Reading Assignment II: Mehr zu Swift

Ziele

Das Ziel des ersten Reading Assignments war es, die Grundlagen der neuen Sprace Swift zu erlernen. Wir gehen nun dazu über mehr über Swift zu lernen.

Die meisten von Ihnen haben wahrscheinlich keine Erfahrung in Objective-C, machen Sie sich darüber aber keine Gedanken, da nichts in der Swift Dokumentation dies voraussetzt.

Lesen alles referenzierte Material bis zur Woche 3.

Material

- Die Reading Assignments stammen aus zwei Online Dokumenten: The Swift Progamming Language und den Swift API Guidelines.

Swift Programming Language

Lesen Sie die unten beschriebenen Sektionen in Swift Progamming Language. Um Ihre wertvolle Zeit besser zu nutzen und auf die wichtgen Konzepte zu konzentrieren, wurden der Lesestoff in vier Kategorien unterteilt.

Rote Sektionen sind SEHR WICHTIG und sind wahrscheinlich schwieriger zu verstehen. Lesen Sie diese sehr aufmerksam.

Gelbe Sektionen sind wichtig, sind aber nicht schwierig zu verstehen.

Grüne Sektionen sind wichtig, decken aber einfache Grundlagen ab (das meiste ist wie C).

Ausgeraute Sektionen müssen Sie für dieses Assignemt (noch) nicht lesen. Sie sind vielleicht später im Semester wichtig.

Überfliegen Sie nicht den NOTE Text (in den grauen Kästen) - viele dieser Dinge sind sehr wichtig. Wenn sich eine NOTE jedoch auf Objective-C oder Bridging bezieht, können Sie sie überspringen.

Wenn ein Link im Text auf eine andere Sektion verweist, müssen Sie diesem nicht folgen, außer der verlinkte Teil ist ebenfalls Teil des Reading Assignments.

Im Language Guide Bereich, lesen Sie die Sektionen in den folgenden Kapiteln:

The Basics

Numeric Literals
Numeric Type Conversion
Type Aliases
Tuples
Error Handling

Basic Operators

Nil Coalescing Operator

Strings and Characters

In Initializing an Empty String, ignorieren Sie nun nicht mehr die "Initializer Syntax".
In Strings Are Value Types sollte die NSString Referenz in NOTE nun Sinn ergeben.

Unicode
Accessing and Modifying a String
Unicode Representations of Strings

Es gibt nichts spezielles was Sie über Unicode für diesen Kurs wissen müssen, aber es ist dennoch gut ein paar Dinge darüber zu wissen.

Collection Types

Die Diskussion zu Initializers in Arrays sollte nun Sinn ergeben.

Sets (ignorieren Sie das erste NOTE Kästchen in dieser Sektion)
Performing Set Operations

Control Flow

In Conditional Statements, lesen Die Tuples, Value Bindings & Where (letztes mal übersprungen).
Wahrscheinlich nutzen Sie die nächsten beiden niemals, der Vollständigkeit halber, überfliegen Sie sie...

Early Exit
Checking API Availability

Functions

In Function Parameters and Return Values, lesen Sie Functions with Multiple Return Values.
In Function Parameters and Return Values, lesen Sie Optional Tuple Return Types
In Function Argument Labels and Parameter Names, das Lesen über Variadic und In-Out Parameters obliegt Ihnen, aber nutzen Sie keines davon in diesem Kurs.

Closures

Wir kommen auf die folgenden Themen in den nächsten Wochen zurück, wenn diese wirklich wichtig werden, fahren Sie aber schon mal fort und lesen Sie sie die ersten sieben Kapitel des Referenzdokumentes jetzt. Dies sind keine schwierigen Konzepte per se, bis Sie aber anfangen diese Ideen anzuwenden, sind sie vielleicht ein wenig abstrakt. Sie müssen in den ersten drei Assignments keine Values capturen, also machen Sie sich nicht zu viele Gedanken dazu.

Capturing Values
Closures are Reference Types
Nonescaping Closures
Autoclosures

Enumerations

Raw Values
Recursive Enumerations

Classes and Structures

Classes und Structures sind sehr ähnlich in Swift (Initializers, Functions, Properties, etc.), aber es gibt wichtige Unterschiede (Value vs. Reference Type, Vererbung, etc.) und dieses Kapitel stellt diese vor. Lesen Sie sorgfältig.

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

In Stored Properties, Lazy Stored Properties (letztes mal übersprungen).
In Stored Properties, Stored Properties and Instance Variables (letztes mal übersprungen)
Property Observers
Global and Local Variables
Type Properties

Methods

Stellen Sie sicher, dass Sie den Unterschied zwischen einer Instanzmethode und einer Typmethode verstehen.

Instance Methods (speziell alles über Parameter Names)
Type Methods

Subscripts

Subscript Syntax (dies ist optional zu lesen, aber ziemlich cool)
Subscript Usage
Subscript Options (ebenfalls optional, aber cool)

Inheritance

Defining a Base Class
Subclassing
Overriding
Preventing Overrides

Initialization

Es ist schwer dies vollständig zu verstehen ohne das komplette Kapitel zu lesen. Da es aber sehr viel zu lesen ist in Kombination mit allem von oben, konzentrieren wir uns auf das Wesentliche. Nebenbei, die Initializer Situation von `UIViewController` ist sehr kompliziert, daher ein kleiner Zusatz weiter unten.

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 Propery Value with Closure or Function

(Letzteres ist ziemlich cool, richtigen Sie Ihre Aufmerksamkeit auf das NOTE Kästchen in der Mitte).

