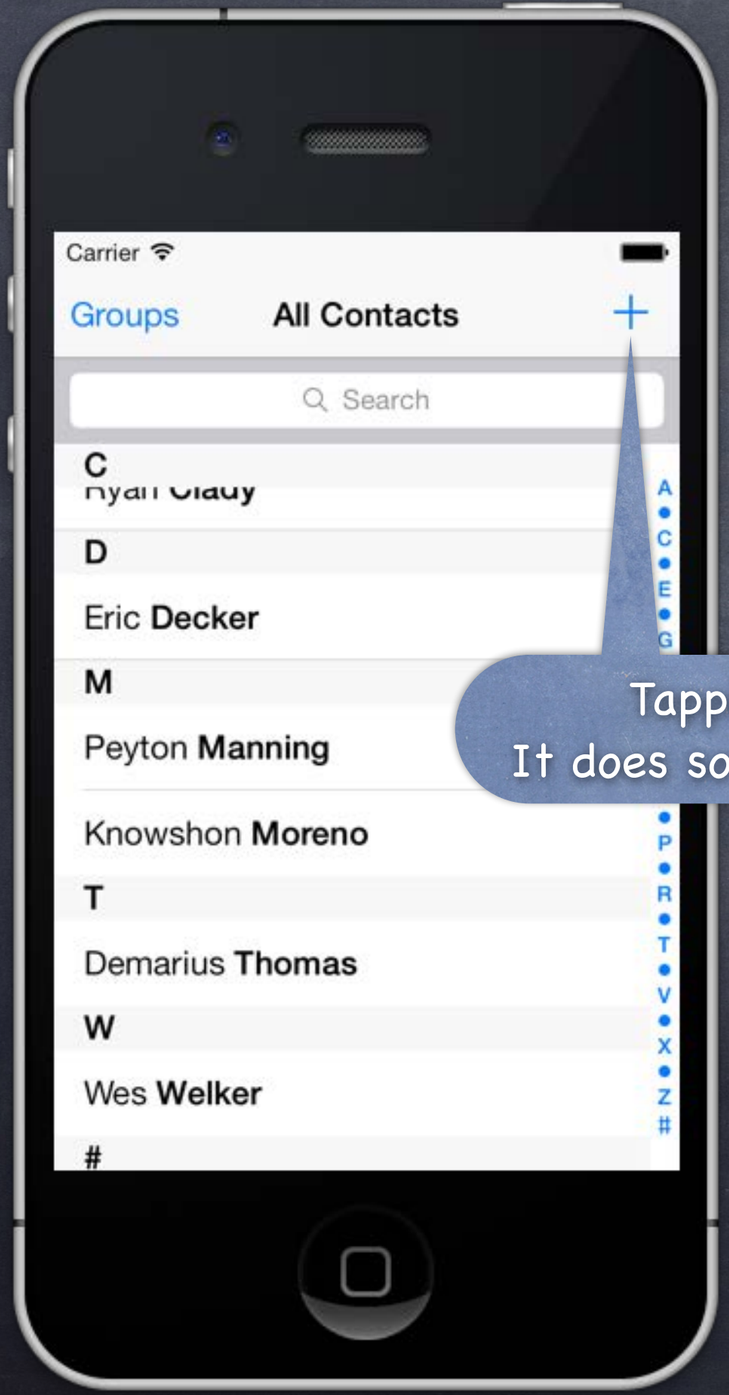
Developing Applications for iOS
Fall 2013-14

Today

- Modal Segues
  Transitioning to a Controller which "takes over your UI" until it's done with the user.
- Text Fields
  How to get text input from the user.
- Alerts and Action Sheets
  Notifying the user and getting "branching decisions" from the user.
- Demo
  Adding a photo taken by the user to Photomania.
- Camera (time permitting)
  And Photo Library.

Modal View Controllers

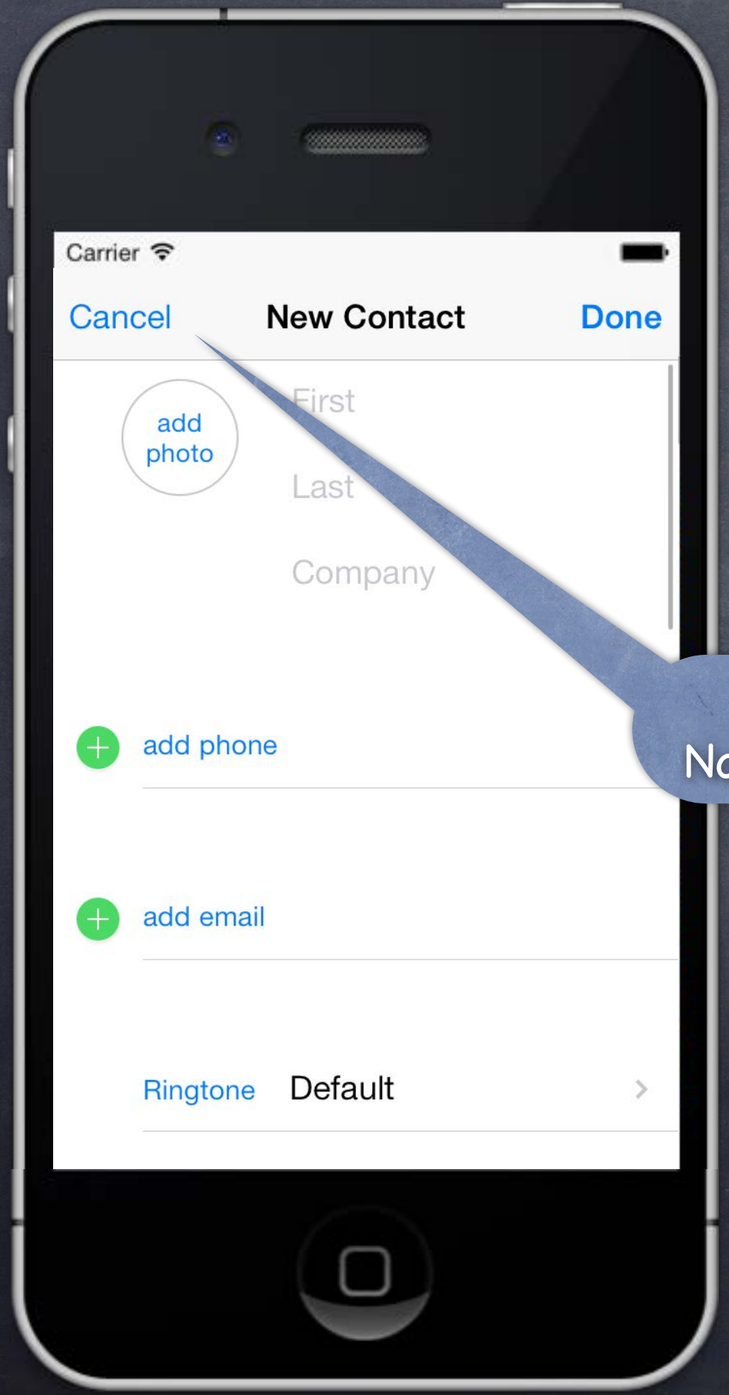
- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Tapping here adds a new contact.
It does so by taking over the entire screen.

