Erfahrungen mit Autolayout

UI Erstellung ohne Storyboard

Cocoaheads Saar

Warum?

warum will man die UI komplett mit Code erstellen

- Drei Arten der Erstellung von UI
  - Storyboard
  - XIBs
  - per Code
- Jede hat Vor- und Nachteile
- Individuelle Entscheidung und gewisse Weiterentwicklung als Softwareentwickler

warum Storyboards?

- Vorteile:
  - gute Übersicht über die Navigation bei kleinerer und mittlerer Anzahl an Views
  - Manipulieren der Navigation in der App ohne viel Code
- Nachteile:
  - schwer im Team zu bearbeiten
  - Konflikte in git bei Storyboards sind grausam
  - unübersichtlich bei vielen Views
  - langsam bei Start der App in Xcode

warum XIBs?

- Vorteile:
  - man kann die UI schnell zusammenbauen
  - einfache Implementierung für kleine Apps
  - verschiedene XIBs für verschiedene Lokalisierungen (unterschiedliche UI pro Land, nicht nur Textaustausch)
- Nachteile:
  - Konflikte beim Bearbeiten im Team schwieriger zu lösen
  - sehr dynamische Views kann man nicht mit XIBs bauen
  - schwerer zu Debuggen (falsche oder fehlende Verbindungen)

warum mit Code?

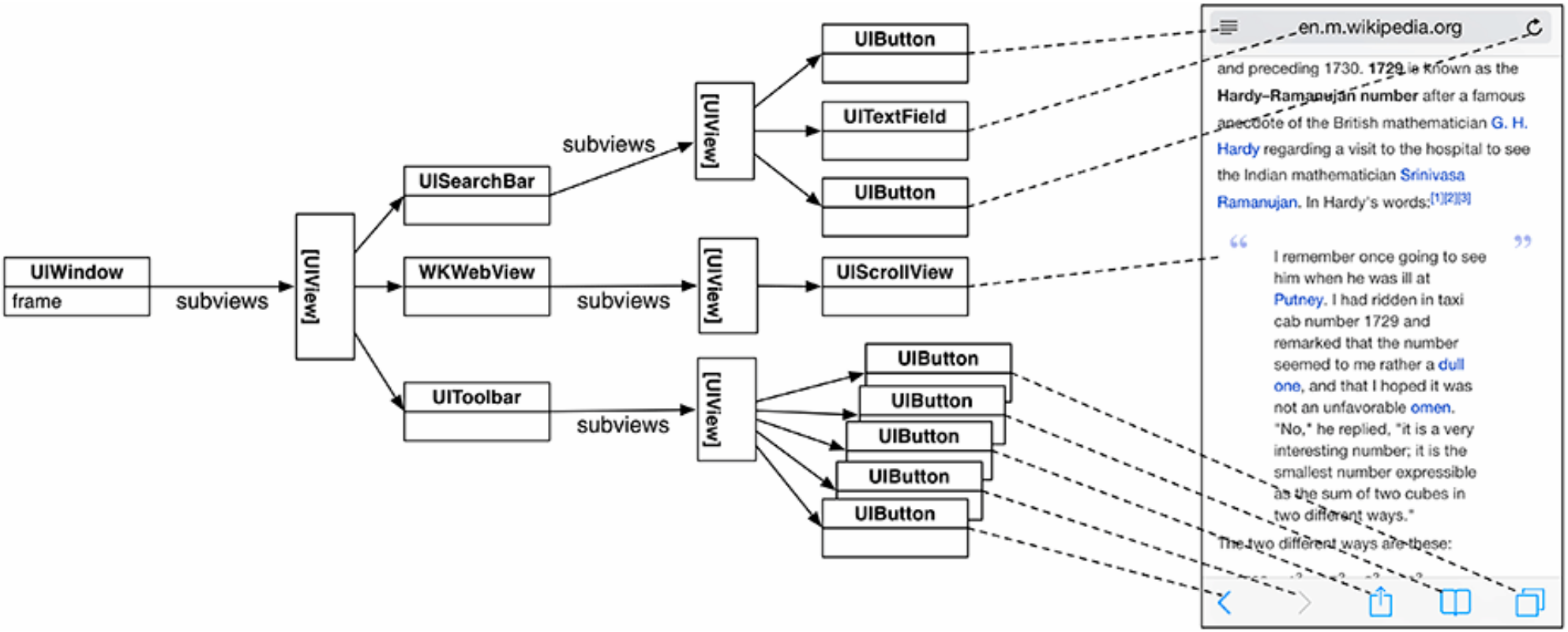
- Vorteile:
  - komplette Kontrolle über alles
  - Konflikte leichter zu beheben
- Nachteile:
  - keine Visualisierung der UI
  - Verständnis des UI Systems nötig (frame, bounds,..)
  - das Layout kostet mehr Zeit (viel Ausprobieren)
  - Einarbeitung für neue Teammitglieder schwerer

UI ohne Storyboard

- Die Views werden erzeugt und dann in die View Hierarchie eingefügt
- so genannte Constraints werden erzeugt und eingefügt
- Was bei Storyboards mit Klicken geht, macht man hier per Code

