

App-Entwicklung für iOS und OS X

SS 2016
Stephan Gimbel



View Controller Lifecycle

View Controller haben einen “Lifecycle”

Eine Sequenz von Nachrichten (Funktionsaufrufe) wird an den View Controller geschickt, über den Zeitraum seiner “Lebenszeit”.

Wieso ist dies wichtig?

Es ist üblich diese Funktionen zu überschreiben um bestimmte Dinge zu erledigen.

Beginn des Lifecycle...

Erstellung.

MVCs werden meistens aus dem Storyboard installiert (wie gesehen).

Es existieren Möglichkeiten dies in Code zu machen (selten).

Und dann...?

Vorbereitung zu Segue.

Outlet Setting

Appearing und Disappearing.

Geometrie Änderung.

Wenig-Speicher Situationen.

View Controller Lifecycle

Nach Installierung und Outlet-Setting wird viewDidLoad ausgeführt
Sehr guter Ort um viel setup-Code zu platzieren.
Besser als init, da hier bereits alle Outlets gesetzt sind.

```
override func viewDidLoad() {  
    super.viewDidLoad()  
    // setup code  
}
```

Eine Sache die wir hier vielleicht machen möchten ist das Update des UI aus dem Model. Da wir wissen, dass hier alle Outlets gesetzt sind.

Aber Vorsicht, da die Geometrie des Views (dessen Bounds) hier noch nicht gesetzt ist. Zu diesem Zeitpunkt können wir nicht sicher sein, ob wir auf einem iPhone 5-Screen, auf einem iPad oder ??? sind.

Daher hier keine Dinge initialisieren, die Geometrie-abhängig sind.

View Controller Lifecycle

Kurz bevor der View auf dem Screen auftaucht, werden wir benachrichtigt

```
func viewWillAppears(animated: Bool) // animated gibt an, ob der View  
// über eine bestimmte Zeit  
// auftaucht.
```

Der View wird nur ein mal “geladen”, taucht aber vielleicht viele male auf und verschwindet wieder.

Hier also nichts machen, was besser in viewDidLoad aufgehoben wäre.

Ansonsten machen wir unnötigerweise bestimmte Dinge wieder und wieder.

Hier Dinge machen, die sich vielleicht ändern, wenn der MVC off-Screen ist.

Kann zur Optimierung der Performance genutzt werden, durch warten bis die Funktion aufgerufen ist (im Gegensatz zu viewDidLoad) um eine teure Operation zu starten (vielleicht in einem anderen Thread).

Die View Geometrie ist hier gesetzt, aber es existieren andere Orte um auf Geometrie zu reagieren.

Es existiert ebenfalls eine “did” Version

```
func viewDidAppear(animated: Bool)
```

View Controller Lifecycle

Wir werden auch benachrichtigt, wenn der View vom Screen verschwindet
Hier kommt der Code “zum erinnern” und Clean-Up-Code hin.

```
override func viewWillDisappear(animated: Bool) {  
    super.viewWillDisappear(animated)  
    // clean up code, da der View off-Screen ist  
    // Nichts zu Zeitaufwändiges machen, sonst wird die App langsam  
    // Vielleicht in einem extra Thread machen  
}
```

Auch hiervon existiert eine “did” Version

```
func viewDidAppear(animated: Bool)
```

View Controller Lifecycle

Änderungen der Geometrie?

Meistens passiert das automatisch, durch Autolayout.

Wir können dies aber auch direkt über diese Funktionen beeinflussen ...

```
func viewWillLayoutSubviews()  
func viewDidLayoutSubviews()
```

Wird immer dann ausgeführt, wenn der View's frame sich ändert und dessen subviews neu ausgelegt werden.

Zum Beispiel bei autorotation.

Hier können wir frames der subviews **resetten** oder **Geometrie-Abhängige Properties setzen**.

Zwischen "will" und "did" erfolgt das autolayout.

Diese Funktionen werden vielleicht häufiger aufgerufen als gedacht (z.B. pre- und post-animation Ausrichtung, usw.)

Hier also nichts machen, dass nicht korrekt (und effizient) wiederholt werden kann.

View Controller Lifecycle

Autorotation

Normalerweise ändert sich die Form, wenn der User das Device zwischen Portrait/Landscape rotiert.

