iOS Mobile Development

Today

- UITextView

  Scrollable, editable/selectable view of a mutable attributed string.

- View Controller Lifecycle

  Finding out what's happening as a VC is created, hooked up to the View, appears/disappears, etc.

- NSNotification

  The "radio station" communication mechanism of MVC. Today we'll just talk about "tuning in."

- Demo

  UITextView
  NSAttributedString, NSMutableAttributedString,
  NSTextStorage UIFont, UIColor
  NSNotification
  View Controller Lifecycle

UITextView

- UITextView

  Like UILabel, but multi-line, selectable/editable, scrollable,
  etc.

- Set its text and attributes via its NSMutableAttributedString

  Obtain the NSMutableAttributedString representing the text in the UITextView using ...

  ```
  @property (nonatomic, readonly) NSTextStorage *textStorage;
  ```

  NSTextStorage is a subclass of NSMutableAttributedString.
  You can simply modify it and the UITextView will automatically update. New in iOS 7.

- Setting the font

  Fonts can, of course, vary from character to character in UITextView.
  But there is also a property that applies a font to the entire
  UITextView ...

  ```
  @property (nonatomic, strong) UIFont *font;
  ```

  Changing the font of every character will not reset other attributes (e.g. color, underline, etc.). However, you will lose your symbolic traits!

UITextView

- Advanced layout in `UITextView` with TextKit

  This property defines "where text can be" in the `UITextView` ...

  ```
  @property (readonly) NSTextContainer *textContainer;
  ```

  This object will read characters from `textStorage` and lays down glyphs into `textContainer` ...

  ```
  @property (readonly) NSLayoutManager *layoutManager;
  ```

  These objects are quite powerful.

  For example, `textContainer` can have "exclusion zones" specified in it (flowing around images). Check them out if you're interested in typography.

Demo

- Attributor

  UITextView
  NSAttributedString, NSMutableAttributedString,
  NSTextStorage UIFont, UIColor

View Controller Lifecycle

- View Controllers have a "Lifecycle"

  A sequence of messages is sent to them as they progress through it.

- Why does this matter?

  You very commonly override these methods to do certain work.

- The start of the lifecycle ...

  Creation.
  MVCs are most often instantiated out of a storyboard (as you've seen).
  There are ways to do it in code (rare) as well which we may cover later in the quarter.

- What then?

  Outlet setting.
  Appearing and disappearing.
  Geometry changes.
  Low-memory situations.
  At each stage, iOS invokes method(s) on the Controller ...

View Controller Lifecycle

- After instantiation and outlet-setting, `viewDidLoad` is called

This is an exceptionally good place to put a lot of setup code.

```
- (void)viewDidLoad
{
[super viewDidLoad];              // always let super have a chance in lifecycle methods
    // do some setup of my MVC
}
```

But be careful because the geometry of your view (its `bounds`) is not set yet!
At this point, you can't be sure you're on an iPhone 5-sized screen or an iPad or ???. So do not initialize things that are geometry-dependent here.

View Controller Lifecycle

- Just before the view appears on screen, you get notified

(argument is just whether you are appearing instantly or over time via animation)

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

Your view will only get "loaded" once, but it might appear and disappear a lot.
So don't put something in this method that really wants to be in `viewDidLoad`. Otherwise, you might be doing something over and over unnecessarily.

Do something here if things you display are changing while your MVC is off-screen.

Later we will use this to optimize performance by waiting until this method (as opposed to `viewDidLoad`) to kick off an expensive operation (probably in another thread).

View's geometry is set here, but there are other (better?) places to react to geometry.

View Controller Lifecycle

- And you get notified when you will disappear off screen too

This is where you put "remember what's going on" and cleanup code.

```
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];  // call super in all the viewWill/Did... methods
    // let's be nice to the user and remember the scroll position they were at ...
[self rememberScrollPosition];  // we'll have to implement this, of course
    // do some other clean up now that we've been removed from the screen
[self saveDataToPermanentStore]; // maybe do in did instead?
    // but be careful not to do anything time-consuming here, or app will be sluggish
    // maybe even kick off a thread to do what needs doing here (again, we'll cover threads later)
}
```

- There are "did" versions of both of the appearance methods

```
- (void)viewDidAppear:(BOOL)animated;
- (void)viewDidDisappear:(BOOL)animated;
```

View Controller Lifecycle

Geometry changed?

Most of the time this will be automatically handled with Autolayout (next week).

`- (void)view{Will,Did}LayoutSubviews;`

Called any time a view's `frame` changed and its `subviews` were thus re-layed out.
For example, autorotation.
You can reset the `frames` of your `subviews` here or set other geometry-affecting properties. Between "`will`" and "`did`", autolayout will happen (we'll talk about that next week).

View Controller Lifecycle

Autorotation

When the device is rotated, the top level view controller will have its bounds reoriented iff ...

The view controller returns `YES` from `shouldAutorotate`.

The view controller returns the new orientation in `supportedInterfaceOrientations`.

The application allows rotation to that orientation (defined in `Info.plist` file).

Generally it is a good idea to try to support rotation in MVCs. We'll talk about that next week.

Specific notification that rotation will/did happen

```
- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)anOrientation
                                duration:(NSTimeInterval)seconds;

- (void)willAnimateRotationToInterfaceOrientation:(UIInterfaceOrientation)orient
                                duration:(NSTimeInterval)seconds;

- (void)didRotateFromInterfaceOrientation:(UIInterfaceOrientation)anOrientation;
```

This property will have the current orientation when each of the above is called ...

```
@property UIInterfaceOrientation interfaceOrientation;
```

It's pretty rare to implement these. Autolayout and `viewWillLayoutSubviews` usually suffice.