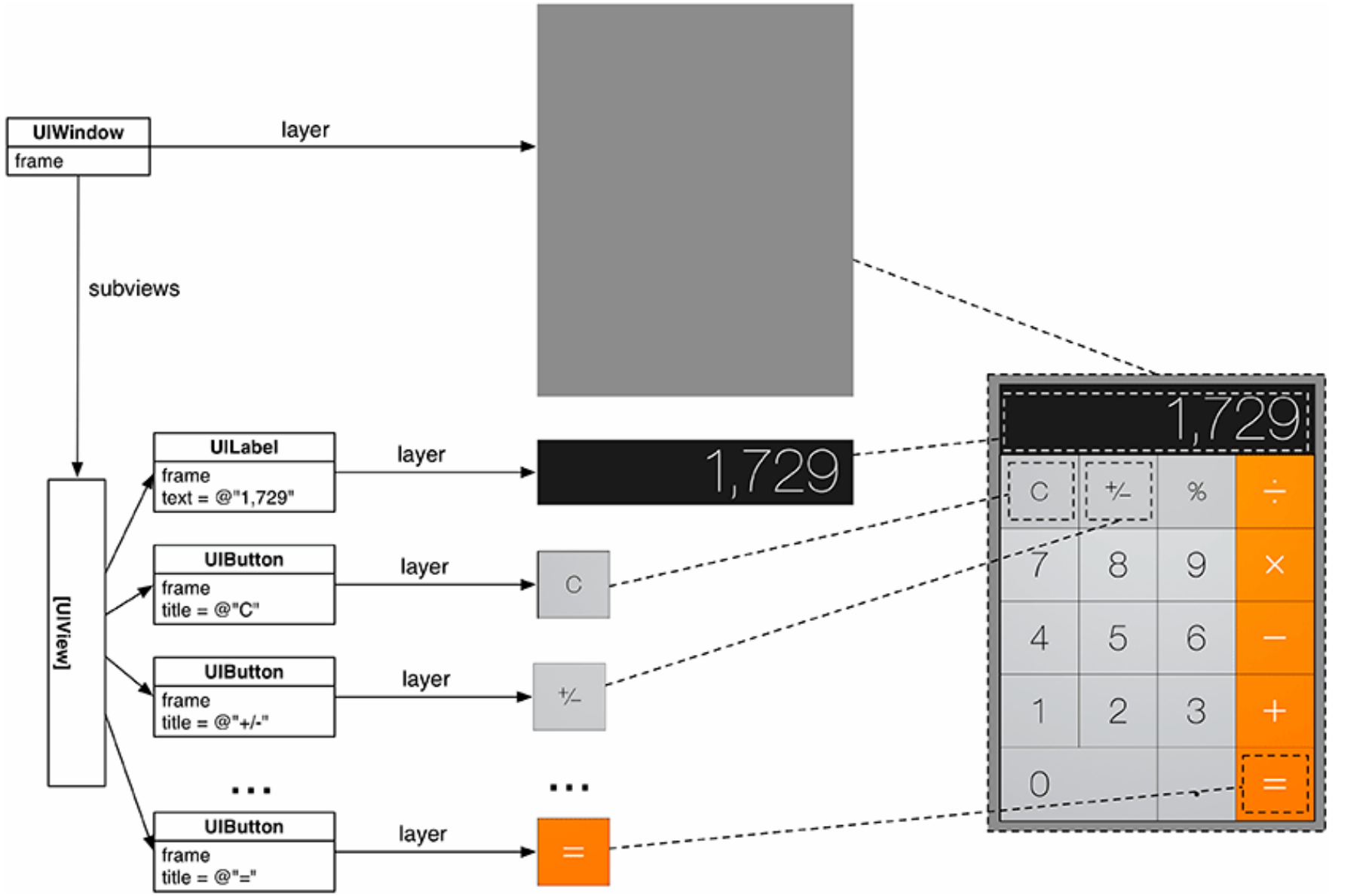
View Hierarchie bei iOS

- Jede App hat eine Instanz von UIWindow, die als Container für alle Views der App dient
- UIWindow ist eine Unterklasse von UIView, also selbst wieder eine View
- Das UIWindow wird erzeugt, sobald die App gestartet wird. Danach können beliebige Views als Subviews hinzugefügt werden
- Jede View kann selbst wieder Subviews enthalten. Es entsteht die so genannte View Hierarchie


UIWindow
frame
subviews
[UIView]
UISearchBar
WKWebView
UIToolbar
subviews
[UIView]
subviews
[UIView]
subviews
[UIView]
UIButton
UITextField
UIButton
UIScrollView
UIButton
UIButton
UIButton
UIButton
UIButton
en.m.wikipedia.org
and preceding 1730. 1729 is known as the Hardy–Ramanujan number after a famous anecdote of the British mathematician G. H. Hardy regarding a visit to the hospital to see the Indian mathematician Srinivasa Ramanujan. In Hardy's words:[1][2][3]
I remember once going to see him when he was ill at Putney. I had ridden in taxi cab number 1729 and remarked that the number seemed to me rather a dull one, and that I hoped it was not an unfavorable omen. "No," he replied, "it is a very interesting number; it is the smallest number expressible as the sum of two cubes in two different ways."
The two different ways are these:

Darstellung

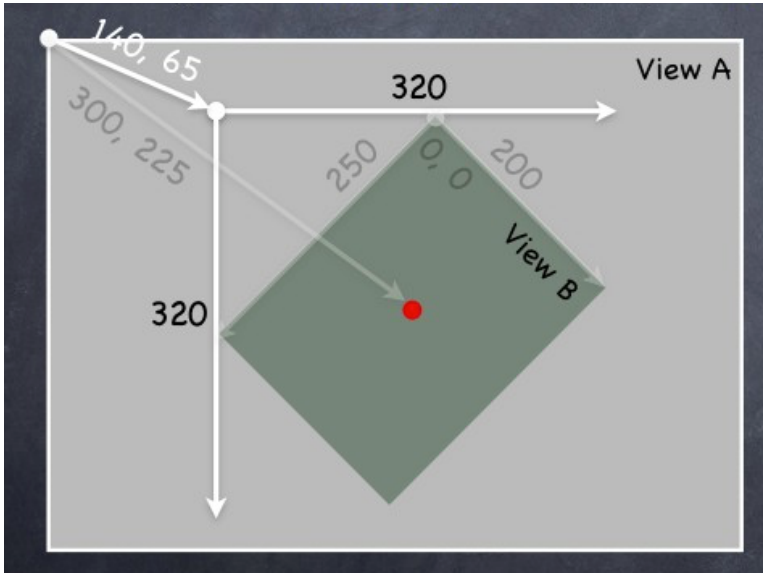
- Sobald die View Hierarchie erzeugt ist, wird sie auf den Screen gezeichnet. Dieser Prozess hat zwei Phasen:
  - jede View in der Hierarchie (inkl. Window) zeichnet sich selbst als Layer (Instanz von CALayer)
  - Die Layer aller Views werden übereinander gelegt und dargestellt.



