Leitura II: Mais Swift

Observação

Este material foi traduzido e adaptado do curso CS 193P - *iPhone Application Development* oferecido pela Universidade de Stanford através do iTunesU. Ele está protegido pela licença *Creative Commons Attribution-Noncommercial-Share*[1].

Objetivo

O objetivo de nossa primeira leitura foi conhecer melhor a linguagem que você está usando para desenvolver: Swift. Desta vez vamos aprender mais sobre ela.

Muitos de vocês não tiveram experiência com Objective-C, mas não se preocupe com isso. Nada na documentação de Swift realmente assume isso. Contudo, se você nunca programou em C (C#, Java, C++ ou qualquer outra variante), então Swift pode ser extremamente nova para você (mas tenho a esperança de que mesmo assim você aprenderá).

Materiais Necessários

Toda a leitura deve ser feita no documento *Swift Programming Language* (Linguagem de Programação Swift). Este documento está disponível em: https://developer.apple.com/library/mac/documentation/Swift/Conceptual/Swift_Programming_Language/index.html

Seções a Ler

Para que o seu tempo seja utilizado de forma mais útil e para enfatizar os conceitos mais importantes neste momento, as seções foram divididas em quatro categorias:

- **vermelhas** são seções MUITO IMPORTANTES e devem ser mais difíceis para serem entendidas. Elas devem ser lidas com bastante atenção;

1 http://creativecommons.org/licenses/by-nc-sa/3.0/us/

- **azuis** são seções importantes, mas não tão difíceis de serem entendidas;
- **verdes** são seções importantes, mas cobrem assuntos bastantes básicos e simples (muitos deles são parecidos com os encontrados na linguagem C);
- **cinzas** são seções que não precisam ser lidas neste momento. Provavelmente serão nas próximas semanas.

Não deixe de ler os textos que estão dentro dos retângulos acinzentados (*NOTES*), pois muitos deles são importantes.

Se há uma ligação a outra seção no texto, você não precisa seguir a ligação a não ser que também seja parte desta tarefa de leitura.

Na seção *Language Guide*, leia as seguintes seções dos seguintes capítulos:

The Basics

Numeric Literals
Numeric Type Conversion
Type Aliases
Tuples

Basic Operators

Nil Coalescing Operator

Strings and Characters

Ao iniciar uma *String* vazia, não ignore **initializer syntax**
Em *Strings Are Value Types*, a **NSString reference in NOTE** fará sentido desta vez.
Unicode
Unicode Representations of Strings

Não há nada em particular sobre Unicode que você terá que saber para a próxima etapa do trabalho, mas é sempre bom ter este conhecimento.

Collection Types

Em Arrays, a discussão sobre **initializers** deve fazer sentido a você.
Dictionaries

Functions

Em Function Parameters and Return Values, leia **Function with Multiple Return Values**
Em Function Parameters and Return Values, leia **Optional Tuple Return Types**
Function Parameters Names

Closures

Revisitaremos os seguintes dois tópicos em algumas semanas quando eles realmente se tornarem importantes, mas vá em frente e leia-os agora para terminar os sete primeiros capítulos do documento de referência. Eles não são conceitos complicados, por si, mas até que você inicie a utilização da ideia, podem ser um pouco abstratos. Você não precisa capturar valores em seus *closures* nos primeiros três trabalhos, então não vá tão fundo.

Capturing Values
Closures are Reference Types

Enumerations

Read this entire chapter

Já que enumerações são um pouco mais poderosas em Swift do que em qualquer outra linguagem, haverá uma demonstração extensiva de enumerações na aula 3, mas não cobrirá valores crus (cujo uso não é realmente fortemente encorajado de qualquer forma - há algumas circunstâncias nas quais é "apenas a coisa certa a se usar", mas em muitas outras circunstâncias as pessoas estão simplesmente revertendo aos seus conceitos antigos de enumerações).

Classes and Structures

Classes e estruturas são muito similares em Swift (iniciadores, funções, propiedades, etc), mas existem diferenças importantes (tipo por valor x por referência, herança, etc) e este capítulo os

apresenta. Leia com cuidado.

Structures and Enumerations are Value Types
Classes are Reference Types
Choosing Between Classes and Structures
Assignment and Copy Behavior for Strings, Arrays, and Dictionaries

Properties

Em *Stored Properties*, **Lazy Stored Properties**
Property Observers
Global and Local Variables
Type Properties

Methods

Instance Methods (especialmente tudo sobre **Parameter Names**)
Type Methods

Subscripts

Subscript Syntax
Subscript Usage
Subscript Options

Inheritance

Defining a Base Class
Subclassing
Overriding
Preventing Override

Initialization

O entendimento completo deste tópico é bastante duro sem a leitura do capítulo inteiro, mas é muito para ser lido quando combinado com toda a leitura acima, então vamos nos concentrar no

básico. A propósito, a situação de iniciação de *UIViewController* é bastante complicada, então tem um adendo bem detalhado sobre isso neste item.