Modal View Controllers

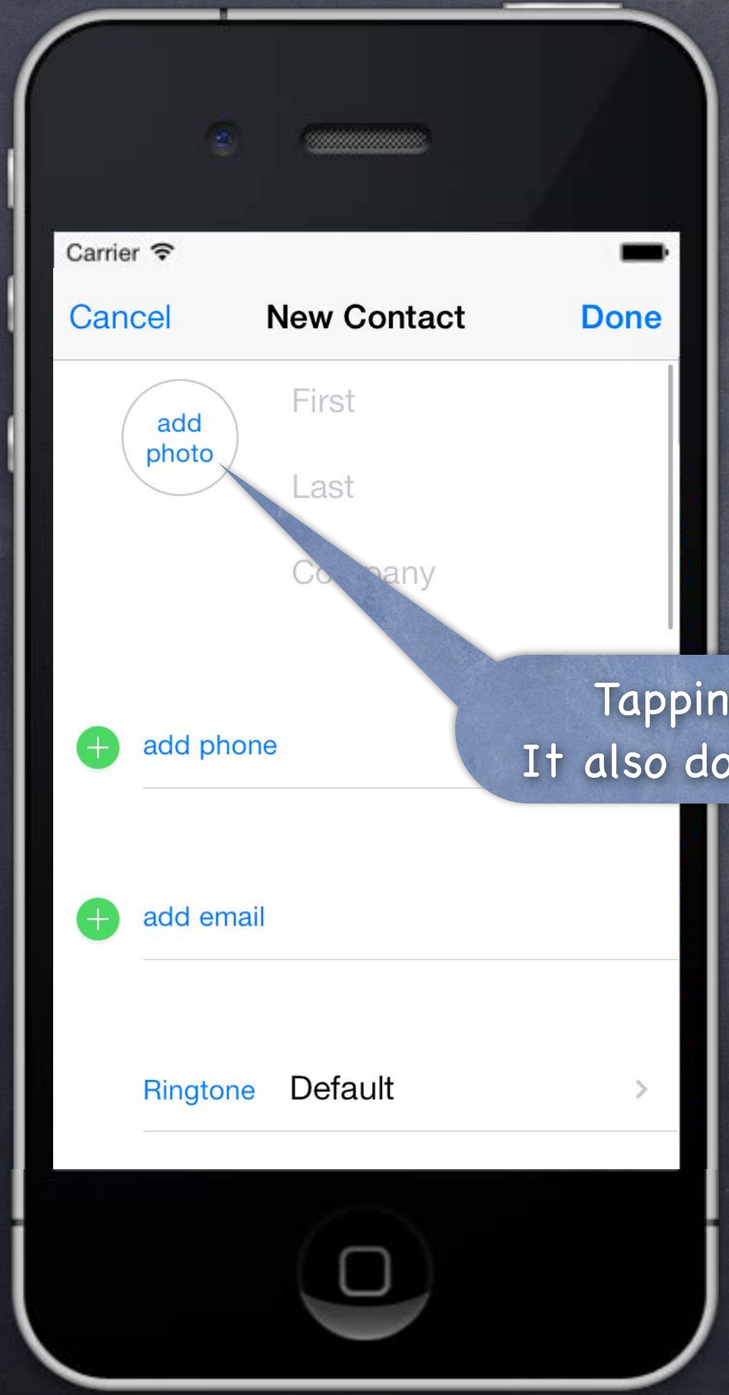
- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



This is not a push.
Notice, no back button (only Cancel).

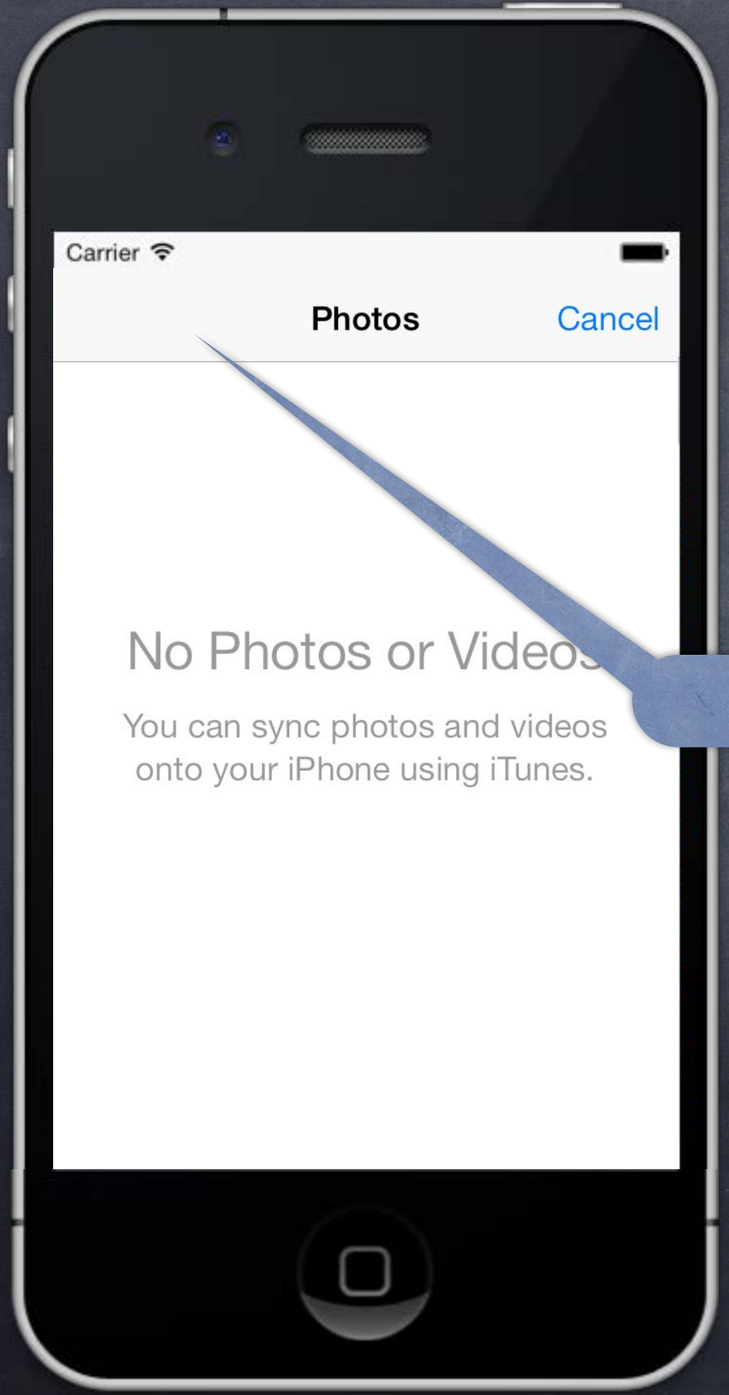
Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Tapping here adds a photo to this contact.
It also does so by taking over the entire screen.