Cocoaheads Saar

Frame und Bounds

- Bounds einer UIView ist das umfassende Rechteck als Ursprung (x,y) und Größe (width, height) relativ im eigenen Koordinatensystem
- Frame einer UIView ist das umfassende Rechteck als Ursprung (x,y) und Größe (width, height) relativ zur Superview, in der die UIView liegt
- immer Frame = Bounds? Natürlich nicht :)
- Hat man z.B. eine View Größe (100,100) an Position (25,25) im Superview, so sind:
- bounds.origin.x = bounds.origin.y = 0, bounds.size.width = bounds.size.height = 100
- frame.origin.x = frame.origin.y = 25, frame.size.width = frame.size.height = 100
- Also immer frame.size = bounds.size ? Natürlich ebenfalls nicht :)



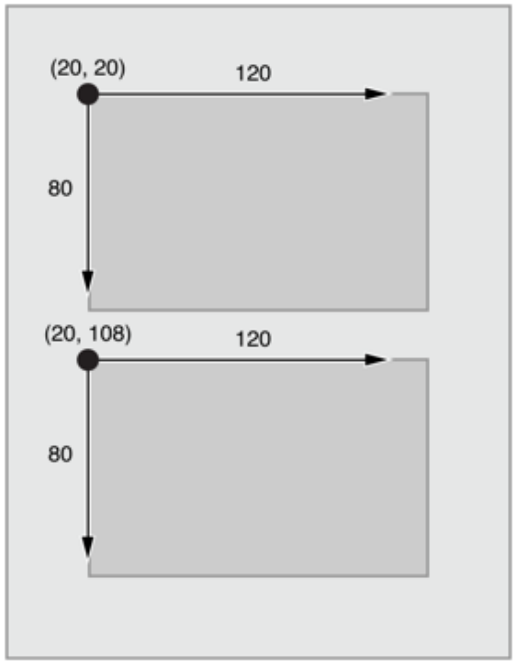
- superview (View A)
- subview (View B)
- View B:
  bounds = ((0,0),(200,250))
  frame = ((140,65),(320,320))

UI Erstellung

- Wie positioniert man dann die Views der Hierarchie?
- zwei Ansätze:
  - Frame based Layout
  - Auto Layout

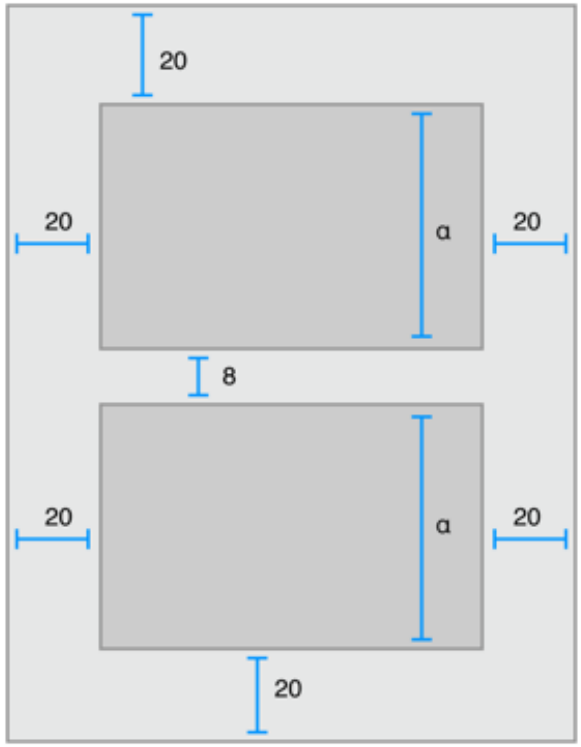
Frame-based Layout

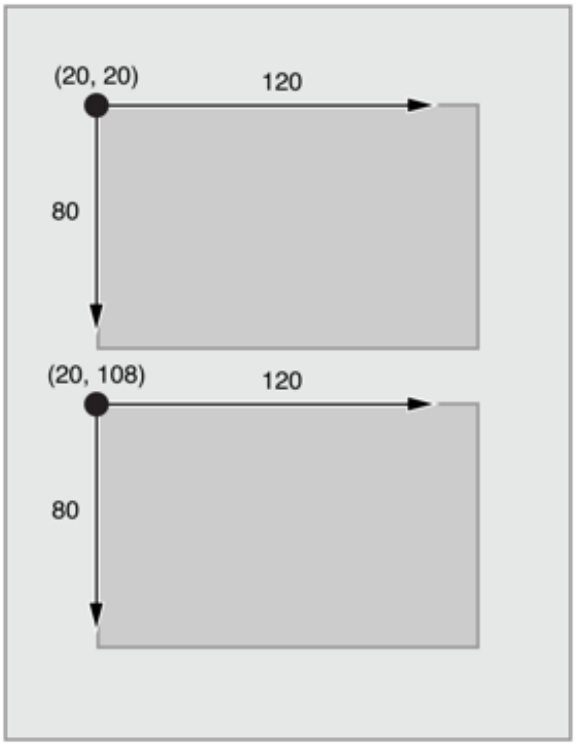
- UI durch Definition des Frames jeder View der Hierarchie definiert
- Man definiert einfach die Koordinaten und Größen des Frames in den Superviews
- Problem: unterschiedliche Device Größen bzw. Resizing
- Lösung: Man berechnet jede Größe und Position für jede View. Bei Änderung muss man neu berechnen
- Vorteil: sehr flexibel
- Nachteil: viel Aufwand
- Autoresizing Masks: Definiert die Änderung eines Frames bei Änderung des Superview Frames
  Bspl.: spinner.autoresizingMask = [.FlexibleRightMargin, .FlexibleLeftMargin, .FlexibleBottomMargin, .FlexibleTopMargin]



