Stanford CS193p

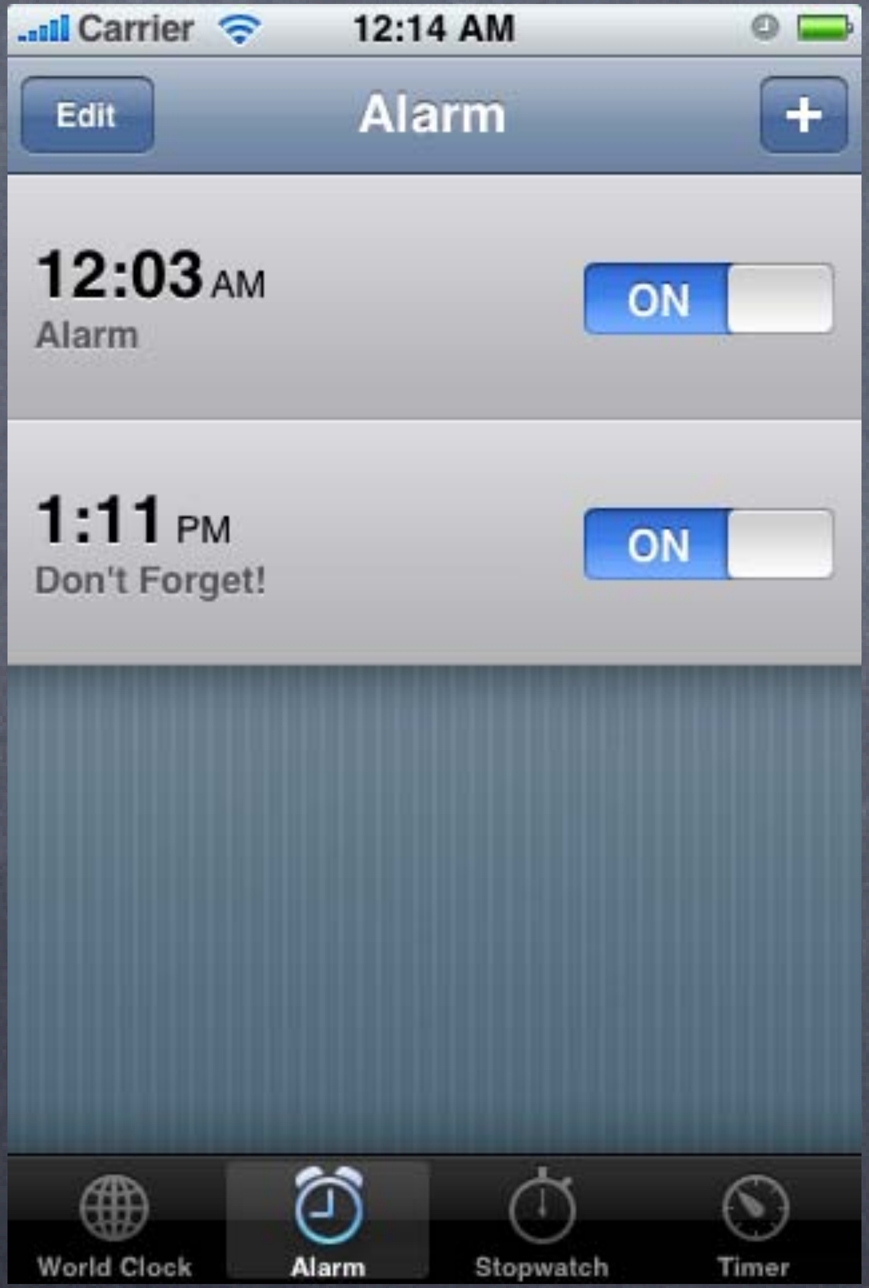
Developing Applications for iOS
Fall 2011