Modal View Controllers

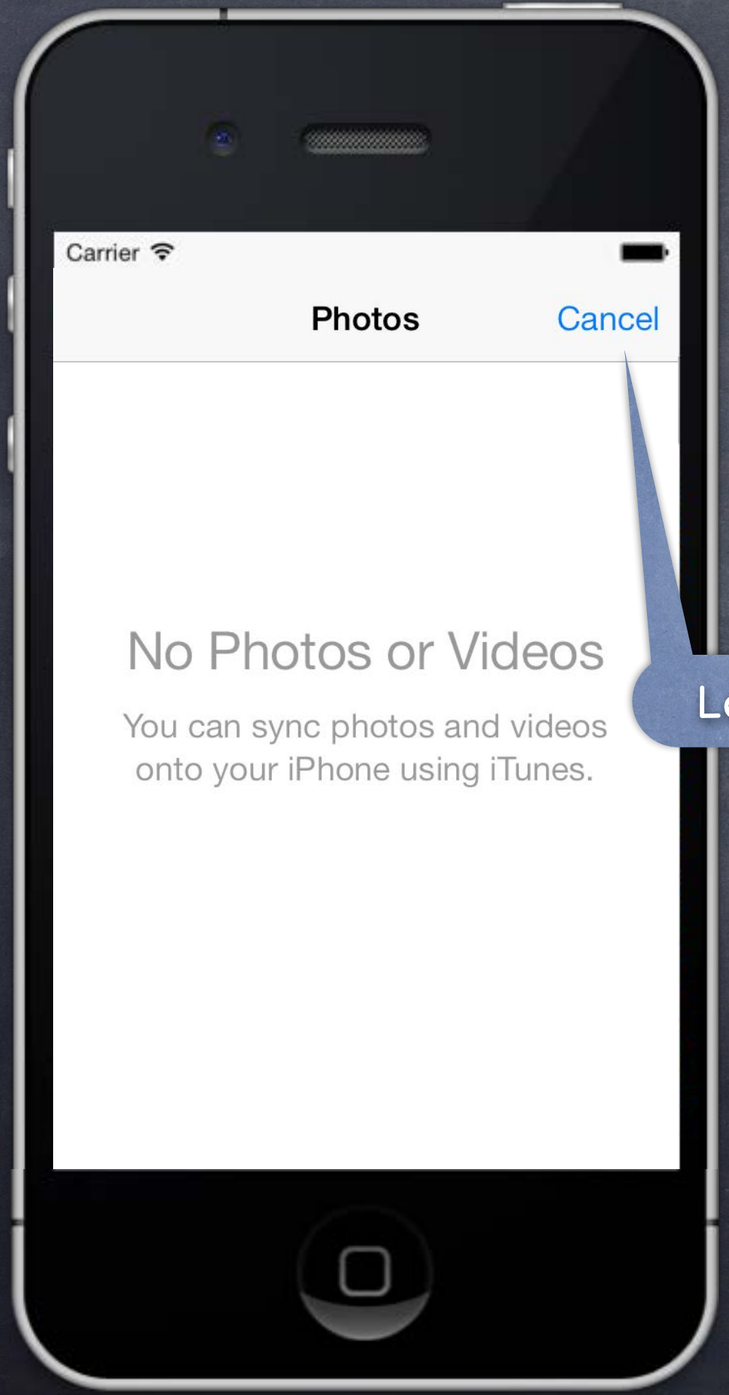


- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

Again, no back button.

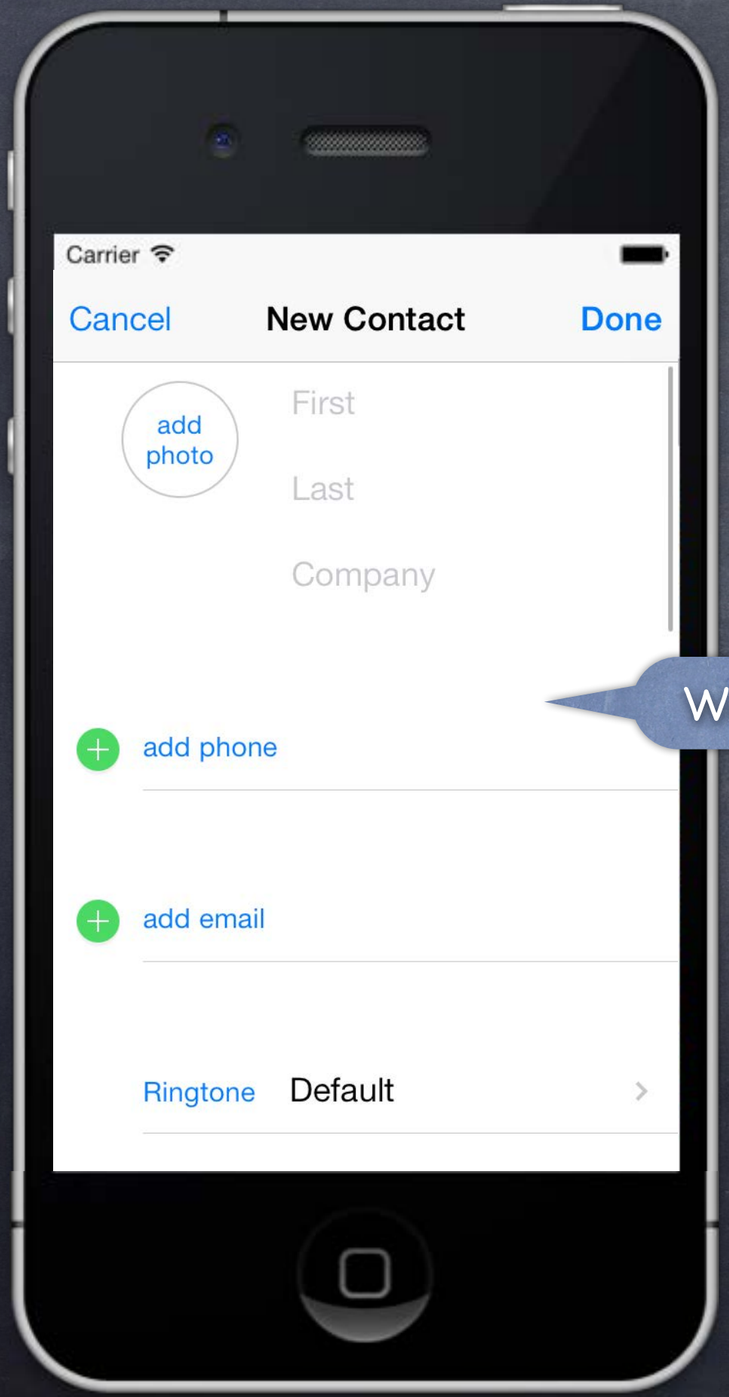
Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Let's Cancel and see what happens.