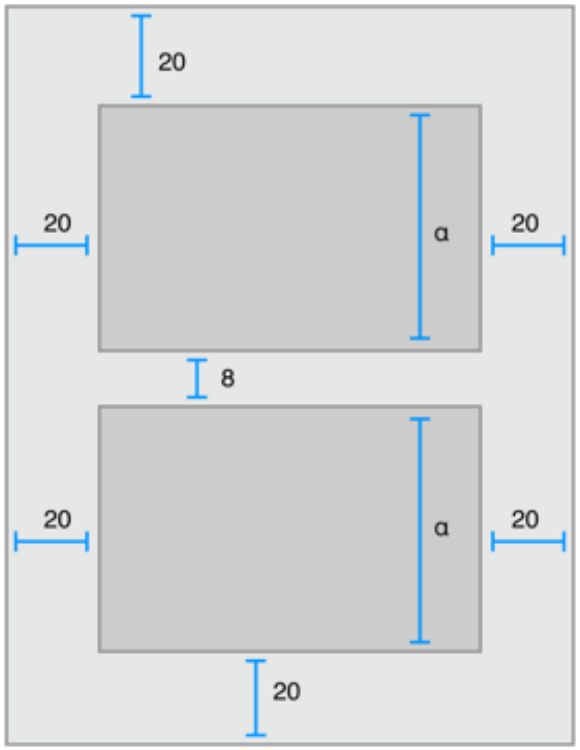
Auto Layout

- eingeführt mit Xcode 4.1 ab 10.7 Lion
- Xcode Dokumentation: Suche nach „Auto Layout Guide“
- Autolayout berechnet Größe und Position aller Views in der View Hierarchie aufgrund der für die Views definierten Constraints
- Dadurch kann die die UI dynamisch auf Änderungen reagieren
- Constraints definieren die Layout Beziehung zwischen Views





Frame based Layout

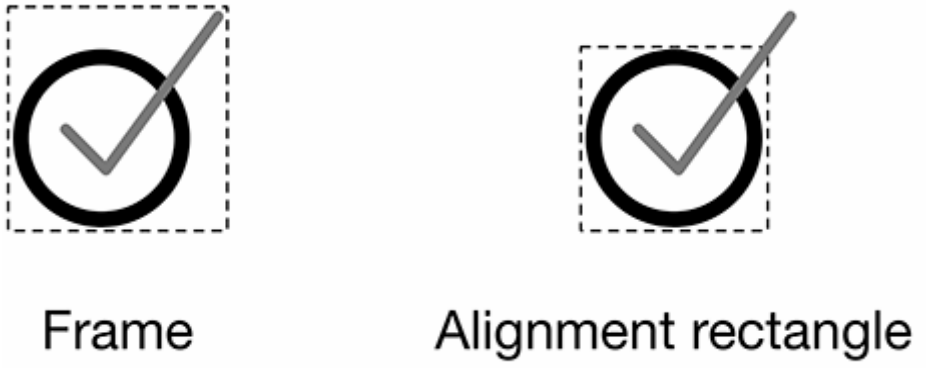


Layout per Constraints

Cocoaheads Saar

Alignment rectangle

- Auto Layout basiert auf Alignment Rectangles - umfassende Rechtecke
- Man muss genügend Informationen geben, damit die Position und Größe des Alignment Rectangle definiert ist. Damit ist der Frame definiert und damit kann Auto Layout die View positionieren. In der Regel braucht man dazu mindestens zwei Constraints, meist mehr
- In der Regel ist das Alignment Rectangle gleich dem Frame. Aber nicht immer



Constraints erstellen

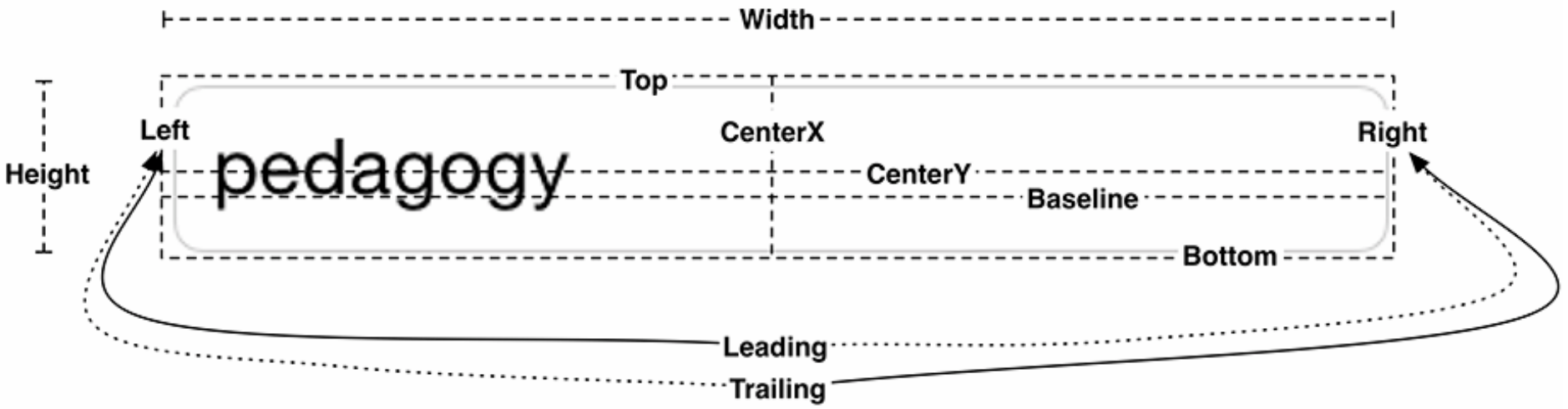
- Zwei Schritte:
  - Festlegung der Positionierung der Views zueinander (auf Papier oder mit Tools)
  - Übersetzen in die „Constraints Sprache“ und einzufügen in den Code