Wir können kontrollieren welche Orientierung unsere App unterstützt (in den Project Settings)

Wenn wir aber z.B. die Rotation Animation beeinflussen wollen, dann können wir das über diese Funktion tun ...

```
func viewWillTransitionToSize(  
    size: CGSize,  
    withTransitionCoordinator: UIViewControllerTransitionCoordinator) {  
}
```

Der coordinator stellt eine Methode zur Animation zusammen mit der Rotation Animation zur Verfügung.

Wir betrachten dies zum jetzigen Zeitpunkt nicht näher (vielleicht später), wollen aber im Hinterkopf behalten, dass dies existiert.

View Controller Lifecycle

In Low-Memory Situationen wird `didReceiveMemoryWarning` aufgerufen ...

Dies passiert selten, aber gut designeder Code mit hohem Speicherbedarf erwartet dies.

Beispiel: Bilder und Audio.

Alle "großen" Objekte die aktuell nicht gebraucht werden und schnell wieder erstellt werden können, sollten released werden (durch setzen von Pointern auf nil)

View Controller Lifecycle

awakeFromNib

Dies wird für alle Objekte aufgerufen, die aus dem Storyboard kommen (auch für den Controller)

Geschieht bevor Outlets gesetzt sind! (bevor der MVC geladen ist)

Nach Möglichkeit Code an anderer Stelle platzieren (z.B. viewDidLoad oder viewDidAppear)

View Controller Lifecycle

Zusammenfassung

Instanziert (normalerweise aus dem Storyboard)

awakeFromNib

Segue Preparation

Outlets werden gesetzt

viewDidLoad

Diese Paare werden jedesmal aufgerufen wenn der Controller' view on-/off-Screen geht

viewWillAppear und viewWillAppear

viewWillDisappear und viewWillDisappear

Die "Geometry changed" können jederzeit nach viewDidLoad aufgerufen werden...

viewWillLayoutSubviews (... dann autolayout, dann...) viewDidLayoutSubviews

Wenn der Speicher knapp wird...

didReceiveMemoryWarning

Autolayout

Wir kennen bereits viel von Autolayout

Gestrichelte blaue Linien die Xcode sagen, was wir erreichen möchten.

Ctrl-Drag zwischen Views um Relationships zu erstellen (Spacing, etc.).

“Pin” und “Arrange” Popovers unten rechts im Storyboard.

Reset to Suggested Constraints (wenn die blauen Linien ausreichend waren für eindeutige Constraints).

Document Outline (alle Constraints anzeigen, Fehlplatzierungen und Konflikte beseitigen).

Size Inspector (Details der Constraints eines selektierten Views anzeigen und editieren).

Constraint anklicken zur Auswahl und Anzeige (und Edit) im Attribute Inspector

Dies alles zu beherrschen erfordert Erfahrung

Nur Übung macht den Meister.

Autolayout geht aber auch aus Code heraus

Es ist allerdings meist besser es im Storyboard zu machen, wenn dies möglich ist.

Autolayout

Rotation?

Manchmal ändert eine Rotation die Geometrie so stark, dass Autolayout nicht ausreichend ist.

Die Views müssen dann neu positioniert werden, damit sie wirklich passen.

Calculator

Was passiert bei einem Taschenrechner mit 20 Buttons?

Im Landscape Mode ist es vielleicht besser 5 Buttons in einer Reihe und 4 in einer Spalte zu haben.

Im Portrait Mode sind es vielleicht besser 4 Buttons in der Reihe und 5 in der Spalte.

View Controller haben dies in anderen Situationen vielleicht auch

Vielleicht ist unser MVC der Master eines Side-by-Side Split Views.

In einem solchen Fall wollen wir vielleicht wie ein iPhone im Portrait Mode zeichnen.

Die Lösung? Size Classes

Unser View Controller existiert immer in einer bestimmten "Size Class" Umgebung für Breite und Höhe.

Zur Zeit ist es entweder Compact oder Regular (also nicht compact)

Autolayout

iPhone 6+

Das iPhone 6+ in Portrait Orientierung ist Compact bzgl. Width und Regular bzgl. Height.
Im Landscape Mode ist es Compact in Height und Regular in Width.

iPhone

Andere iPhones im Portrait Mode sind auch Compact bzgl. Width und Regular bzgl. Height.
Aber im Landscape Mode sind non-iPhone 6+ Compact bzgl. beider Dimensionen.

iPad

Immer Regular bzgl. beider Dimensionen.

Ein MVC welcher der Master eines Side-by-Side Split View ist, ist Compact bzgl. Width und Regular bzgl. Height.

