Reading Assignment III: Most of the Rest of Swift

Objective

We're going to finish off the rest of the important topics in the document this week. Most notably Error Handling, Concurrency and the rest about Protocols.

Due

You should have all of this reading done before lecture 8.

Materials

You'll continue reading from the same document(s) (e.g. Swift Programming Language) as last week.

Swift Programming Language

Don't gloss over reading any `NOTE` text (inside gray boxes) since many of those things are quite important. However, if a `NOTE` refers to Objective-C or bridging, you can ignore it.

If there is a link to another section in the text, you don't have to follow that link unless what it links to is also part of this week's reading assignment.

Always read the overview at the top of each major section (e.g., in The Basics, be sure to read the part that starts "Swift is a new programming language for iOS …").

You've read everything in A Swift Tour except the last two sections.

Error Handling and Concurrency sort of go together. We'll be demonstrating concurrency in a couple of weeks. It can be a difficult concept to grasp, but it's an important one because we never want the responsiveness of our application blocked by long-running things like fetching things over the internet. You'll read about these in even more detail in the main sections of the Language Guide.

Concurrency
Error Handling

In the Language Guide area, read the following sections in the following chapters. It is important to understand the information in these sections for you to be able to continue to follow along in lecture, so don't blow off this reading.

The Basics

No more putting off learning about Error Handling in Swift!

- Error Handling
- Assertions and Preconditions

Strings and Characters

If you skipped this section the first two times, you must now read this!

- String Literals
- Initializing an Empty String
- String Mutability
- Strings Are Value Types
- Working with Characters
- Concatenating Strings and Characters
- String Interpolation
- Unicode
- Counting Characters
- Accessing and Modifying a String
- Substrings
- Comparing Strings
- Unicode Representations of Strings

Control Flow

Generally control transfer out of loops and such is to be avoided. It is, however, important to know how to check the availability of an API in a given OS or version.

- Control Transfer Statements
  - Continue
  - Labeled Statements
- Checking API Availability

Closures

Autoclosures are a bit of a niche syntax for closures. Good to know about. You won't be writing many of your own functions that use an autoclosure though.

Autoclosures

Enumerations

We probably want to avoid recursive enumerations, so we'll leave that out again this week.

Recursive Enumerations

Properties

Property observers are pretty simple (we'll be showing them in lecture next week). Property wrappers, on the other hand, are a bit more difficult to grasp but they are really important because they are the language feature that enables things like @State, @Published, @ObservedObject, etc. Do your best to understand how property wrappers work, but don't feel like you need to have mastered it enough to, for example, author your own property wrapper.

Property Observers
Property Wrappers

Subscripts

Subscripts can often be a clean syntax for accessing data in a collection of things or even to semantically get a value from an unstructured source of data. Having said that, it's probably not something you're going to do in your first 10 weeks of learning Swift. Read this entire section.

Inheritance

We're kind of going to gloss over object-oriented programming in Swift because (outside of your ViewModel), you're not going to use it much in SwiftUI. We'll put reading about this into the 4th and final reading assignment (which is mostly going to be "esoterica").

Initialization

Ditto the above about OOP. Here we will just read about the last non-object-oriented initialization topic: failable initializers. Failable initializers are actually used quite often. Basically they just support having an init function be allowed to return an Optional that is nil if it was not possible to initialize the struct or class from the arguments provided.

> Class Inheritance and Initialization
> Failable Initializers
> Required Initializers

Deinitialization

Deinitialization is another OOP/reference type topic.

Error Handling

Error handling is important to understand, especially as you advance into the more powerful APIs available in Swift and SwiftUI. In fact, Error Handling and Concurrency are the #1 topics in the reading this week. Read this entire section.

Concurrency

Concurrency is important because we don't want our zippy UIs to get blocked by slow-moving activity like fetching from the network or processing a big swathe of machine learning data or some such. Read this entire section.

Type Casting

Type casting has become more and more unnecessary as SwiftUI completely takes over from UIKit as the way people write iOS apps. So we'll leave this gray until the final reading assignment. Generally a well-formed Swift application would not use any type casting, but occasionally we see it for backwards compatibility with UIKit-era APIs. If you find yourself doing type casting in any other context, there's probably a better way to do it (via protocols and generics usually).

Extensions

Now that you've read about subscripts above, you can likely read this last unread section about extensions and understand it as well.

Subscripts

Protocols

Finishing off learning about protocols is one of the more prominent things in this reading assignment. However, we're going to leave out the sections below that are tied to the backwards-compatibility to UIKit (the old way of writing iOS apps) until our final reading assignment.

Initializer Requirements
Protocols as Types
Delegation
Collections of Protocol Types
Class-Only Protocols
Protocol Composition
Checking for Protocol Conformance
Optional Protocol Requirements

Generics

Now that you fully understand protocols (and subscripts, for that matter), you can finish off this section.

Associated Types
Associated Types with a Generic Where Clause
Generic Subscripts

Opaque Types (this explains “some View” in detail)

It’s really valuable to have the ability to express in your API that you want to pass around something that behaves like a protocol, but you don’t want to put any other restrictions on that thing. Opaque types allow this. This does probably fall into the category of “a tool used by relatively advanced functional programmers” and likely you’ll be able to write a SwiftUI app just fine without mastering this, but you should at least know what’s going on when you see things like some View in the code.

Automatic Reference Counting

This is how reference types get their memory cleaned up. Again, we mostly use value types, so we can postpone this to the last reading assignment.

Memory Safety

Yikes, if you have to concern yourself with stomping on memory in the heap, you’ve really reached the outer limits of Swift-ness! You should never have to worry about this in the normal course of writing a SwiftUI application.

Access Control

We really only delayed this reading until now to keep your reading load down. Note that until you start writing your own modules (Swift-speak for code libraries), many of the access control keywords (like public) will not be something you use. private and private(set) (and the default internal) are your go-to access control levels.

Custom Types
Subclassing
Constants, Variables, Properties, and Subscripts
Initializers
Protocols
Extensions
Generics
Type Aliases

Swift API Guidelines

Read this Swift API Guidelines document in its entirety.

Read this over **yet again** this week. You should now be fully a master of this information. As the quarter progresses, you should eventually become an *expert namer of properties, methods and other Swift constructs*. This will require you to refer back to this document often.

Be sure to click everywhere that it says "MORE DETAIL".

Pay special attention to the "Write a documentation comment" section.

Pay special attention to the "Follow case conventions" section.

Pay special attention to the entire "Argument Labels" section.

You can also ignore the final subsection of the final section "Special Instructions -> Take extra care with unconstrained polymorphism". We won't be doing anything with the `Any` and `AnyObject` types.