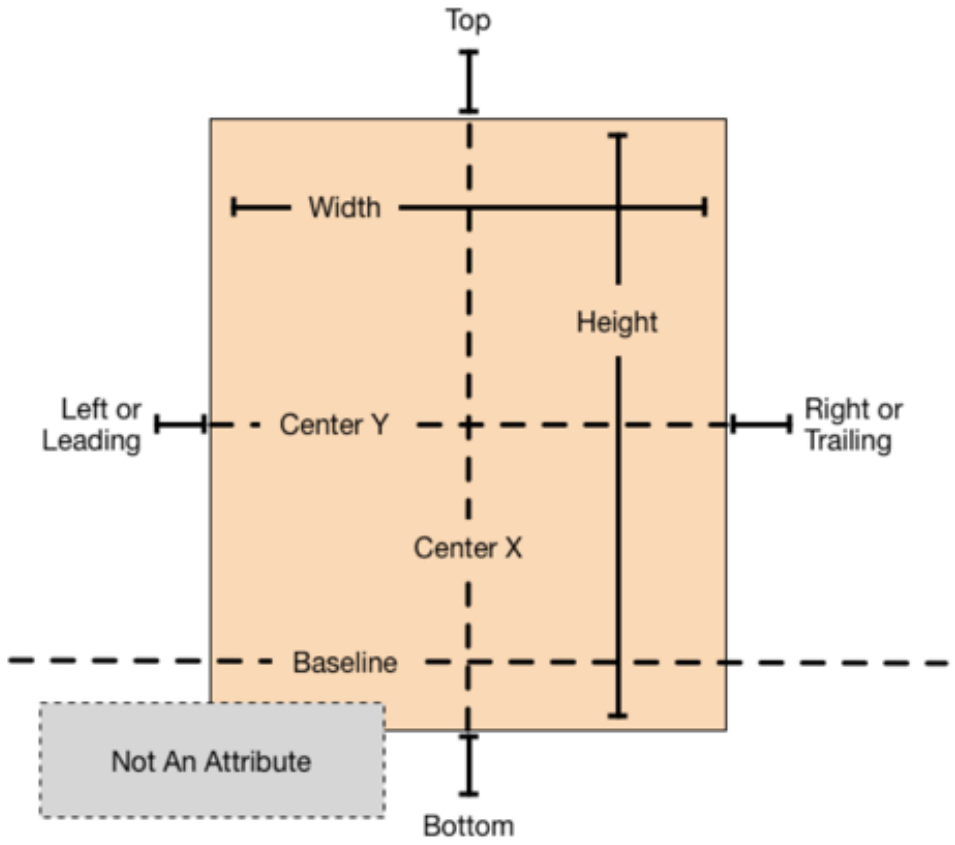
Constraints

- Constraints sind einfache lineare Gleichungen der Form :

  Item1.Attribut1 = Multiplier x Item2.Attribut2 + Constant

- Item1 und Item2 sind die Views
- Attribute in Enum NSLayoutAttribute definiert (nächste Folie)
- Multiplier und Constant sind CGFloats
- = kann auch >= / > / < / <= sein
  Blue.leading >= 1.0 * Red.trailing + 8.0
  (minimaler Abstand zwischen Blue und Red ist 8)
- NotAnAttribute bei keiner Abhängigkeit von anderer View
  View.height = 0.0 * nil.NotAnAttribute + 40.0
  (Höhe konstant 40)





when language is left to right

when language is right to left

Prioritäten

- Constraints können Prioritäten haben
- Per Default haben alle eine Priorität 1000. Diese Constraint MUSS erfüllt werden. Falls nicht, gibt es einen Fehler
- Priorität 1-999 sind optionale Constraints. Diese können erfüllt werden. Falls nicht, werden sie ignoriert.
- Beispiel:
  Constraint 1: Höhe eines bestimmten Buttons soll 50 sein
  Constraint 2: Höhe aller Buttons soll 40 sein

TopLayoutGuide und Co.

- Es fehlt nur noch die Möglichkeit, die Position und Größe der Views zu Rändern von Views oder Superviews zu definieren
- ToolBar, StatusBar, TabBar und NavigationBar reduzieren die nutzbare Größe und sind beim Drehen des Gerätes u.U. sichtbar oder unsichtbar
- Um das Problem zu lösen, erstellt der **UIViewController** automatisch zwei unsichtbare Views, TopLayoutGuide und BottomLayoutGuide, fügt sie in die View Hierarchie ein und updated automatisch die Größe
- die untere Seite des TopLayoutGuides ist automatisch gleich dem Bottom der untersten Bar oben, entsprechend die obere Seite des BottomLayoutGuides dem Top der obersten Bar unten
- Da dies Views sind, kann man relativ zu BottomLayoutGuide.top bzw. TopLayoutGuides.bottom Views positionieren, deren Position automatisch angepasst
- **Achtung**: das geht erst ab iOS 9. Es gibt immer noch bei jeder UIView ein LayoutMargins, ein UIEdgeInset, was die vier Werte top, left, bottom, right hat

Übersetzung in Constraints

- Nehmen wir ein sehr einfaches Beispiel:
- Ein UITextField (textField) mit einem UILabel (textLabel) in einer NSView (superView) irgendwo in der Hierarchie
- Überlegungen für Constraints:

    1. textLabel, links, horizontal zentriert zum textField, rechts
    2. Abstand 10 von textLabel und textField
    3. textLabel beginnt links am Rand, textField endet rechts am Rand
    4. textLabel hat wie textField Abstand 20 von oben
    5. textField soll sich in der Breite ändern, textLabel nicht

- textLabel.Baseline = 1.0 x textField.Baseline + 0.0 (1.)
- textField.left = 1.0 x textLabel.right + 10.0 (2.)
- textLabel.left = 1.0 x superView.LeftMargin + 0.0 (3.)
- textField.right = 1.0 x superView.RightMargin + 0.0 (3.)
- textLabel.top = 1.0 x TopLayoutGuide.bottom + 20.0 (4.)
- textField.top = 1.0 x TopLayoutGuide.bottom + 20.0 (4.)
- (5.) wird von Auto Layout erwartet