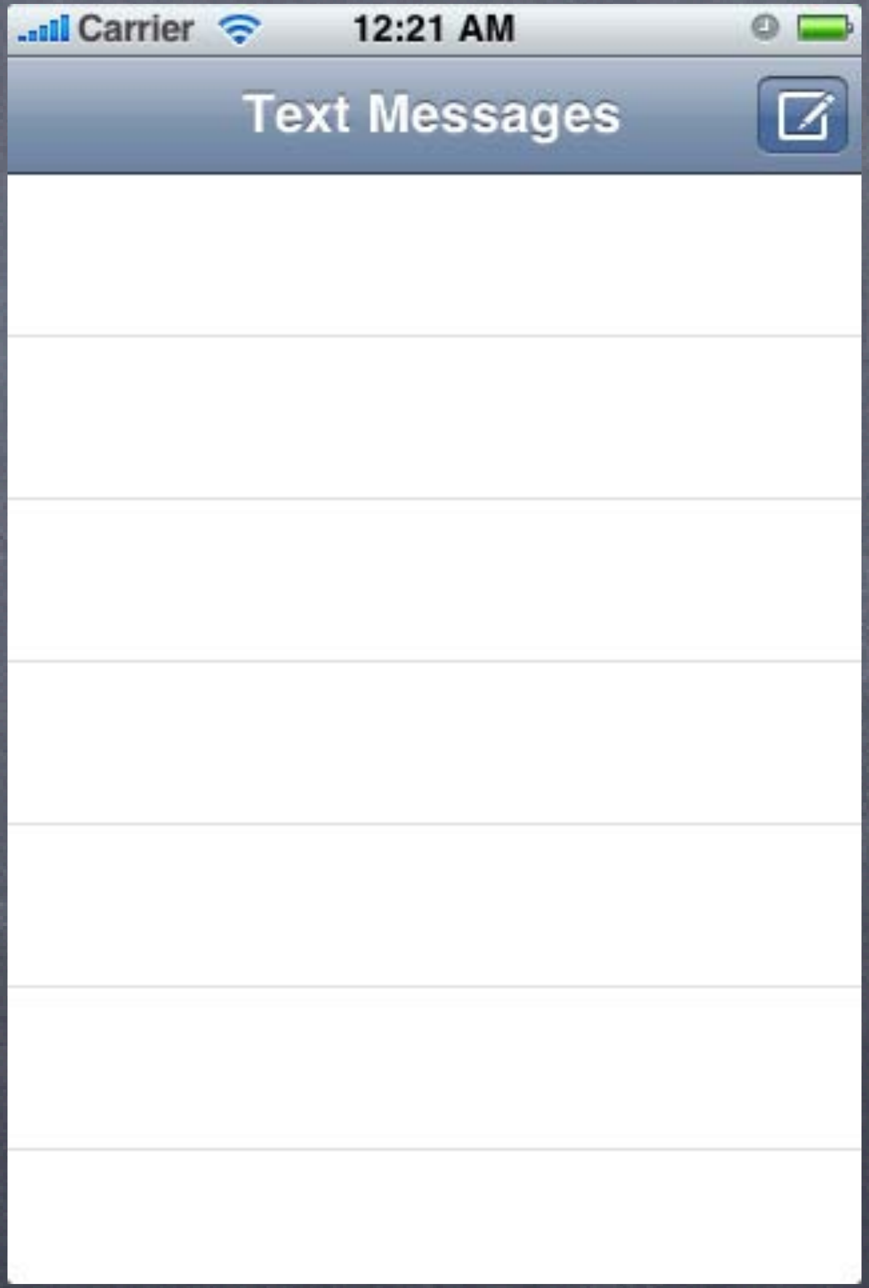
Today

- Modal View Controllers
  View Controllers that "take over the screen."
- UITextField and UITextView
  Inputting text.
- UIView Animation
  Making changes to a UIView and having the result be shown over time.
- NSTimer
  Doing things on a scheduled basis.

Modal View Controllers

- Making a view controller's view appear temporarily
  And blocking all other "navigation" in the application until the user has dealt with this view.