Modal View Controllers

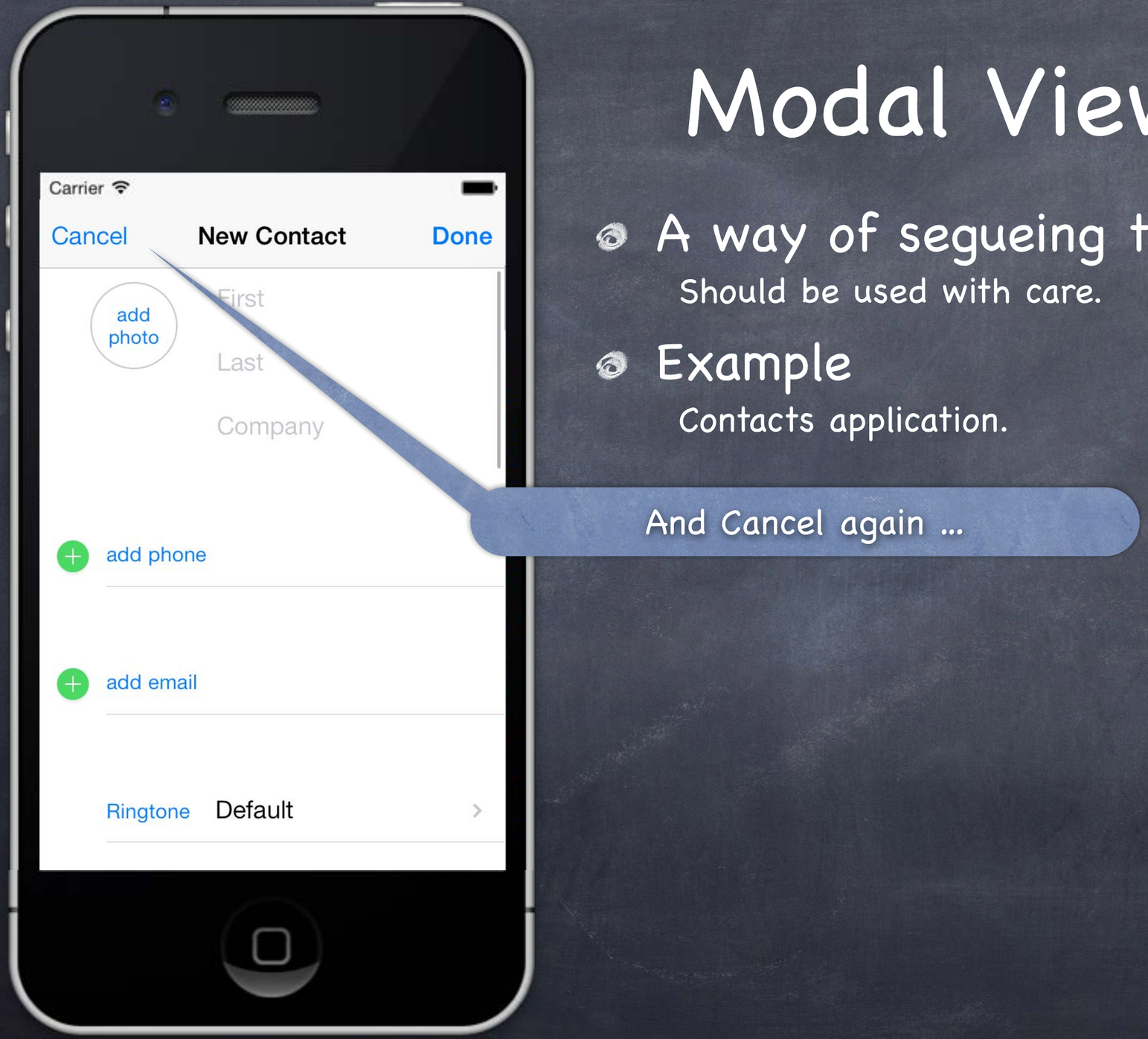


- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

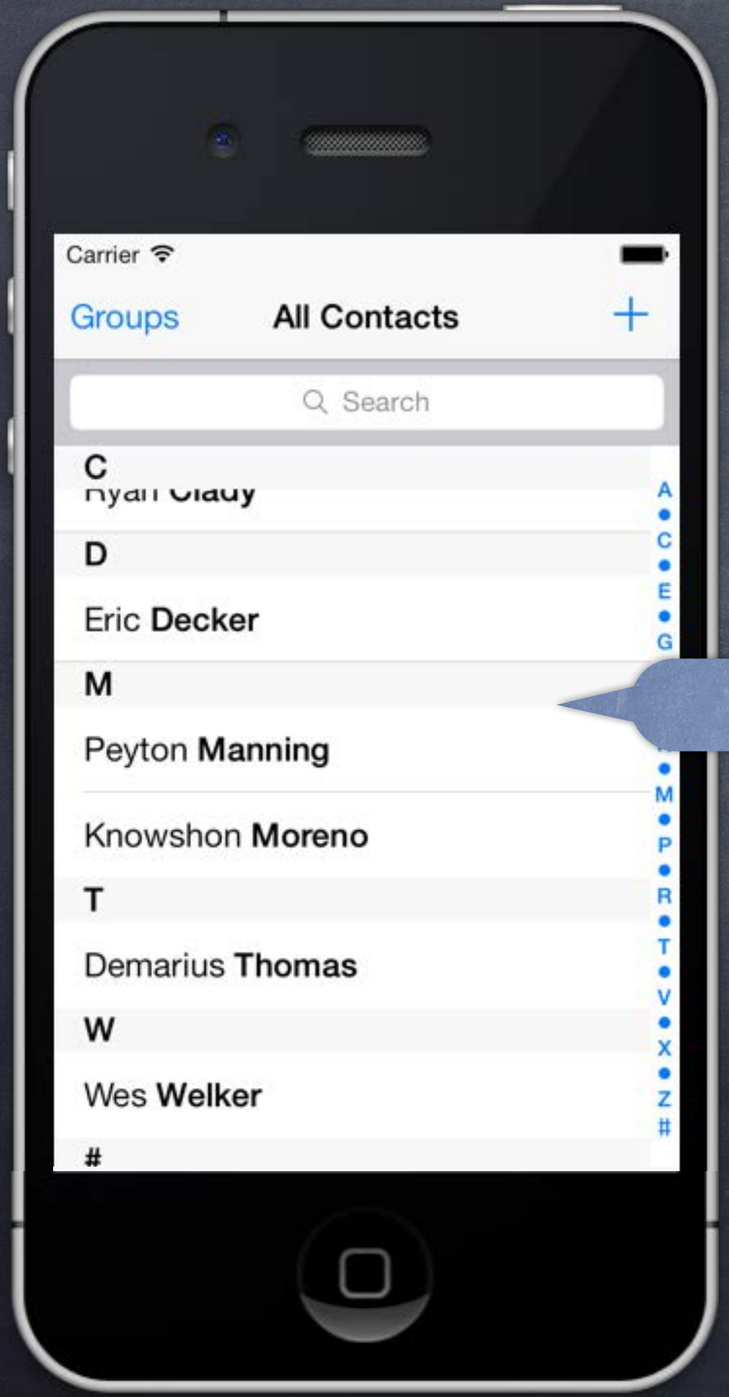
We're back to the last Modal View Controller.