NSLayoutConstraints

- Constraints sind Instanzen von NSLayoutConstraints
- Sie beschreiben entweder einen absoluten Wert für Width oder Height oder eine Beziehung zwischen den Attributen zweier Views
- Die Attribute können verschieden sein, müssen aber nicht
- Es müssen nicht die Attribute von Subviews einer Superview sein. Sie müssen auch nicht Subview und zugehörige Superview sein. Sie müssen nur den gleichen Vorfahren haben, d.h. irgendwo in der Hierarchie muss es einen gemeinsame Superview haben (anchestor)

- Constraints werden genauso geschrieben, wie wir sie eben definiert haben

- textLabel.Baseline = 1.0 x textField.Baseline + 0.0 wird zu :

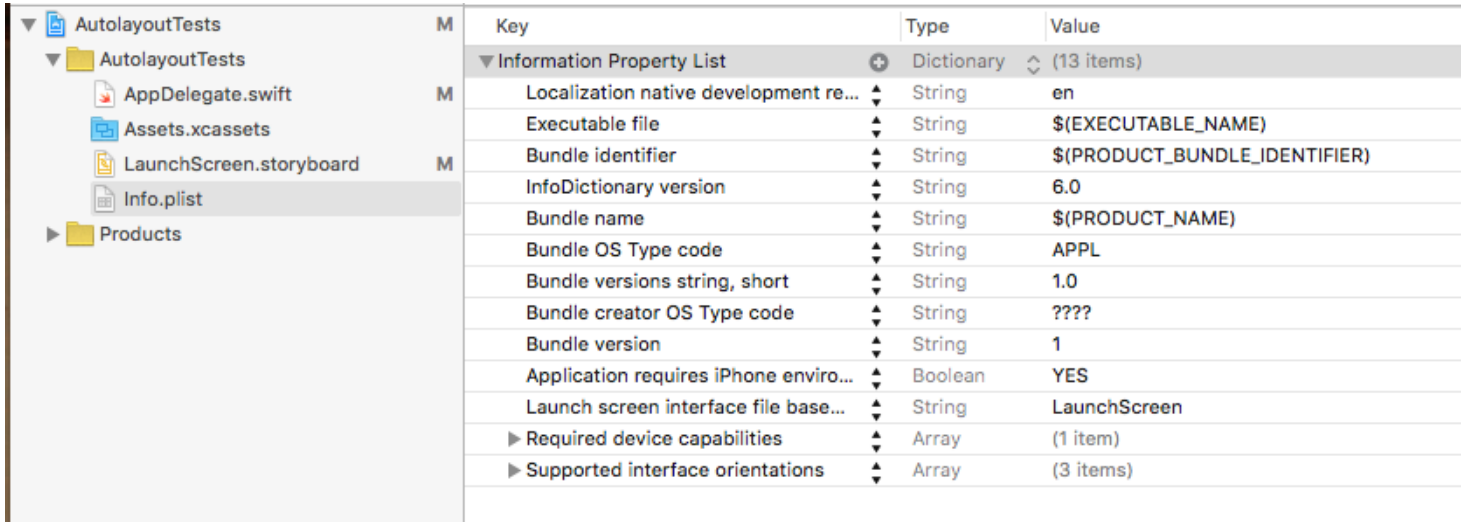
  superView.addConstraint(NSLayoutConstraint(item: textField, attribute: .Baseline, relatedBy: .Equal, toItem: textLabel, attribute: .Baseline, multiplier: 1, constant: 0))

Es geht los

wir bauen uns die UI zusammen

Vorarbeit

- Neues Projekt: iOS App, Single View Template
- Entfernen des Storyboards:
  - Info.plist: Entfernen des Eintrags „Main Storyboard file base name“
  - Löschen von Main.storyboard und Viewcontroller.swift
  - App startet trotzdem, natürlich mit weißem Screen



MVC

- M = Model, V = View, C = Controller
- Das M kennt nicht das V und umgekehrt
- Der C kennt das M und das V und „vermittelt“
- Austausch von M oder V recht einfach
- Konsequenz: M, V und C sind jeweils eigene (u.U. auch mehrere) Dateien (wegen Übersicht)

- Erstellen von MainView.swift von UIView abgeleitet : Das wird das V
- Erstellen von MainViewController.swift von UIViewController abgeleitet : Das wird das C
- Erstellen von Model.swift : Das wird das M (in dem Beispiel hier brauchen wir das nicht)

Verknüpfen

- AppDelegate muss jetzt nur dazu gebracht werden, die neuen Dateien zu verwenden
- Dazu erzeugt man eine neue Instanz von MainViewController und setzt diese als rootViewController des UIWindow

AppDelegate.swift

```
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow? = UIWindow(frame: UIScreen.mainScreen().bounds)

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[NSObject: AnyObject]?) -> Bool {

        let mainViewController = MainViewController()

        window?.rootViewController = mainViewController
        window?.makeKeyAndVisible()

        return true
    }

}
```

