App-Entwicklung für iOS und OS X

SS 2016

Stephan Gimbel

UIColor

- Farben können mittels `UIColor` gesetzt werden
  - Es existieren Funktion für Standard-Farben, z.B.
    `let green = UIColor.greenColor()`
  - Farben können auch erstellt werden aus RGB-, HSV-Werten oder einem Pattern (mittels `UIImage`)
- Hintergrund Farbe eines Views
  `var backgroundColor: UIColor`
- Farben können einen Alpha-Wert haben (Transparenz)
  `let transparentYellow = UIColor.yellowColor()`
  - Alpha liegt zwischen 0.0 (transparent) und 1.0 (opaque)
- Wenn im View mit Transparenz gezeichnet werden soll…
  - Dann muss das System darüber benachrichtigt werden durch setzen von `var opaque = false` im `UIView`
- Der gesamte `UIView` kann transparent gemacht werden
  `var alpha: CGFloat`

View Transparenz

Was passiert bei überlappenden Views die Transparenz haben?

Wie bereits bekannt, bestimmt die Reihenfolge in der `subviews` Liste welcher View vor einem anderen liegt.

Niedrigere (früher im Array) können durch transparente Views, die über ihnen liegen "durchscheinen".

Transparenz ist teuer, bitte mit Bedacht verwenden.

Verstecken von Views ohne sie aus der View Hierarchie zu entfernen

```
var hidden: Bool
```

Ein versteckter View zeichnet nichts on Screen und bekommt auch keine Events.

Nicht so ungewöhnlich wie vielleicht anfangs gedacht, um einen View temporär zu verstecken.

Zeichnen von Text

Normalerweise wird ein `UILabel` verwendet um Text on Screen darzustellen

In manchen Fällen soll aber vielleicht Text in einem `drawRect` gezeichnet werden.

Um in drawRect zu zeichnen, verwenden wir `NSAttributedString`

```
let text = NSAttributedString(string: “hello”)
text.drawAtPoint(NSPoint)
let textSize: NSSize = text.size // wie viel Platz der String
                                 // braucht
```

Mutability wird mittels `NSMutableAttributedString` realisiert

Anders als bei String (wo `let` immutable und `var` mutable ist).

Hier verwenden wir eine andere Klasse für mutable attributed Strings...

```
let mutableText = NSMutableAttributedString(string: “some
string”)
```

`NSAttributedString` ist kein `String` oder `NSString` wird mittels

Wir können dessen Inhalte als String über das `string` oder `mutableString` Property erhalten.

Attributed Strings

Setzen von Attributen eines Attributed Strings

```
func setAttributes(attributes: Dictionary, range: NSRange)
func addAttributes(attributes: Dictionary, range: NSRange)
```

Hinweis! Hierbei handelt es sich um pre-Swift API.

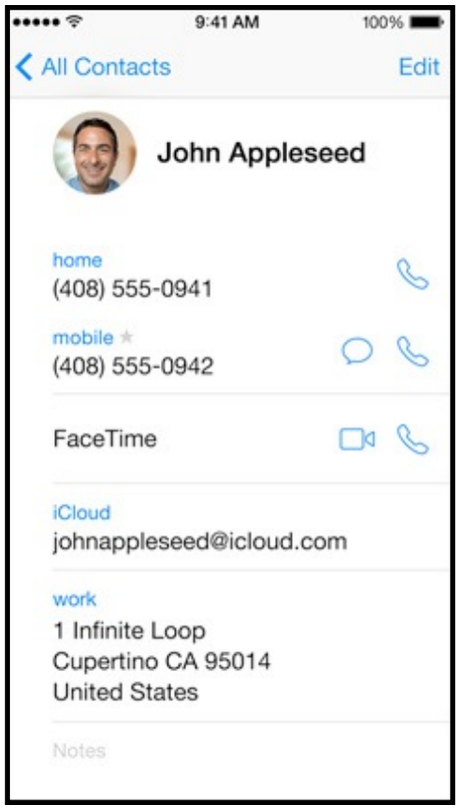
Attribute

```
NSForegroundColorAttributeName: UIColor
NSStrokeWithAttributeName: CGFloat
NSFontAttributeName: UIFont
```

Und viele mehr... siehe Dokumentation.