Modal View Controllers

- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.



Modal View Controllers



- A way of segueing that takes over the screen
  - Should be used with care.
- Example
  - Contacts application.

Back to where we started.

Modal View Controllers

- Considerations

  The view controller we segue to using a Modal segue will take over the entire screen.
  This can be rather disconcerting to the user, so use this carefully.

- How do we set a Modal segue up?

  Just ctrl-drag from, for example, a button to another View Controller & pick segue type "Modal".
  Inspect the segue to set the style of presentation (more on this later).

  If you need to present a Modal VC not from a button, use a manual segue (last lecture).
  Or it can be done in code (not via segue) with `presentViewController:animated:completion:` method
  (that's kind of "old style" way to do it, though, pretty rare).

Modal View Controllers

- Preparing for a Modal segue

  You prepare for a Modal segue just like any other segue …

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
    if ([segue.identifier isEqualToString:@“GoToMyModalVC”]) {
        MyModalVC *vc = segue.destinationViewController;
        // set up the vc to run here
    }
}
```

- Hearing back from a Modally segue-to View Controller

  When the Modal View Controller is “done”, how does it communicate results back to presenter?
  You do this by having the segued-to View Controller “segue back” using an “unwind segue.”
  “Unwind segues” are special because they are the only segues that do not instantiate a new VC!
  Instead, they segue to an already existing VC.
  But they are limited to VC’s that “presented” the VC that is segueing back.
  This can be this Modal mechanism, but could also be, e.g., “pushing” in a navigation controller.

Modal View Controllers

- Setting up an Unwind Segue

In the presenting view controller (the one to which you want to "segue back" or "unwind"),
you implement an `IBAction` with any name, but with a `UIStoryBoardSegue` as its argument.

For example …

```
- (IBAction)done:(UIStoryboardSegue *)segue
    MyModalVC *vc = (MyModalVC *)segue.sourceViewController;
    // get results out of vc, which I presented
}
```

Then, ctrl-drag from some UI (button?) in the presented view controller's scene to this icon
in the presented view controller's scene (not in the presenter's scene).

Select the method (e.g. `done:` above) you want to use to unwind.

Now the method above will be called in the presenting view controller when that UI is activated.

When this happens, a modally presented view controller will also automatically dismiss.

The presented view controller will also be sent `prepareForSegue:sender:` before `done:` gets called.
(You can set an unwind segue's `identifier` using the Document Outline.)

Modal View Controllers

- Can you dismiss a view controller from code?

  Yes, but it is generally not the preferred way to do it (unwind instead) …

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

  You do NOT send this to the modal VC! You send it to the view controller that presented it.

- Modal view controllers dismissing themselves

  This is usually frowned upon.

  However, it sometimes happens on cancel (i.e. the user did nothing in the modal view controller).

  But you still do it by sending `dismissModalViewController:` to the presenting view controller:

```
[self.presentingViewController dismissViewControllerAnimated: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;
UIModalTransitionStyleCoverVertical  // slides up and down from bottom of screen
UIModalTransitionStyleFlipHorizontal // flips the current view controller view over to modal
UIModalTransitionStyleCrossDissolve  // old fades out as new fades in
UIModalTransitionStylePartialCurl    // only if presenter is full screen (and no more modal)
```

- What about iPad?

  Sometimes it might not look good for a presented view to take up the entire screen.

```
@property UIModalPresentationStyle modalPresentationStyle;  // in the modal VC
UIModalPresentationFullScreen     // full screen anyway (always on iPhone/iPod Touch)
UIModalPresentationPageSheet      // full screen height, but portrait width even if landscape
UIModalPresentationFormSheet      // centered on the screen (all else dimmed)
UIModalPresentationCurrentContext // parent's context (e.g. in a popover)
```

  Also possible for the presenting VC to control these things (see `definesPresentationContext`).

UITextField

- Like UILabel, but editable
  - Typing things in on an iPhone is secondary UI (keyboard is tiny).
  - More of a mainstream UI element on iPad.
  - Don't be fooled by your UI in the simulator (because you can use physical keyboard!).
  - You can set attributed text, text color, alignment, font, etc., just like a UILabel.
- Keyboard appears when UITextField becomes "first responder"
  - It will do this automatically when the user taps on it.
  - Or you can make it the first responder by sending it the becomeFirstResponder message.
  - To make the keyboard go away, send resignFirstResponder to the UITextField.
- Delegate can get involved with Return key, etc.
  - `- (BOOL)textFieldShouldReturn:(UITextField *)sender;` // sent when Return key is pressed
  - Oftentimes, you will `[sender resignFirstResponder]` in this method.
  - Returns whether to do normal processing when Return key is pressed (e.g. target/action).

UITextField

- Finding out when editing has ended
  - Another delegate method …
  - `- (void)textFieldDidEndEditing:(UITextField *)sender;`
  - Sent when the text field resigns being first responder.
- Finding out when the text changes
  - `UITextFieldTextDidChangeNotification`
  - You can sign up for this `NSNotification` to find out when the user changes the text.
- `UITextField` is a `UIControl`
  - So you can also set up target/action to notify you when things change.
  - Just like with a button, there are different `UIControlEvents` which can kick off an action.
  - Right-click on a `UITextField` in a storyboard to see the options available.

Keyboard

- Controlling the appearance of the keyboard

  Set the properties defined in the `UITextInputTraits` protocol (which `UITextField` implements).