Modal View Controllers

- This is what we mean by segueing to a View Controller "modally"
  "Modally" means "in a mode": nothing else can go on while that other View Controller is up.
  There are different transition styles and presentation styles (more on this in a moment).
  Just ctrl-drag from a button to the modal View Controller. Inspect segue to set style.

- Can be done from code as well (less common)
  Example. Putting up a modal view that asks the user to find an address.

```
- (IBAction)lookupAddress
{
    AddressLookupViewController *alvc =
        [self.storyboard instantiateViewControllerWithIdentifier:@"AddressLookup"];
    [self presentModalViewController:alvc animated:YES completion:^{
        // alvc is now on screen; often we do nothing here
    }];
}
```

This method will fill the entire screen with alvc's view and immediately return after the block.
The user will then not be able to do anything except interact with alvc'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 `alvc` up ...

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

  You do NOT send this to `alvc`! You send it to the view controller that presented `alvc`.

  (i.e. the one that implements the method `lookupAddress` above).

- Modal view controllers dismissing themselves

  This is generally 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] dismissModalViewController:YES];
  ```

- So if dismissing oneself is frowned upon, how does it happen?

  Using delegation.

  Modal view controller reports its results back to its delegate (almost always its presenter).

Modal View Controllers

- Reporting results from a modal view controller

  Example delegation.

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
    if ([segue.identifier isEqualToString:@“Lookup Address”]) {
        AddressLookupViewController *alvc = (AddressLookupViewController *)segue.dest…;
        // other setup here
        alvc.delegate = self;
    }
}
// (One of) AddressLookupViewController’s delegate method(s) implemented in presenter …
- (void)addressLookupViewController:(AddressLookupViewController *)sender
                  didSelectAddress:(Address *)anAddress
{
    // do something with the address the user selected (anAddress)
    [self dismissModalViewControllerAnimated:YES];  // takes sender off screen
}
```

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 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.
- The text is obtained from the UITextField via its delegate

```
- (BOOL)textFieldShouldReturn:(UITextField *)sender;   // sent when return key is pressed
- (void)textFieldDidEndEditing:(UITextField *)sender;
```

  This last one is sent when the text field resigns being first responder.
  This is usually where you extract the text from the field using the text property.

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 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];
```

  Your theKeyboardAppeared: method will get called with an NSNotification as the argument.

  Inside the NSNotification is a userInfo which will have details about the appearance.

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;
```

- 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
```

UITextView

- UITextView is for multi-line, scrolling text
  Editable.
  Can set font and color of (the entire) text, of course.
  But does not support per-character formatting (use UIWebView and HTML for that).
- UITextViewDelegate
  Get notified when editing starts/ends.
  Control editing (prevent changes, etc.).
- UITextView is a UIScrollView
  Has a text-specific scrolling method ...

```
- (void)scrollRangeToVisible:(NSRange)rangeOfCharactersToScrollToVisible;
```

UIView Animation

- Changes to certain UIView properties can be animated over time
  View hierarchy (adding and removing subviews)
  `hidden`
  `frame`
  `transform` (translation, rotation and scale)
  `alpha` (opacity)
- Done with UIView class method and blocks
  The class method takes animation parameters and an animation block as arguments.
  The animation block contains the code that makes the changes to the UIView(s).
  Most also have a "completion block" to be executed when the animation is done.
  The changes inside the block are made <u>immediately</u> (even though they will <u>appear</u> "over time").
- Built on top of underlying Core Animation framework
  We're not going to cover that in this course, but you should know it exists.

UIView Animation

- Animation class method in UIView

```
+ (void)animateWithDuration:(NSTimeInterval)duration
                      delay:(NSTimeInterval)delay
                    options:(UIViewAnimationOptions)options
                 animations:(void (^)(void))animations
                 completion:(void (^)(BOOL finished))completion;
```

- Example

```
[UIView animateWithDuration:3.0
                      delay:0.0
                    options:UIViewAnimationOptionBeginFromCurrentState
                 animations:^{ myView.alpha = 0.0; }
                 completion:^(BOOL fin) { if (fin) [myView removeFromSuperview]; }];
```

This would cause myView to "fade" out over 3 seconds (starting immediately).
Then it would remove myView from the view hierarchy (but only if the fade completed).
If, within the 3 seconds, someone animated the alpha to non-zero, the removal would not happen.

UIView Animation

- Another example