Fonts

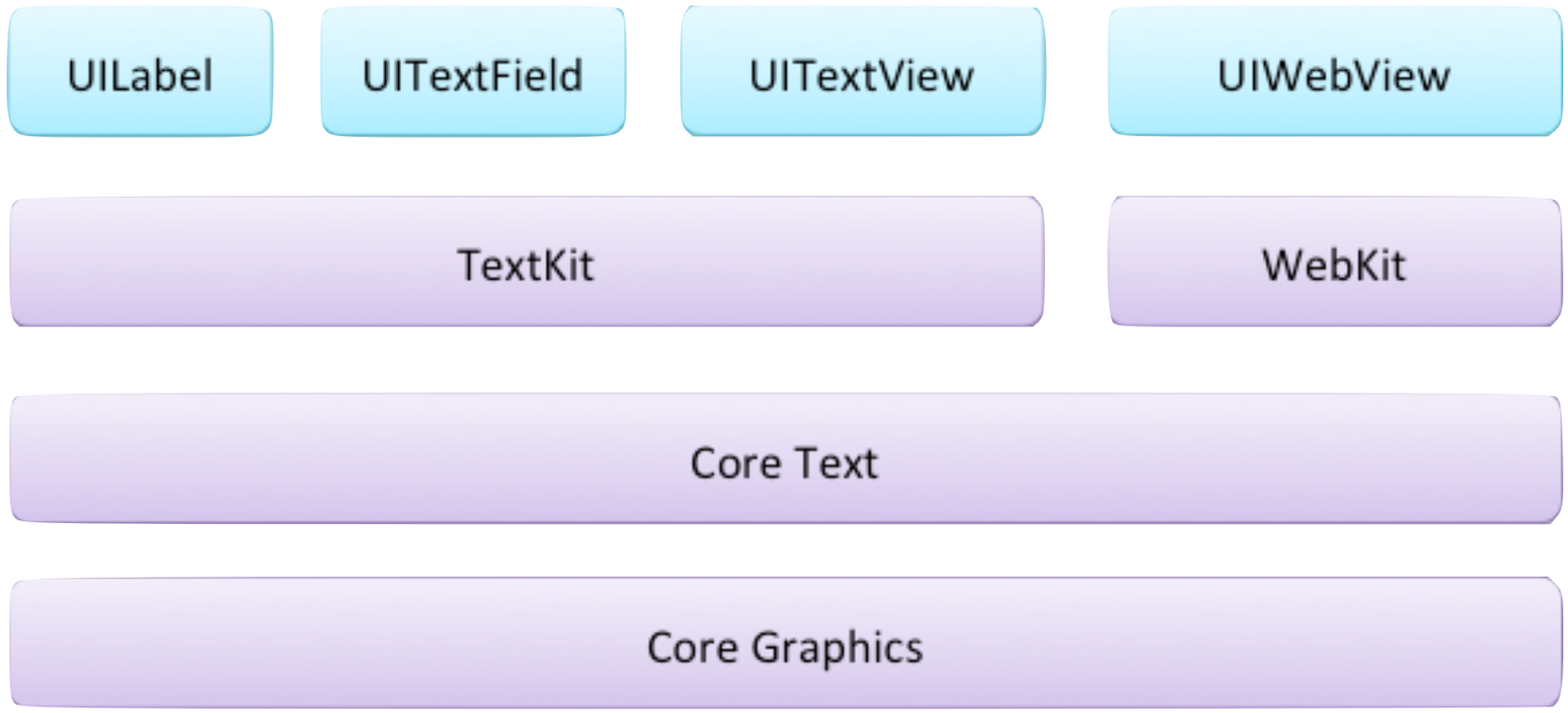
Es ist wichtig Fonts in iOS richtig zu wählen
Dies ist fundamental für Look & Feel des UI.



Fonts

Ab iOS 7 TextKit

- Dynamic Type
- Letterpress Effect
- Exclusion Paths
- Dynamic Text Formatting and Storage
- …



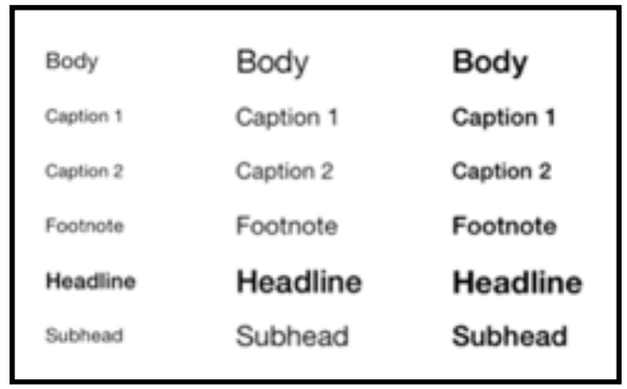
Fonts

- Um Font in Code zu erhalten
  - Font für einen gegebenen Text Style (z.B. body, etc.) erhalten, mittels `UIFont` Type Funktion...

```
class fun preferredFontForTextStyle(UIFontTextStyle) -> UIFont
```

