IS 543 Fall 2023

Mobile Platform Development

Foundation

UserDefaults

Project 1 Questions

Dr. Stephen W. Liddle
782 TNRB, office hours Tu 9:30-10:30am, Th 2-3pm
is543@byu.edu
801-422-8792 (rings on my cell)
Zoom: https://bit.ly/liddlezoom

Today

- Foundation
- UserDefaults persistent key-value store
- Updated algorithm for pushups-that-count
- Project 1 Q&A

Thoughts on Tuesday's Devotional

- Dr. Abigail Allen
  Professor of Accountancy

  How Do We Carry His Name?

  "Disciples of Jesus Christ are not perfect, nor do they expect others to be; rather, they trust that only by and through Him perfection is possible."

Fun Facts

- What is Liddle's Law?
  - A) Everyone should love a good coding frenzy
  - B) Refactoring is always worthwhile
  - C) There are infinite solutions to every problem
  - D) You're never done programming

Any and AnyObject

- `Any` can refer to any type at all, except a function type

```
let anyMix: [Any] = [0, 1, 3.14, "Mars", (5, 5)]
// array of two Ints, one Double, one String, one tuple
```

- `AnyObject` can only refer to any class instance

```
let anyMix: [AnyObject] = [0, 1, 3.14, "Mars", (5, 5)]
// compile-time error: tuple isn't instance of a class

let objects: [AnyObject] = ["Mars", PlayingCard(), UIButton()]
// array of one String, one PlayingCard, one UIButton
```

Hold on! Aren't 0, 1, and 3.14 primitive values, not objects? Why are they allowed to be class instances?

Swift "bridges" Int/Double/etc. to NSNumber (more in a few minutes)

- Try to avoid using these, but sometimes you need them

  Objective-C does not have typed arrays

  Cocoa APIs often use generic collections (e.g. `[AnyObject]` or `[NSObject: AnyObject]`)

Foundation Framework

- Time to look at some Objective-C/Cocoa classes

  `NSObject`: the root class of all Objective-C classes

  Important methods: `copy()` and `mutableCopy()`

  Many foundation classes have mutable and immutable variants

  `NSArray`: generic, untyped array class that is immutable (contents cannot be changed)

  `NSMutableArray`: subclass of `NSArray`, contents can be changed

  `NSSet`/`NSMutableSet`: unordered collection of distinct objects

  `NSOrderedSet`/`NSMutableOrderedSet`: ordered collection of distinct objects

  `NSDictionary`/`NSMutableDictionary`: collection of key/value pairs

  `NSNumber`: object wrapper around primitive numeric types (integer, floating point, boolean)

  `NSValue`: generic wrapper for non-primitive, non-object types (like C structs)

  `NSDate`: date time objects

  `NSData`: "bag of bits" (e.g. the bytes that make up a PNG image or a compressed zip file)

Bridging Objective-C/Swift

- Swift has extensive mapping mechanisms
  Every Foundation class/method accessible in Objective-C is available in Swift as well
- When Swift provides its own types for corresponding Foundation classes, the mapping is usually automatic
  `NSString` bridges to Swift `String`
  `NSNumber` bridges to Swift `Int`, `UInt`, `Float`, `Double`, `Bool`
  `NSArray`/`NSMutableArray` bridges to Swift array `[AnyObject]?`
  `NSDictionary`/`NSMutableDictionary` bridges to Swift dictionary `[NSObject : AnyObject]?`
  `var` or `let` signals whether array or dictionary should be mutable
- Most of the time when you see `NSArray` or `NSString`, just think Swift array or string
  Sometimes you have to deal with it explicitly though (the compiler will usually help you with this)

Foundation History

- What does the NS stand for?
  "NeXTSTEP", the object-oriented operating system Steve Jobs' team developed at NeXT

Swift 3 Dropped Many "NS" Prefixes

- A major feature of Swift 3 was improved API naming consistency, including dropping "NS" prefix from key Foundation classes
  See http://bit.ly/2cqZBDP for a comprehensive list of changes

  We don't drop "NS" for these classes:
  - Classes that are specifically for Objective-C, like `NSObject`
  - Classes that are platform-specific, like `NSUserNotification`
  - Classes that have a value-type equivalent, like `NSArray`, `NSString`, `NSDate`
    - But the value-type structs used in bridging also exist: `String`, `Date`

  We do drop "NS" for other Foundation classes:
  `DateFormatter`, `Operation`, `Thread`, etc.

  Some classes get renamed in the process:
  e.g. `NSTask` becomes `Process`

Property List

- The term "Property List" just means a collection of collections
  It's just a phrase, not a language thing, that means any graph of objects containing only:
  `NSArray`, `NSDictionary`, `NSNumber`, `NSString`, `NSDate`, `NSData` (or subclasses thereof)
- An `Array` is a Property List if all its members are too
  E.g. an `Array` of `String` is a Property List
  So is an `Array` of `Array` as long as those nested `Arrays`' members are Property Lists
- A `Dictionary` is a Property List only if all keys and values are too
  An `Array` of `Dictionarys` whose keys are `Strings` and values are `NSNumbers` is one
- Why define this term?
  The SDK has a number of methods that operate on Property Lists
  Usually to read them from somewhere or write them out to somewhere; example:
  `writeToFile(path: String, atomically: Bool)`
  This can (only) be sent to an `Array` or `Dictionary` that contains only Property List objects

Per-App Key/Value Store

- `UserDefaults` (formerly `NSUserDefaults`)

  Lightweight storage of Property Lists

  It's basically a `Dictionary` that persists between launches of your app

  Not a full-on database, so only store small things like user preferences

  Read and write via a shared instance obtained via class property `standard`

```
UserDefaults.standard.set(rvArray, forKey: "recentlyViewed")
```

  Sample methods:

```
set(_ value: Double, forKey defaultName: String)
integer(forKey defaultName: String) -> Int
array(forKey defaultName: String) -> [Any]?
```

Applying UserDefaults

- How could we use `UserDefaults` in our calculator app?
  For example, to store the user's sound preference (a Boolean value)
  Which layer of MVVM does `UserDefaults` correspond to?

  To me it feels like generally a Model-layer element

  Model: the "what" of your app
  - Anything that persists between launches of your app probably belongs here

  View: the user interface layer
  - Displays information accessed by the ViewModel and reports User Intents to the ViewModel

  ViewModel: the controller that manages connections between Model and View
  - Focuses on (1) interpreting the Model and (2) supporting User Intents

Algorithm Update

- I spent some time updating the pushups-that-count algorithm

  Strategy:

  - Build a dictionary where key is the month-day and value is a triple of count, excess, and date
  - For each value that has excess > 0:
    - Find the next and previous days
    - If the next day has no count, apply the excess to the next day
    - Else if the previous day has no count, apply the excess to the previous day
    - Else if the next day has < 50, add excess to the next day
    - Else if the previous day has < 50, add excess to the previous day
    - Else we can't use any excess for this day
  - Then sum up all the counts in the dictionary

  It does require some validation, like making sure the next/previous day is in the range 8/3 to 11/10

Next Week

- No classes while I'm gone
  I'll see you on the 17th again

  But I've recorded plenty of material for you
  Time to start making progress on Project 1

  Questions?