```
if (myView.alpha == 1.0) {
    [UIView animateWithDuration:3.0
                          delay:2.0
                        options:UIViewAnimationOptionBeginFromCurrentState
                     animations:^{ myView.alpha = 0.0; }
                     completion:nil];
    NSLog(@“alpha is %f.”, myView.alpha);
}
```

This would also cause `myView` to “fade” out over 3 seconds (starting in 2 seconds in this case).
The `NSLog()` would happen immediately (i.e. not after 3 or 5 seconds) and would print “alpha is 0.”
In other words, the animation block’s changes are executed immediately, but the animation itself
(i.e. the visual appearance of the change to `alpha`) starts in 2 seconds and takes 3 seconds.

UIView Animation

- UIViewAnimationOptions

BeginFromCurrentState	// interrupt other, in-progress animations of these properties
AllowUserInteraction	// allow gestures to get processed while animation is in progress
LayoutSubviews	// animate the relayout of subviews along with a parent's animation
Repeat	// repeat indefinitely
Autoreverse	// play animation forwards, then backwards
OverrideInheritedDuration	// if not set, use duration of any in-progress animation
OverrideInheritedCurve	// if not set, use curve (e.g. ease-in/out) of in-progress animation
AllowAnimatedContent	// if not set, just interpolate between current and end state image
CurveEaseInEaseOut	// slower at the beginning, normal throughout, then slow at end
CurveEaseIn	// slower at the beginning, but then constant through the rest
CurveLinear	// same speed throughout
TransitionFlipFromLeft/Right	// only for hiding/removing views from the view hierarchy
TransitionCurlUp/Down	// only for hiding/removing views from the view hierarchy

UIView Animation

- Animating changes to the view hierarchy is slightly different

```
+ (void)transitionFromView:(UIView *)fromView
                    toView:(UIView *)toView
                  duration:(NSTimeInterval)duration
                   options:(UIViewAnimationOptions)options
                completion:(void (^)(BOOL finished))completion;
```

Include UIViewAnimationOptionShowHideTransitionViews if you want hidden property to be set.
Otherwise it will actually remove fromView from the view hierarchy and add toView.

Or you can do the removing/adding/hiding yourself in a block with ...

```
+ (void)transitionWithView:(UIView *)view
                  duration:(NSTimeInterval)duration
                   options:(UIViewAnimationOptions)options
                animations:(void (^)(void))animations
                completion:(void (^)(BOOL finished))completion;
```

NSTimer

- Scheduled invocation of a method in the main queue

```
NSTimer *timer = [NSTimer scheduledTimerWithTimeInterval:(NSTimeInterval)seconds
                                                  target:self
                                                selector:@selector(doSomething:)
                                                userInfo:(id)anyObject
                                                 repeats:(BOOL)yesOrNo];
```

- Not "real time" since it can run only each time around run loop

  Setting the time interval too short will essentially block the main thread.
  Taking too long each time you're called could also essentially block the main thread.
  Do any time consuming stuff in a thread and just use the timer to update state quickly.

- Check documentation for more

  For example, you can `invalidate` a repeating timer when you want it to stop.
  Or you can create a timer that will fire at a specific time (NSDate) in the future.

Demo

- Kitchen Sink
  Modal presentation of a view controller
  Getting text input via a UITextField
  UIView animation
  NSTimer
- Watch for ...
  How modal view controller communicates its results back to view controller via delegation.
  How the modal view controller gets dismissed.
  How we use the UITextField delegate to get the text out of it.
  How animation happens visually over time, but the value in the UIView gets set immediately.
  How we turn timers on and off (usually) in viewWillAppear:/Disappear:

Coming Up

- Thursday
  More random topics that might help you with your Final Project
- Friday Section
  Ge Wang, Smule