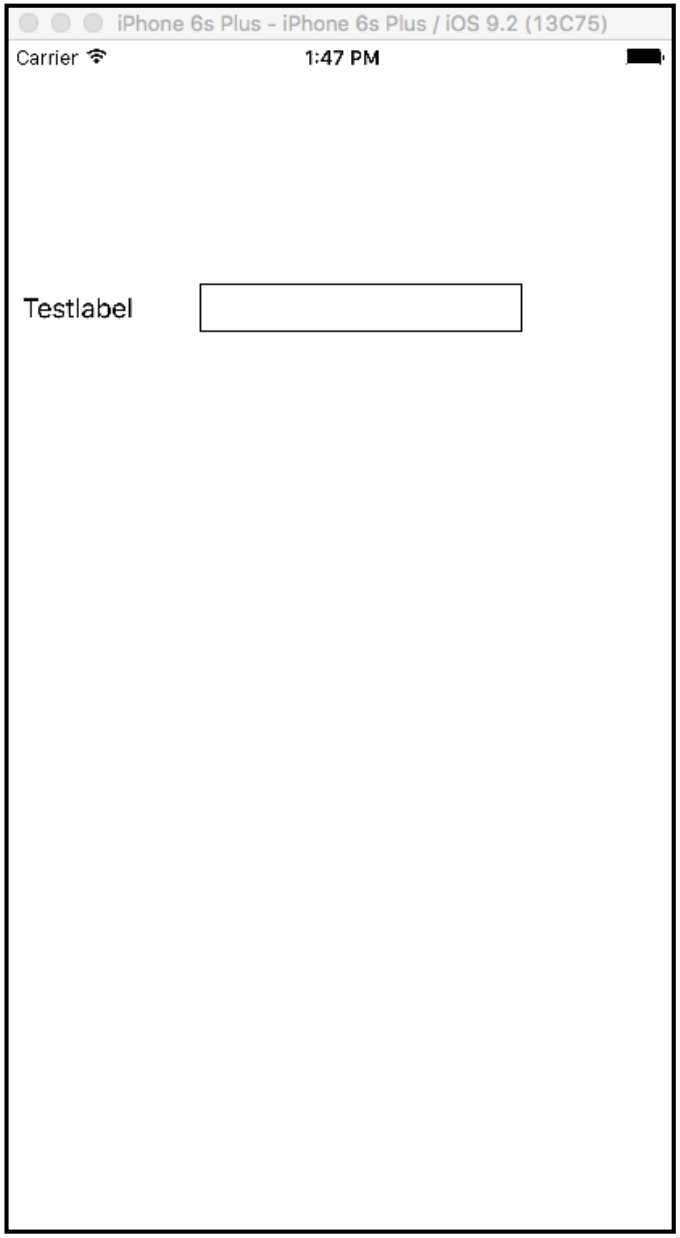
Cocoaheads Saar

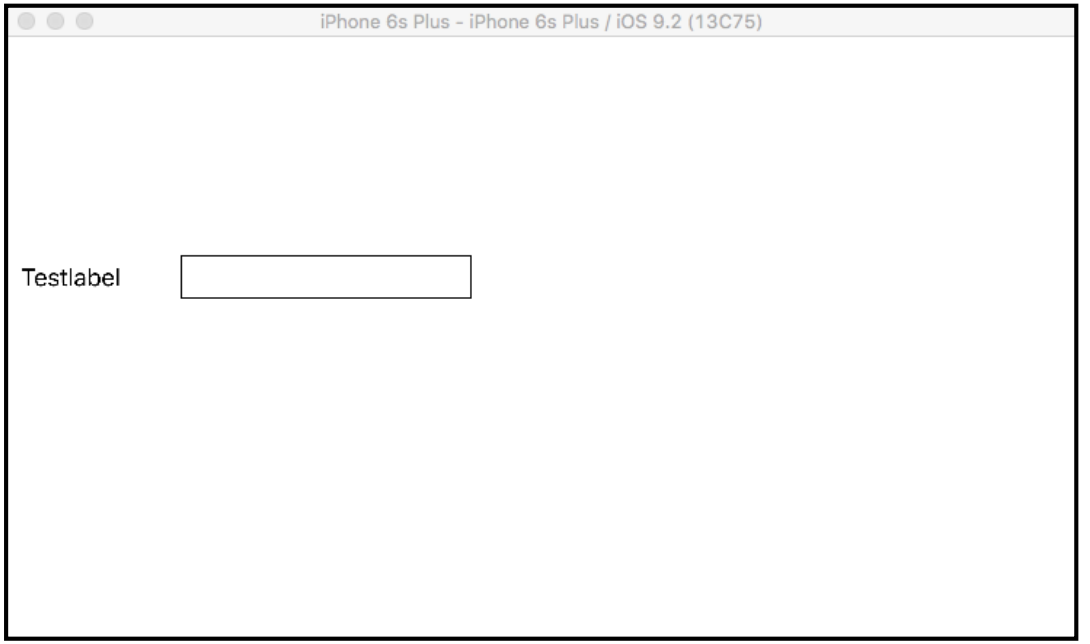
MainViewController.swift

```
class MainViewController: UIViewController {

    var mainView: MainView {
        return view as! MainView
    }

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func loadView() {

        let contentView = MainView(frame: .zero)
        view = contentView
    }

}
```


MainView.swift

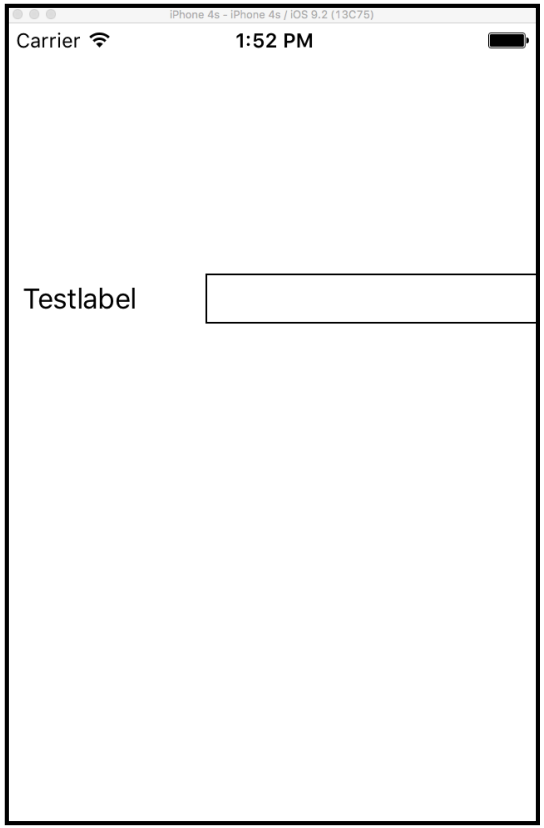
```
class MainView: UIView {

    var textLabel: UILabel
    var textField: UITextField

    override init(frame: CGRect) {

        // views
        let labelViewRect = CGRectMake(10, 150, 100, 30);
        let textViewRect = CGRectMake(120, 150, 200, 30);

        textLabel = UILabel(frame: labelViewRect)
        textLabel.translatesAutoresizingMaskIntoConstraints = false
        textLabel.text = "Testlabel"

        textField = UITextField(frame: textViewRect)
        textField.borderStyle = .Line
        textField.translatesAutoresizingMaskIntoConstraints = false

        super.init(frame: frame)

        backgroundColor = .whiteColor()

        addSubview(textLabel)
        addSubview(textField)
    }

    required init(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}
```