View Controller Lifecycle

- In low-memory situations, `didReceiveMemoryWarning` gets called ...

  This rarely happens, but well-designed code with big-ticket memory uses might anticipate it.
  Examples: images and sounds.
  Anything "big" that can be recreated should probably be released (i.e. set `strong` pointer to `nil`).

View Controller Lifecycle

- awakeFromNib

This method is sent to all objects that come out of a storyboard (including your Controller). Happens before outlets are set! (i.e. before the MVC is "loaded")
Put code somewhere else if at all possible (e.g. `viewDidLoad` or `viewWillAppear:`).

Anything that would go in your Controller's `init` method would have to go in `awakeFromNib` too (because `init` methods are not called on objects that come out of a storyboard).

```
- (void)setup { };  // do something which can't wait until viewDidLoad
- (void)awakeFromNib { [self setup]; }
// UIViewController's designated initializer is initWithNibName:bundle: (ugh!)
- (instancetype)initWithNibName:(NSString *)name bundle:(NSBundle
*)bundle {
    self = [super initWithNibName:name
                           bundle:bundle];     [self setup];
return self;
}
```

It is unlikely you will ever need to use `awakeFromNib` in this course.

View Controller Lifecycle

- Summary

Instantiated (from storyboard - many ways for this to happen which we'll cover later) `awakeFromNib`

outlets get set

`viewDidLoad`

(when geometry is determined)

`viewWillLayoutSubviews:` and `viewDidLayoutSubviews:`

(next group can happen repeatedly as your MVC appears and disappears from the screen ...)

`viewWillAppear:` and `viewDidAppear:`

(whenever geometry changes again while visible, e.g. device rotation)

`viewWillLayoutSubviews:` and `viewDidLayoutSubviews:`

(if it is autorotation, then you also get will/didRotateTo/From messages--rare to use these)

`viewWillDisappear:` and `viewDidDisappear:`

(possibly if memory gets low ...)

`didReceiveMemoryWarning`

(there is no "unload" anymore, so that's all there is)

Demo

- Attributor

  View Controller Lifecycle

  How (and when in the VCL) would we "outline" the text on the Outline button?

NSNotification

- Notifications

  The "radio station" from the MVC slides.

- NSNotificationCenter

  Get the default "notification center" via `[NSNotificationCenter defaultCenter]`

  Then send it the following message if you want to "listen to a radio station":

```
- (void)addObserver:(id)observer     // you (the object to get notified)
          selector:(SEL)methodToInvokeIfSomethingHappens
              name:(NSString *)name // name of station (a constant somewhere)
            object:(id)sender;      // whose changes you're interested in (nil is anyone's)
```

- You will then be notified when there are broadcasts

```
- (void)methodToInvokeIfSomethingHappens:(NSNotification *)notification
{
    notification.name     // the name passed above
    notification.object   // the object sending you the notification
    notification.userInfo // notification-specific information about what happened
}
```

NSNotification

- Be sure to "tune out" when done listening

```
[center removeObserver:self];
```

or

```
[center removeObserver:self name:UIContentSizeCategoryDidChangeNotification object:nil];
```

Failure to remove yourself can sometimes result in crashers.

This is because the `NSNotificationCenter` keeps an "unsafe retained" pointer to you.

A good place to remove yourself is when your MVC's View goes off screen. We'll see how to find out about that later in this lecture.

Or you can remove yourself in a method called `dealloc` (called when you leave the heap).

```
- (void)dealloc
{
    // be careful in this method! can't access properties! you are almost gone from heap!
[[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

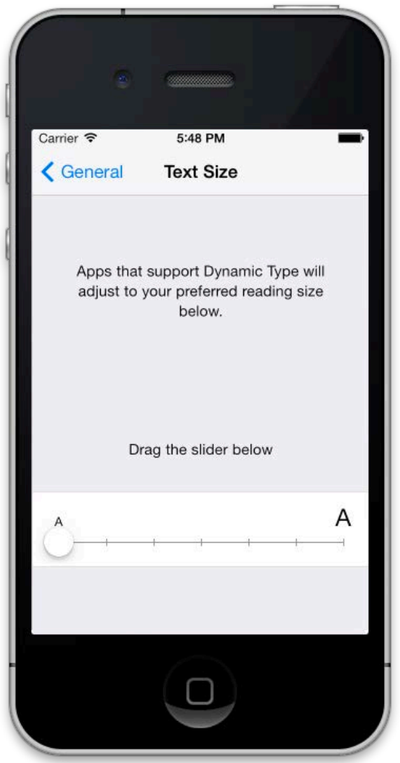
NSNotification

- Example

Watching for changes in the size of preferred fonts (user can change this in Settings) ...

```
NSNotificationCenter *center = [NSNotificationCenter
defaultCenter]; [center addObserver:self
           selector:@selector(preferredFontsSizeChanged:)
               name:UIContentSizeCategoryDidChangeNotification
             object:nil]; // this station's broadcasts aren't object-specific

- (void)preferredFontsSizeChanged:(NSNotification *)notification
{
    // re-set the fonts of objects using preferred fonts
}
```



Demo

- Attributor

  NSNotification

  More View Controller Lifecycle

Next Time

Multiple MVCs in one application

`UITabBarController`

`UINavigationController`