```
@property UITextAutocapitalizationType autocapitalizationType; // words, sentences, etc.
@property UITextAutocorrectionType autocorrectionType;         // UITextAutocorrectionTypeYES/NO
@property UIReturnKeyType returnKeyType;                       // Go, Search, Google, Done, etc.
@property BOOL secureTextEntry;                                // for passwords, for example
@property UIKeyboardType keyboardType;                         // ASCII, URL, PhonePad, etc.
```

- The keyboard comes up over other views

  So you may need to adjust your view positioning (especially to keep the text field itself visible).

  You do this by reacting to the `UIKeyboard{Will,Did}{Show,Hide}Notifications` sent by `UIWindow`.

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                         selector:@selector(theKeyboardAppeared:)
                                             name:UIKeyboardDidShowNotification
                                           object:self.view.window];
```

  The `userInfo` in the `NSNotification` will have details about the appearance.

  `UITableViewController` listens for this and scrolls table automatically if a row has a `UITextField`.

UITextField

- Other UITextField properties

```
@property BOOL clearsOnBeginEditing;
@property BOOL adjustsFontSizeToFitWidth;
@property CGFloat minimumFontSize;  // always set this if you set adjustsFontSizeToFitWidth
@property NSString *placeholder;    // drawn in gray when text field is empty
@property UIImage *background/disabledBackground;
@property NSDictionary *defaultTextAttributes; // applies to entire text
```

- Other UITextField functionality

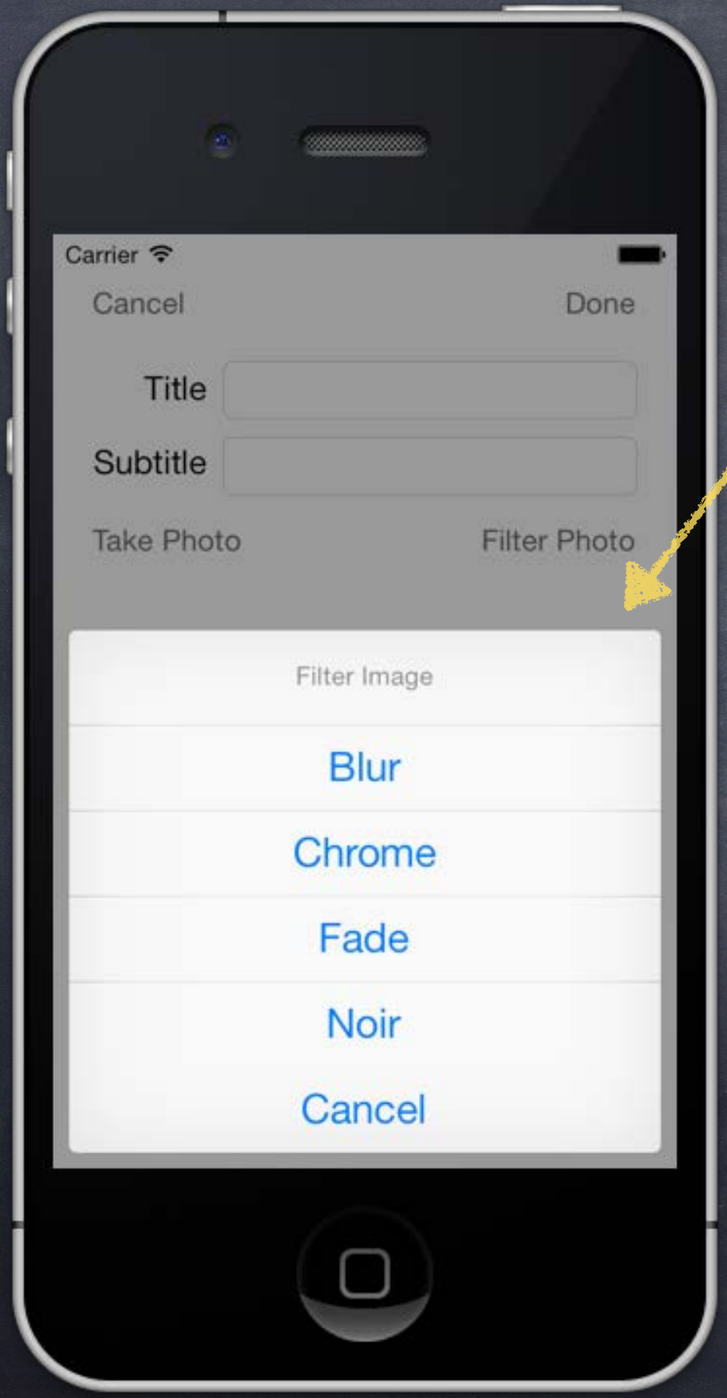
  UITextFields have a "left" and "right" overlays (similar to accessory views in MKAnnotationView).
  You can control in detail the layout of the text field (border, left/right view, clear button).