  - Einige Styles (siehe `UIFontDescriptor` Doku für mehr)

```
UIFontTextStyle.Headline
UIFontTextStyle.Body
UIFontTextStyle.Footnote
```



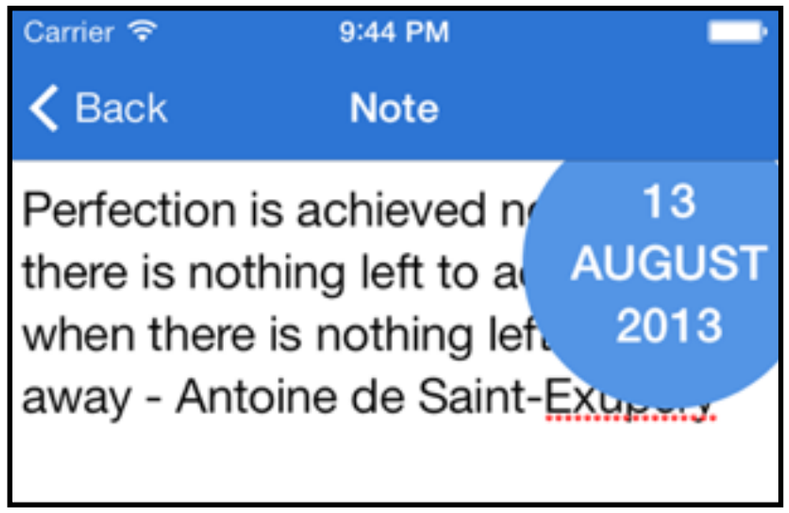
- Es existieren ebenfalls "System Fonts"
  - Meistens für Dinge wie z.B. Buttons

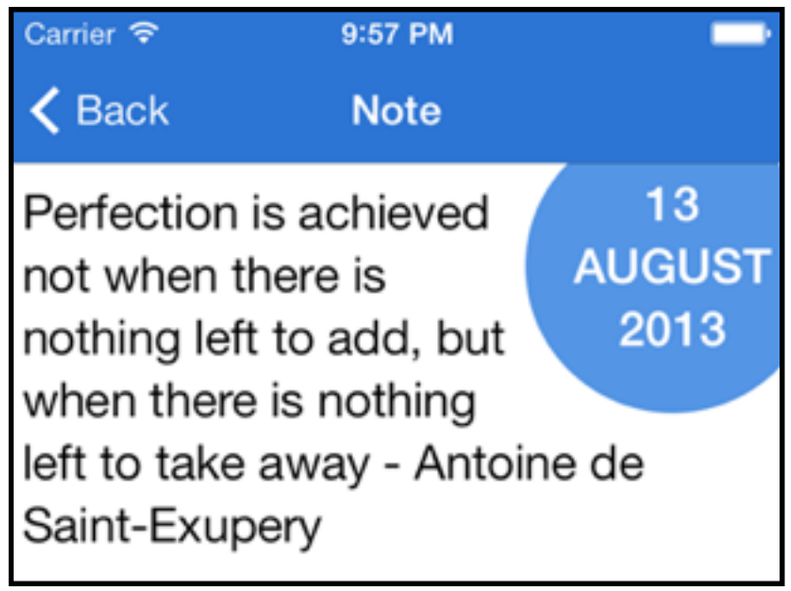
```
class fun systemFontSize(pointSize: CGFloat) -> UIFont
class fun boldsystemFontSize(pointSize: CGFloat) -> UIFont
```

  - Nicht verwenden für den User Content! Preferred Fonts dafür verwenden.

- Weitere Möglichkeiten Fonts zu erhalten
  - Auch wenn Sie es wahrscheinlich nicht häufig brauchen, UIFont und UIFontDescriptor Doku

Exclusion Paths





TextKit

Dynamic Text Formatting and Storage

- ~

  ~Some Text~ → ~Some Text~

- _

  Some Text → _Some Text_

- -

  -Some Text- → -~~Some Text~~-

- CAPS

  SOMETEXT → SOMETEXT

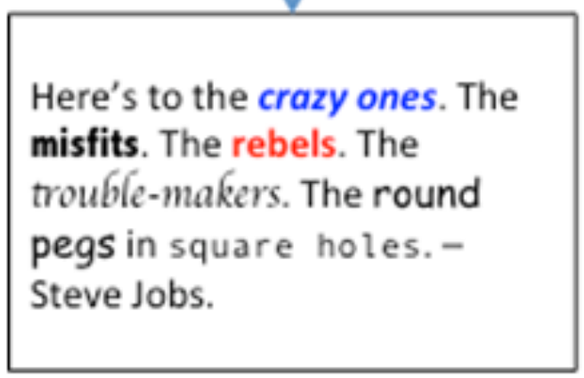
TextKit

NSLayoutManager

Takes the text from text storage and lays it out into the shape provided by the text container