Erweiterbar

Das Konzept ist erweiterbar auf jede Situation eines "MVCs innerhalb eines anderen MVC" (nicht nur Split View).

Ein MVC kann seine Size Class Umgebung erhalten via dieser Funktion in UIViewController...

```
let mySizeClass: UIUserInterfaceSizeClass = self.traitCollection.horizontalSizeClass
```

Der Rückgabewert ist ein enum .Compact oder .Regular (oder .Unspecified)

Size Classes

Compact

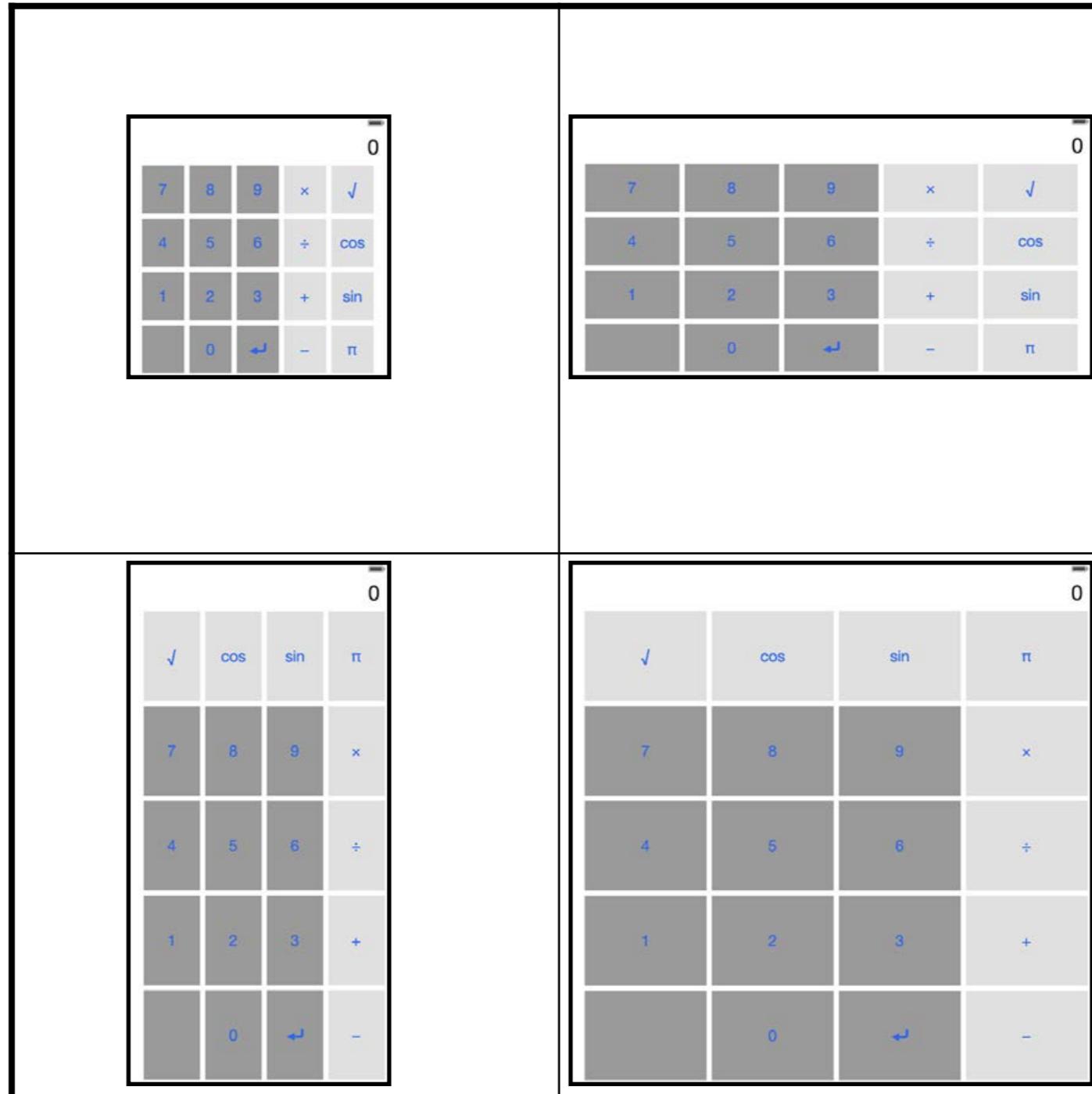
Compact

Regular

Vertical

Regular

Horizontal



Size Classes