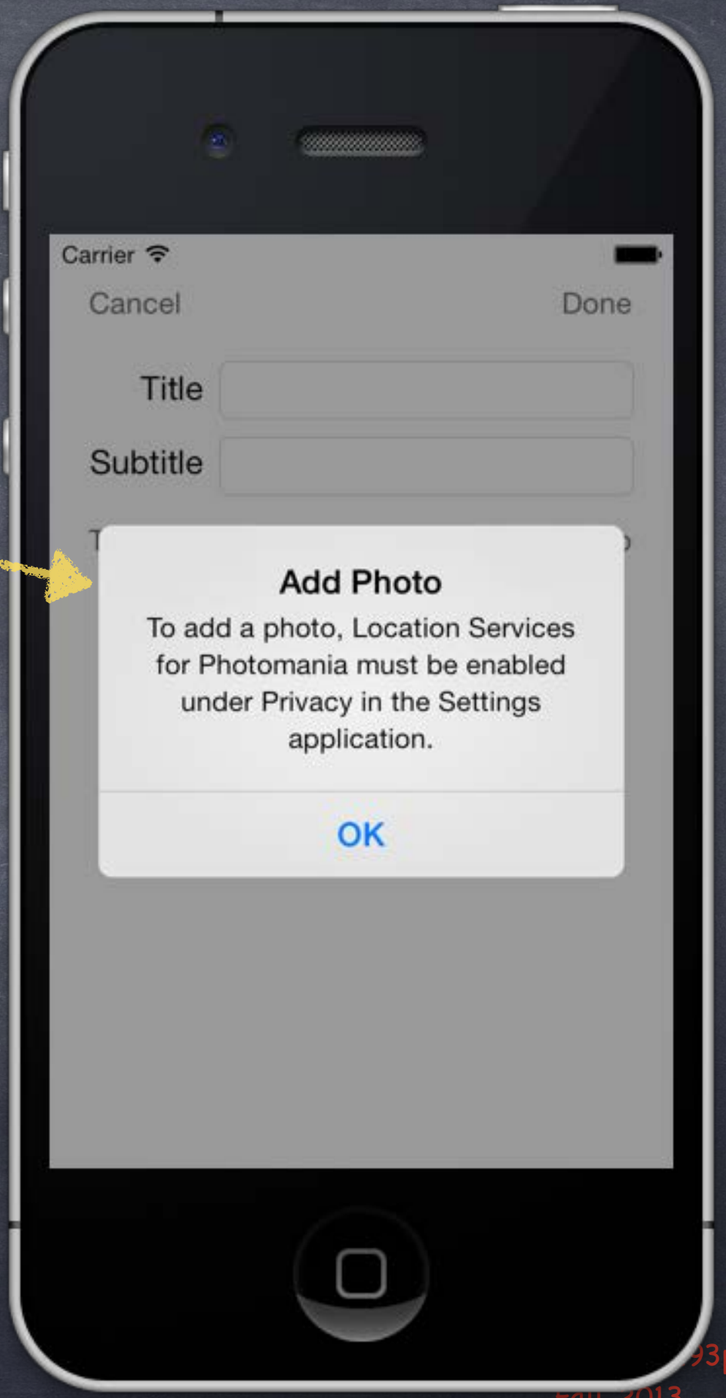
- Other Keyboard functionality

  Keyboards can have accessory views that appear above the keyboard (custom toolbar, etc.).

```
@property (retain) UIView *inputAccessoryView;  // UITextField method
```

Action Sheet & Alert





Alerts and Action Sheets

- Two kinds of "pop up and ask the user something" mechanisms
  - Alerts
  - Action Sheets
- Alerts
  - Pop up in the middle of the screen.
  - Usually ask questions with only two (or one) answers (e.g. OK/Cancel, Yes/No, etc.).
  - Can be disruptive to your user-interface, so use carefully.
  - Often used for "asynchronous" problems ("connection reset" or "network fetch failed").
- Action Sheets
  - Usually slides in from the bottom of the screen on iPhone/iPod Touch, and in a popover on iPad.
  - Can be displayed from a tab bar, toolbar, bar button item or from a rectangular area in a view.
  - Usually asks questions that have more than two answers.
  - Think of action sheets as presenting "branching decisions" to the user (i.e. what next?).

UIActionSheet

- Initializer

```
-(id)initWithTitle:(NSString *)title
          delegate:(id <UIActionSheetDelegate>)delegate
 cancelButtonTitle:(NSString *)cancelButtonTitle
destructiveButtonTitle:(NSString *)destructiveButtonTitle
 otherButtonTitles:(NSString *)otherButtonTitles, ...;
```

- And you can add more buttons programmatically

```
- (void)addButtonWithTitle:(NSString *)buttonTitle;
```

- Displaying the Action Sheet

```
UIActionSheet *actionSheet = [[UIActionSheet alloc] initWithTitle:…];
[actionSheet showInView:(UIView *)];  // centers the view on iPad (don't use this on iPad)
[actionSheet showFromRect:(CGRect) inView:(UIView *) animated:(BOOL)];   // good on iPad
[actionSheet showFromBarButtonItem:(UIBarButtonItem *) animated:(BOOL)]; // good on iPad
```

Universal apps require care here (though some can work on both platforms, e.g., `showFromRect:`).

UIActionSheet

- Finding out what the user has chosen via the delegate

```
- (void)actionSheet:(UIActionSheet *)sender didDismissWithButtonIndex:(NSInteger)index;
```

- Remember from initializer that Cancel/Destructive are special

```
@property NSInteger cancelButtonIndex;      // don't set this if you set it in initializer
@property NSInteger destructiveButtonIndex; // don't set this if you set it in initializer
```

- Other indexes

```
@property (readonly) NSInteger firstOtherButtonIndex;
@property (readonly) NSInteger numberOfButtons;
- (NSString *)buttonTitleAtIndex:(NSInteger)index;
```

The "other button" indexes are in the order you specified them in initializer and/or added them.

- You can programmatically dismiss the action sheet as well

```
- (void)dismissWithClickedButtonIndex:(NSInteger)index animated:(BOOL)animated;
```

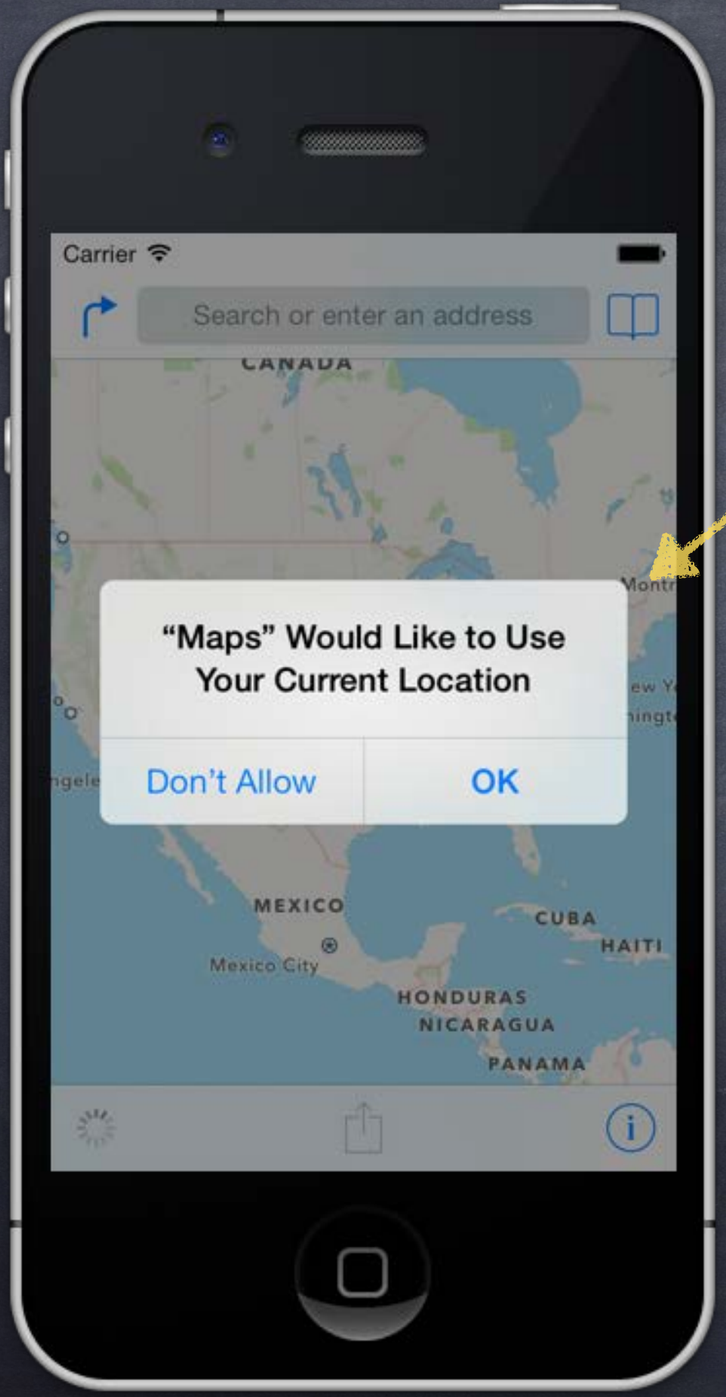
It is generally recommended to call this on `UIApplicationDidEnterBackgroundNotification`.
Remember also that you might be terminated while you are in the background, so be ready.