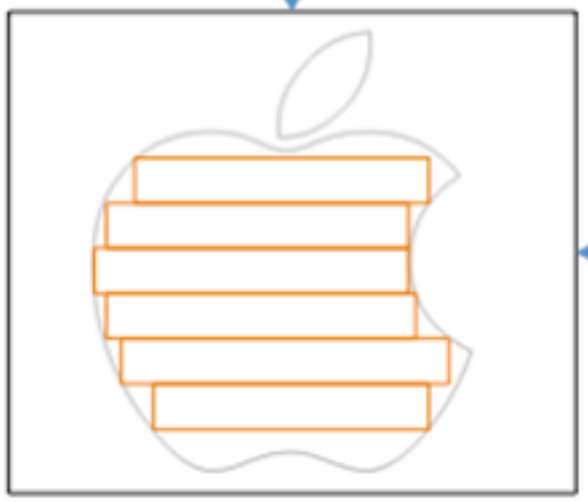
UITextView

Provides a text rendering context and edit functionality



NSTextStorage

Stores text and attributes



NSTextContainer

Describes a shape, and supplies this to the layout manager as line fragments

TextKit

Dynamic Text Format and Storage

Für `UITextView`, `UILabel` und `UITextField` werden die Klassen `NSTextStorage`, `NSLayoutManager` und `NSTextContainer` automatisch in deren Default-Implementierung verwendet.

Für die eigene Implementierung eine Subclass von `NSTextStorage` und `NSTextContainer` erstellen und entsprechend anpassen.

Ersetzen von Text mittels Regular Expression

```
replacements = [
    "(\\*\\w+(\\s\\w+)*\\*)" : boldAttributes,
    "(_\\w+(\\s\\w+)*_)"     : italicAttributes,
    "([0-9]+\\.)\\s"         : boldAttributes,
    "(-\\w+(\\s\\w+)*-)"     : strikeThroughAttributes,
    "(~\\w+(\\s\\w+)*~)"     : scriptAttributes,
    "\\s([A-Z]{2,})\\s"      : redTextAttributes
]
```

Images

Es existiert ein `UILabel`-Equivalent für Bilder

`UIImageView` (später mehr dazu)

Wir möchten aber vielleicht Bilder in `drawRect` zeichnen…

Erstellen einer `UIImage` Instanz

```
let image: UIImage? = UIImage(named: “foo”) // ist ein Optional
```

Hinzufügen von `foo.jpg` zum Projekt in das `Images.xcassets` File.

Bilder haben unterschiedliche Auflösung für unterschiedliche Devices (alles gemanaged in `Images.xcassets`)

Lässt sich auch aus einem File des Dateisystems erstellen

(Das Dateisystem schauen wir uns später noch genauer an…)

```
let image: UIImage? = UIImage(contentsOfFile: aString)
let image: UIImage? = UIImage(data: anNSData) // raw jpg, png,
                                              // tiff, etc.
```

Bild kann auch erstellt werden durch Zeichnen mit Core Graphics

Siehe Doku für `UIGraphicsBeginImageContext(CGSize)`

Images

Sobald wir ein UIImage haben können wir es on Screen bringen...

```
let image: UIImage = ...
image.drawAtPoint(aCGPoint)          // linke obere Ecke für
                                     // CGPoint
image.drawInRect(aCGRect)            // skaliert zu einem CGRect
image.drawAsPatternInRect(aCGRect) // tiles in einem CGRect
```

Redraw on Bounds Change

Wenn die Bounds eines UIViews sich ändern, wird per default nicht neu gezeichnet (redraw)

Stattdessen werden die “Bits” des existierenden Bildes skaliert und in die neuen Bounds gezeichnet.

Dies ist meist nicht das, was wir wollen…

Glücklicherweise existiert dafür ein `UIView` Property, welches wir auch in Xcode setzen können.

```
var contentMode: UIViewContentMode
```

UIViewContentMode

Nicht skalieren, nur irgendwo platzieren…

```
.Left/.Right/.Top/.Bottom/.TopRight/.TopLeft/.BottomRight/.BottomLeft/.Center
```

Skalieren der “Bits” des Views…

```
.ScaleToFill/.ScaleAspectFill/.ScaleAspectFit // .ScaleToFill ist default
```

Redraw durch aufrufen von drawRect

```
.Redraw
```