Announcements

- Lab 3 is due tonight by 11:59 PM
- Lab 4 is due on July 17th

Lab 4 Demo
Today’s Topics

• Web Services
• WKWebView
• Threading
• In class demo 2 (Multiple Tab, JSON App)

Recap

• Property lists
  – Quick & easy, but limited

• Archived objects
  – More flexible, but require writing a lot of code

• SQLite and Core Data
  – Elegant solution for many types of problems

• XML and JSON
  – Low-overhead options for talking to “the cloud”
Firebase Demo

Web Content in iOS
Displaying Web Content

- **Web content can be displayed with WKWebView**
  - Introduced in iOS 8, part of WKWebKit Framework
    - Replaces UIWebView
- **Content can be**
  - local HTML string
  - local raw data + MIME type
  - remote URL
- **Leverages WebKit**
  - full WK functionality not currently exposed
  - simple API for loading & navigating
  - delegate for some control
  - Same JavaScript engine that powers Safari

WKWebView

- **WKWebView subclass, configure in Storyboard or in code**
  - Feed it data to display

  ```swift
  func loadHTMLString(_ string: String, baseURL:URL?) -> WKNavigation?
  
  func load(_ data: Data, mimeType: MIMETYPE: String, characterEncodingName: String, baseURL: URL ) -> WKNavigation?
  ```

- Or give it a URL request
  ```swift
  func load(_ request: URLRequest) -> WKNavigation
  ```

- **WKNavigation**
  - Object that contains information for tracking the loading progress of a webpage

- **What’s this URLRequest?**
  - Encapsulates a URL to load and caching policy for fetched data
  - Older versions of iOS used an NSURL and NSURLConnection
WKWebView

- Properties and actions you’d expect from a web view
  
  - isLoading: Bool
  - canGoBack: Bool
  - canGoForward: Bool
  - reload()
  - stopLoading()
  - goBack()
  - goForward()

- A couple others that are handy
  
  - estimatedProgress: Double
  - evaluateJavaScript(_:completionHandler:)

WKNavigationDelegate

- Callbacks for load progress
  
  - webView(_: didCommit: ) //called when content starts arriving
  - webView(_: didFinish: ) //called when navigation is complete

- Error handling
  
  - webView(_: didFail: withError: )

- Navigation Loading Policy
  
  - Decide whether to allow or cancel a navigation
  - webView(_: decidePolicyFor: decisionHandler: )
Demo
WKWebView

Multithreading in iOS

- Work done with Queues
- Functions (closures) are assigned as units of work to the queues
- Queues execute on a CPU thread
- Queues are either serial or concurrent
- Queues are synchronous or asynchronous
Types of Queues

- **Main Queue**
  - Special serial queue where all UI-Activity happens
  - Non-UI actions should take place on background queue
    - Important to do this to free up main queue

- **Global Queues**
  - Four queues shared by the system with different priority levels

- **Custom Queues**
  - User generated queues with custom attributes (name, priority level, etc)

Multithreading (from CS193P)

- **Executing a function on another queue**
  ```swift
  let someQueue = DispatchQueue(label: "name")
  someQueue.async { /* do work here */ }
  ```

- **The main queue (serial queue)**
  - DispatchQueue.main

- All UI work done on main queue
- All time intensive code or synchronous (blocking) done on another queue
- Swift 3 provides global queues with different priorities

```swift
DispatchQueue.global(qos: .userInitiated).async {
    // do non-UI stuff that may take time
    DispatchQueue.main.async {
        // Call UI functions with with results from other queue
    }
}
Multithreading (from CS193P)

- **Specifying QOS for queues**
  - userInteractive //quick and high priority
  - userInitiated //high priority, may take some time
  - utility //long running
  - background //user not concerned (prefetching)

```swift
let queue1 = DispatchQueue(label: "low priority", qos: DispatchQueue.background)
let queue2 = DispatchQueue(label: "high priority", qos: DispatchQueue.userInteractive)
```

QoS Demo
Multithreading (CS 193P)

- Multithreaded iOS API
  - Many iOS APIs execute on a queue other than the main queue
  - These APIs typically provide a closure as an argument, which is called upon completion of the method
  - If you want to update the UI, you will need to dispatch back to the main queue

```swift
DispatchQueue.global.async {
    DispatchQueue.main.async {
        // Call UI functions with results from other queue
    }
}
```

Multithreading DEMO
More on Concurrent Programming

• Grand Central Dispatch (GCD)
• GCD Tutorial with Examples
  – https://www.raywenderlich.com/148513/grand-central-dispatch-tutorial-swift-3-part-1

Multiple Tab, JSON App Demo