Stanford CS193p

Developing Applications for iOS
Spring 2016

Today

- Demo
  - Cassini Continued
  - Code-Driven Segue
  - Zooming
  - Reusing the Detail of a Split View
  - Managing what appears when Split View first shows up
- Multithreading
  - Keeping the UI responsive
  - Multithreaded Cassini Demo
- Text Field
  - Like `UILabel`, but editable text

Demo

- Cassini Continued
  - Multiple MVC to view some NASA images
  - Segue from code rather than directly in storyboard
  - Zooming in scroll view
  - Reusing the Detail in a Split View (rather than segueing to a replacement)
  - Controlling the Split View's presentation

Multithreading

- Queues
  - Multithreading is mostly about "queues" in iOS.
  - Functions (usually closures) are lined up in a queue.
  - Then those functions are pulled off the queue and executed on an associated thread.
  - Queues can be "serial" (one at a time) or "concurrent" (multiple things going at once).
- Main Queue
  - There is a very special serial queue called the "main queue."
  - All UI activity MUST occur on this queue and this queue only.
  - And, conversely, non-UI activity that is at all time consuming must NOT occur on that queue.
  - We do this because we want our UI to be highly responsive!
  - And also because we want things that happen in the UI to happen predictably (serially).
  - Functions are pulled off and worked on in the main queue only when it is "quiet".
- Other Queues
  - iOS has some other queues we can use for non-UI stuff (more on that in a moment).

Multithreading

- Executing a function on another queue

```
let queue: dispatch_queue_t = <get the queue you want, more on this in a moment>
dispatch_async(queue) { /* do what you want to do here */ }
```

- The main queue (a serial queue)

```
let mainQ: dispatch_queue_t = dispatch_get_main_queue()
```

All UI stuff must be done on this queue!

And all time-consuming (or, worse, potentially blocking) stuff must be done off this queue!

Common code to write …

```
dispatch_async(not the main queue) {
    // do a non-UI that might block or otherwise takes a while
    dispatch_async(dispatch_get_main_queue()) {
        // call UI functions with the results of the above
    }
}
```

So how do we get a "not the main queue" queue? …

Multithreading

- Other (concurrent) queues (i.e. not the main queue)
  - Most non-main-queue work will happen on a concurrent queue with a certain quality of service

```
QOS_CLASS_USER_INTERACTIVE   // quick and high priority
QOS_CLASS_USER_INITIATED     // high priority, might take time
QOS_CLASS_UTILITY            // long running
QOS_CLASS_BACKGROUND         // user not concerned with this (prefetching, etc.)
let queue = dispatch_get_global_queue(<one of the above>, 0) // 0 is a "reserved for future"
```

  - You will probably use these queues to do any work that you don't want to block the main queue
- You can create your own serial queue if you need serialization

```
let serialQ = dispatch_queue_create("name", DISPATCH_QUEUE_SERIAL)
```

  - Maybe you are downloading a bunch of things things from a certain website
    - but you don't want to deluge that website, so you queue the requests up serially
  - Or maybe the things you are doing depend on each other in a serial fashion

Multithreading

- We are only seeing the tip of the iceberg
  - There is a lot more to GCD (Grand Central Dispatch)
  - You can do locking, protect critical sections, readers and writers, synchronous dispatch, etc.
  - Check out the documentation if you are interested
- There is also an object-oriented API to all of this
  - `NSOperationQueue` and `NSOperation`
  - Surprisingly, we use the non-object-oriented API a lot of the time
  - This is because the "nesting" of dispatching reads very well in the code
  - But the object-oriented API is also quite useful (especially for more complicated multithreading)

Multithreading

- Multithreaded iOS API
  - Quite a few places in iOS will do what they do off the main queue
  - They might even afford you the opportunity to do something off the main queue
  - You may pass in a function (a closure, usually) that sometimes executes off the main thread
  - Don't forget that if you want to do UI stuff there, you must dispatch back to the main queue!

Multithreading

- Example of a multithreaded iOS API

  This API lets you fetch something from an http URL to a local file.

  Obviously it can't do that on the main thread!

```
let session = NSURLSession(configuration: NSURLSessionConfiguration.defaultSessionConfiguration())
if let url = NSURL(string: "http://url") {
    let request = NSURLRequest(URL: url)
    let task = session.downloadTaskWithRequest(request) { (localURL, response, error) in
        /* I want to do UI things here with the result of the download, can I? */
    }
    task.resume()
}
```

  The answer to the above comment is "no".

  That's because the block will be run off the main queue.

  How do we deal with this?

  One way is to use a variant of this API that lets you specify the queue to run on.

  Another way is ...

Multithreading

- How to do UI stuff safely

  You can simply dispatch back to the main queue …

```
let session = NSURLSession(configuration: NSURLSessionConfiguration.defaultSessionConfiguration())
if let url = NSURL(string: “http://url”) {
    let request = NSURLRequest(URL: url)
    let task = session.downloadTaskWithRequest(request) { (localURL, response, error) in
        dispatch_async(dispatch_get_main_queue()) {
            /* I want to do something in the UI here, can I? */
        }
    }
    task.resume()
}
```

  Yes! Because the UI code you want to do has been dispatched back to the main queue.
  But understand that that code might run MINUTES after the request is fired off.
  The user might have long ago given up on whatever was being fetched.

Demo

- Multithreaded Cassini

  Let's get that URL network fetch off the main queue!

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.
  - `func textFieldShouldReturn(sender: UITextField) -> Bool` // when "Return" 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 …
  - `func textFieldDidEndEditing(sender: UITextField)`
  - Sent when the text field resigns being first responder.
- 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).

```
var UITextAutocapitalizationType autocapitalizationType  // words, sentences, etc.
var UITextAutocorrectionType autocorrectionType   // yes or no
var UIReturnKeyType returnKeyType                 // Go, Search, Google, Done, etc.
var BOOL secureTextEntry                          // for passwords, for example
var UIKeyboardType keyboardType                   // ASCII, URL, PhonePad, etc.
```

Keyboard

- 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`.
  - We have not talked about `NSNotifications` yet, but it's pretty simple.
  - You register a method to get called when a named "event" occurs like this …

```
NSNotificationCenter.defaultCenter().addObserver(self,
                                        selector: "theKeyboardAppeared:",
                                            name: UIKeyboardDidShowNotification
                                          object: view.window)
```

  - The event here is `UIKeyboardDidShowNotification`.
  - The `object` is the one who is causing the even to happen (our MVC's view's window).
  - `func theKeyboardAppeared(notification: NSNotification)` will get called when it happens.
  - The `notification.userInfo` will have details about the appearance.
  - `UITableViewController` listens for this & scrolls table automatically if a row has a `UITextField`.

UITextField

- Other UITextField properties

```
var clearsOnBeginEditing: Bool
var adjustsFontSizeToFitWidth: Bool
var minimumFontSize: CGFloat      // always set this if you set adjustsFontSizeToFitWidth
var placeholder: String           // drawn in gray when text field is empty
var background/disabledBackground: UIImage
var defaultTextAttributes: Dictionary // applies to entire text
```

- Other UITextField functionality

  UITextFields have a "left" and "right" overlays.

  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.).

```
var inputAccessoryView: UIView  // UITextField method
```