Setting Initial Values for Stored Properties
Customizing Initialization
Default Initializers
Initializer Delegation for Value Types
Class Inheritance and Initialization
Failable Initializers
Required Initializers
Setting a Default Property Value with a Closure or Function

Este último tópico é bem legal, mas preste bastante atenção à caixa *NOTE* no centro.

OBSERVAÇÃO: você não deveria precisar de um iniciador de *UIViewController* para o trabalho 1 ou trabalho 2 (e esperançosamente para nenhum trabalho em todo o semestre!). Então, a não ser que discorde disso, pode pular este adendo e passar para o próximo tópico de leitura!

Aqui está o mínimo que você precisa saber se, por algum motivo, absolutamente precisa de um iniciador em sua subclasse *UIViewController* (novamente, esperançosamente nunca).

A classe *UIViewController* tem dois iniciadores e, ambos, ou nenhum, devem ser implementados em uma subclasse...

```
override init(nibName nibNameOrNil: String?, bundle nibBundleOrNil: NSBundle?) {
super.init(nibName: nibNameOrNil, bundle: nibBundleOrNil)
<your initialization code here>
}
```

e

```
required init(coder aDecoder: NSCoder) {
super.init(coder: aDecoder)
<your initialization code here>
}
```

Não esqueça das palavras-chave *override* e *required*. Obviamente, você provavelmente quer fatorar seu código de iniciação em algum outro método que você chamará de ambos estes dois.

Mas, se você pode evitar implementá-los (na maioria das vezes esse é o caso), por favor faça isso. É um artefato histórico chato (em minha humilde opinião). Muito da iniciação de *UIViewController* ocorre no seguinte método do ciclo de vida de um *View Controller* (sobre o qual falaremos na aula):

```
override func viewDidLoad() {
super.viewDidLoad()
<your initialization code here>
}
```

Quando este método é chamado, todos os *outlets* foram conectados, mas o *View Controller* ainda não está visível na tela, então este é um bom lugar para colocar todo o seu código de iniciação. Eu recomendaria que você sempre projete a subclasse de *UIViewController* de tal forma que a iniciação possa ocorrer neste método ao invés de programar com os (de certa forma arcanos) iniciadores de *UIViewController*. Não esqueça da estratégia de fazer com que uma propriedade seja um *Optional* implicitamente aberto se tiver que (e inicie-o em *viewDidLoad*). Esta é a forma como *UIViewController* manipula *outlets* (embora ele os inicie antes de *viewDidLoad* ser chamado, não em *viewDidLoad* em si).

Optional Chaining

A leitura deste capítulo é opcional. É uma maneira muito boa de fazer as coisas de forma bem concisa (e legível também), mas se o conceito completo de *Optional* não foi entendido completamente por você ainda, você pode achar este capítulo difícil agora.

Type Casting

Enquanto Swift é extremamente seguro em tipos, iOS tem crescido com uma história que não é tão restrita. Assim, existem inúmeras ocasiões, quando trabalhamos com a API do iOS, em que você tem um ponteiro para um objeto e você desejará saber a qual classe ele pertence e/ou você desejará fazer um *cast* para uma certa classe (se possível). Swift fornece meios de fazer isso de forma segura e este capítulo discute isso.

Todo o capítulo (embora eu creia que não utilizaremos o tipo *Any* neste curso)

Nested Types

Todo o capítulo

Optional Chaining

Por enquanto, tudo que você precisa saber sobre Generics é como usá-los. É bastante direto. Por exemplo, *Array <Double>* (matriz de Double) ou *Dictionary <String, Int>* (dicionário cujas chaves são *String* e valores são *Int*). Você pode ler este capítulo se deseja entender mais sobre isso, mas é um mecanismo bem poderoso e você pode ganhar muito com isso agora, então **a leitura deste capítulo é opcional neste ponto**.

Access Control

Mais uma vez, você tem muito para ler, então não farei você ler este capítulo agora, mas o sumário executivo é que nós vamos começar a colocar a palavra-chave *private* na frente de tudo da API que escrevermos, a não ser que realmente pretendamos (e estejamos preparados para suportar) outro código em nossa aplicação para chamar o método ou propriedade em questão. Quando você está aprendendo a desenvolver, sempre imagine que está trabalhando em uma equipe de programadores que poderiam querer usar o código que está desenvolvendo. O controle de acesso permite que você expresse propriamente o "nível de suporte" que um dado método ou propriedade receberá no futuro.

Advanced Operators

Overflow Operators

É possível que você possa querer usar estes operadores se você fizer o requerimento extra no trabalho 2 para relatar erros e também decidir relatar erros de estouro aritmético.