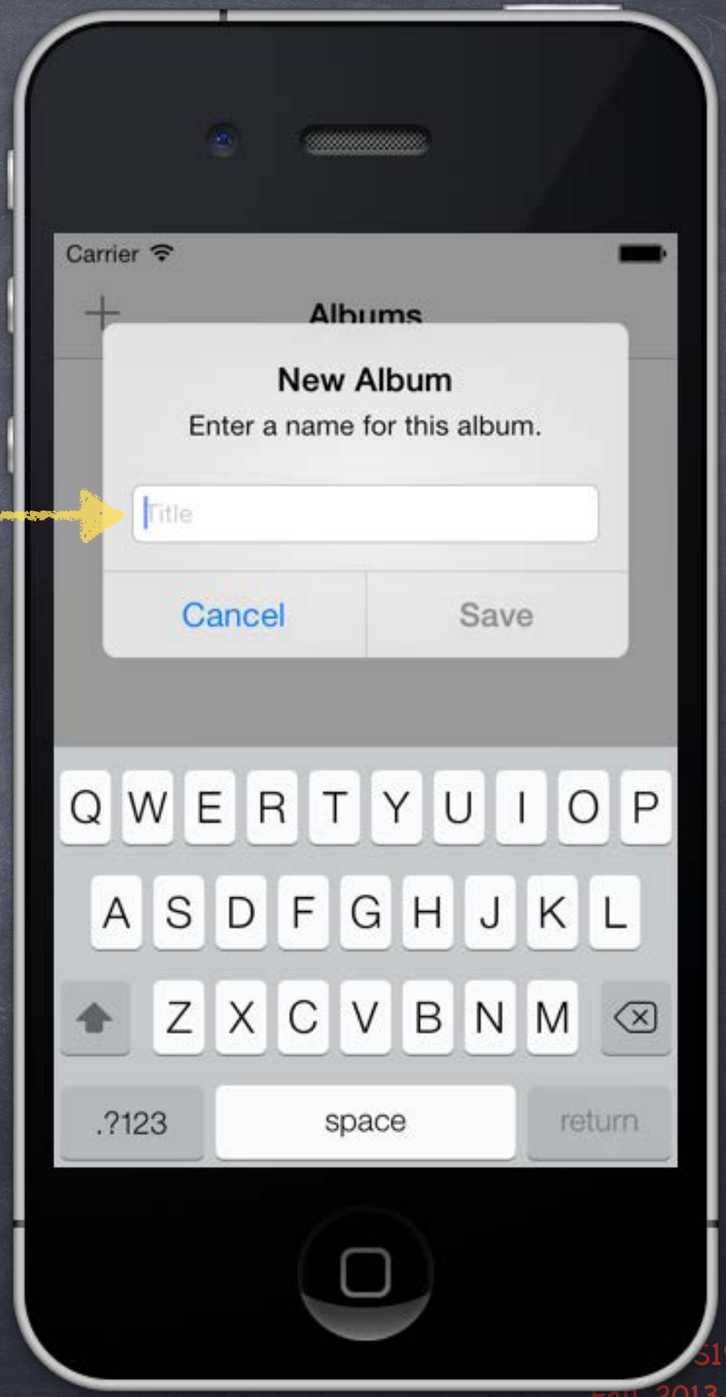
UIActionSheet

- Special popover considerations: no Cancel button
  - An action sheet in a popover (that is not inside a popover) does not show the cancel button.
  - It does not need one because clicking outside the popover dismisses it.
  - It will automatically not show the Cancel button (just don't be surprised that it's not there).
- Special popover considerations: the popover's `passthroughViews`
  - If you `showFromBarButtonItem:animated:`, it adds the toolbar to popover's `passthroughViews`.
  - This is annoying because repeated touches on the bar button item give multiple action sheets!
  - Also, other buttons in your toolbar will work (which might or might not make sense).
  - Unfortunately, you just have to handle this in all of your bar buttons, including the action sheet's.
- Special popover considerations: bar button item handling
  - Have a `weak @property` in your class that points to the `UIActionSheet`.
  - Set it right after you show the action sheet.
  - Check that `@property` at the start of your bar button item's action method.
  - If it is not-`nil` (since it is `weak`, it will only be non-`nil` if it's still on-screen), just dismiss it.
  - If it is `nil`, prepare and show your action sheet.

UIAlertView

Multiple Buttons
&
Embedded Views





UIAlertView

- Very similar to Action Sheet …

```
-(id)initWithTitle:(NSString *)title
           message:(NSString *)message // different from UIActionSheet
          delegate:(id <UIActionSheetDelegate>)delegate
 cancelButtonTitle:(NSString *)cancelButtonTitle
 otherButtonTitles:(NSString *)otherButtonTitles, ...;
```

- And you can add more buttons programmatically

```
- (void)addButtonWithTitle:(NSString *)buttonTitle;
```

- Displaying the Action Sheet

```
UIAlertView *alert = [[UIAlertView alloc] initWithTitle:…];
[alert show];    // different from UIActionSheet, always appears in center of screen
```

- You can even have a UITextField in your Alert

```
alert.alertViewStyle = UIAlertViewStyle{SecureText,PlainText,LoginAndPassword}Input;
```

`[alertView textFieldAtIndex:0]` gives you the `UITextField` (1 is password in `LoginAndPassword`)

Demo

- Photomania Add Photo
  - Let user add a photo to our Photomania database using the camera.
  - We probably won't actually get to the "camera" part today!
  - But we'll set up for that by creating a Modally-segued-to View Controller to do it.
  - Watch for ... Modal Segue, Unwinding Segue, Text Field, Alert

Coming Up

- Wednesday
  - Demo Continued
  - `UIImagePickerController` (Camera)
  - Core Motion
- Friday
  - Sprite Kit
- Next Week
  - Thanksgiving