Bemerkung: Sie sollten keinen `UIViewController` Initializer für Assignment I oder Assignement II benötigen (und hoffentlich für gar kein Assignment in diesem Kurs). Wenn Sie dem zustimmen, überspringen sie folgendes einfach!

Aber... im Interesse der Vollständigkeit... hier ist das absolute Minimum was Sie wissen müssen, wenn Sie das Gefühl haben dass Sie unbedingt Initializer in Ihrer `UIViewController` Subclass haben müssen (nochmal, hoffentlich niemals).

`UIViewController` hat zwei Initializer und beide (oder **keiner**) sollte in einer Subclass implementiert sein...

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

und

```
required init?(coder: aDecoder: NSCoder) {
   super.init(coder: aDecoder)
   <init code hier>
}
```

Vergessen Sie nicht die `override` und `required` Schlüsselworte. Offensichtlich wollen Sie Ihren Initialisierungscode in eine andere Methode auslagern, die Sie dann von diesen beiden aufrufen können.

Wenn Sie allerdings vermeiden können diese zu implementieren (was Sie in den meisten Fällen können), tun sie dies auch. Es ist ein historisches Artefakt. Die Initialisierung der meisten `UIViewController` erfolgt in der folgenden View Controller Lifecycle Methode (welche wir in der Vorlesung besprechen):

```
override fund viewDidLoad() {
   super.viewDidLoad()
   <init code hier>
}
```

Wenn dies aufgerufen wird, sind Ihre Outelts bereits alle gesetzt, aber der View des Controllers ist noch nicht on-screen. Daher ist dies ein großartiger Ort alle one-time Initialization zu platzieren. Es ist empfohlen Ihre `UIViewController` Subclass immer so zu designen, dass die Initialisierung hier erfolgen kann statt mit den obskuren Initializern des `UIViewControllers` herumzuspielen. Vergessen Sie nicht die Strategie ein Property zu einem *implicitly unwrapped Optional* zu machen (und es in `viewDidLoad` zu initialisieren) wenn Sie dies müssen. So handhabt `UIVierController` Outlets (obwohl es diese initialisiert ganz kurz bevor `viewDid-Load` aufgerufen wird, nicht in `viewDidLoad` selbst).

Optional Chaining

Das lesen dieses Kapitels ist (im wahrsten Sinne des Wortes) optional. Es ist eine sehr coole Möglichkeit Dine kurz und präzise (und sehr lesbar) zu machen. Wenn Sie aber das komplette Konzept von Optionals noch nicht vollständig verstanden haben, ist dieses Kapitel zum jetzigen Zeitpunkt vielleicht etwas zu viel.

Type Casting

Während Swift selbst sehr Type Safe ist, ist iOS mit einer Historie gewachsen, die nicht so strikt ist. Daher kommt es beim Arbeiten mit der iOS API häufig vor, dass Sie einen Pointer zu einem Objekt haben und Sie wissen wollen welche Klasse es ist und/oder Sie in eine bestimmte Klasse casten wollen (falls dies möglich ist). Swift stellt Wege zur Verfügung dies sicher zu tun und diese werden in diesem Kapitel beschrieben.

Komplettes Kapitel (obwohl wir wahrscheinlich nicht den Typ `Any` in laufe des Semesters verwenden werden).

Nested Types

Komplettes Kapitel

Generics

Alles was Sie aktuell über Generics wissen müssen ist, wie man sie nutzt. Es ist ziemlich eindeutig. Zum Beispiel `Array<Double>` (Array von `Double`s) oder `Dictionary<String, Int>` (Dictionary dessen Keys `String`s und dessen Values `Int`s sind). Die können dieses Kapitel lesen, wenn Sie mehr darüber erfahren wollen, allerdings ist es sehr mächtig und Sie haben sowieso schon sehr viel zu tun, daher ist das Kapitel optional zum jetzigen Zeitpunkt.

Access Control

Nochmal, da Sie viel zu lesen haben, ist dieses Kapitel hier und jetzt optional. Wir werden jedoch dazu übergehen das Schlüsselwort `private` vor alle API zu setzen die wir schreiben, außer wir beabsichtigen wirklich (und sind bereit dies zu supporten), dass anderer Code in unserer Applikation die in Frage stehende Methode oder Property aufruft. Wärend Sie entwickeln, stellen Sie sich immer vor in einem Team von Programmierern zu arbeiten, die vielleicht den Code nutzen möchten den Sie schreiben. Access Control lässt Sie den "Support Level" einer Methode ausdrücken, den diese in Zukunft erhält.

Advanced Operators

Overflow Operators

Es ist möglich dass Sie dies nutzen wollen, wenn Sie arithmetische Overflow Fehler in Ihrem Taschenfrechner behandeln wollen.

Swift API Guidelines

Sie sollten die Swift API Guidelines von Reading Assignment I vollständig gelesen haben. Nachdem Sie alles von oben gelesen haben, lesen Sie die Guidelines noch mal. Da Sie mehr und mehr über Swift lernen, sollten die Guidelines nun ebenfalls mehr und mehr Sinn ergeben.

Stellen Sie sicher überall drauf zu klicken wo “MORE DETAIL” steht.

Richten Sie Ihre Aufmerksamkeit auf die “Write a documentation comment” Sektion.

Richten Sie Ihre Aufmerksamkeit auf die “Follow case conventions” Sektion.

Sie können weiterhin den zweiten Teil (unconstrained polymorphism) der letzten Special Instructions Sektion ignorieren, Sie sollten aber in der Lage sein den ersten Teil (label closure parameters and tuple members) zu verstehen.