iPhone 6s Plus - iPhone 6s Plus / iOS 9.2 (13C75)
Carrier
1:47 PM
Testlabel


iPhone 6s Plus - iPhone 6s Plus / iOS 9.2 (13C75)
Testlabel

warum war das falsch?

```
class MainView: UIView {

    var textLabel: UILabel
    var textField: UITextField

    override init(frame: CGRect) {

        // views
        let labelViewRect = CGRectMake(10, 150, 100, 30);
        let textViewRect = CGRectMake(120, 150, 200, 30);

        textLabel = UILabel(frame: labelViewRect)
        textLabel.translatesAutoresizingMaskIntoConstraints = false
        textLabel.text = "Testlabel"

        textField = UITextField(frame: textViewRect)
        textField.borderStyle = .Line
        textField.translatesAutoresizingMaskIntoConstraints = false

        super.init(frame: frame)

        backgroundColor = .greenColor()

        addSubview(textLabel)
        addSubview(textField)
    }

    required init(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}
```



Cocoaheads Saar

Grundprinzipien

- Frames sind immer CGRectMake(0, 0, 0, 0) oder abgekürzt .zero
- NUR die Constraints definieren Größe und Position, damit Auto Layout bei unterschiedlichen Größen oder Rotation auch anpassen kann
- translatesAutoresizingMaskIntoConstraints wird für ALLE Views auf false gesetzt, weil es sonst zu Konflikten kommt.

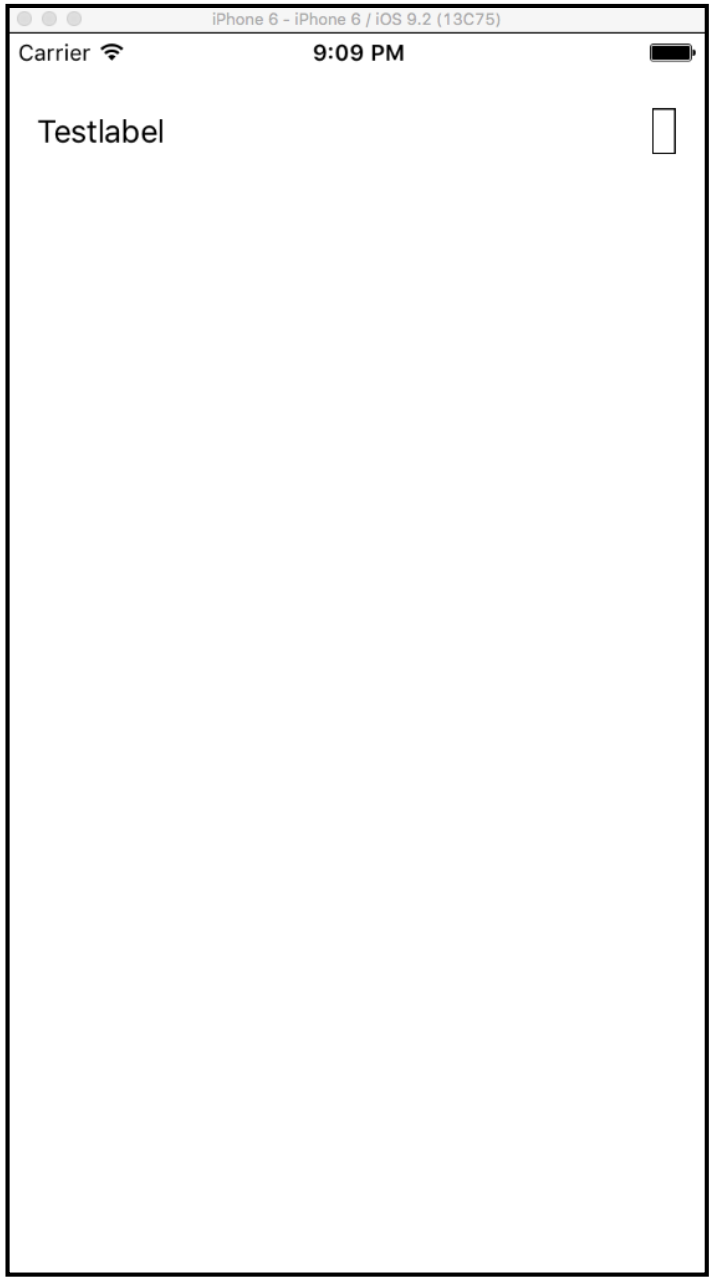
MainViewController

```
class MainViewController: UIViewController {

    var mainView: MainView {
        return view as! MainView
    }

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    override func loadView() {

        let contentView = MainView(frame: .zero)
        view = contentView
    }

    override func viewWillLayoutSubviews() {

        NSLayoutConstraint.activateConstraints([NSLayoutConstraint(item: mainView.textField,
attribute: .Top, relatedBy: .Equal, toItem: self.topLayoutGuide, attribute: .Bottom , multiplier:
1, constant: 20)])
        NSLayoutConstraint.activateConstraints([NSLayoutConstraint(item: mainView.textLabel,
attribute: .Top, relatedBy: .Equal, toItem: self.topLayoutGuide, attribute: .Bottom , multiplier:
1, constant: 20)])
    }

}
```

Cocoaheads Saar

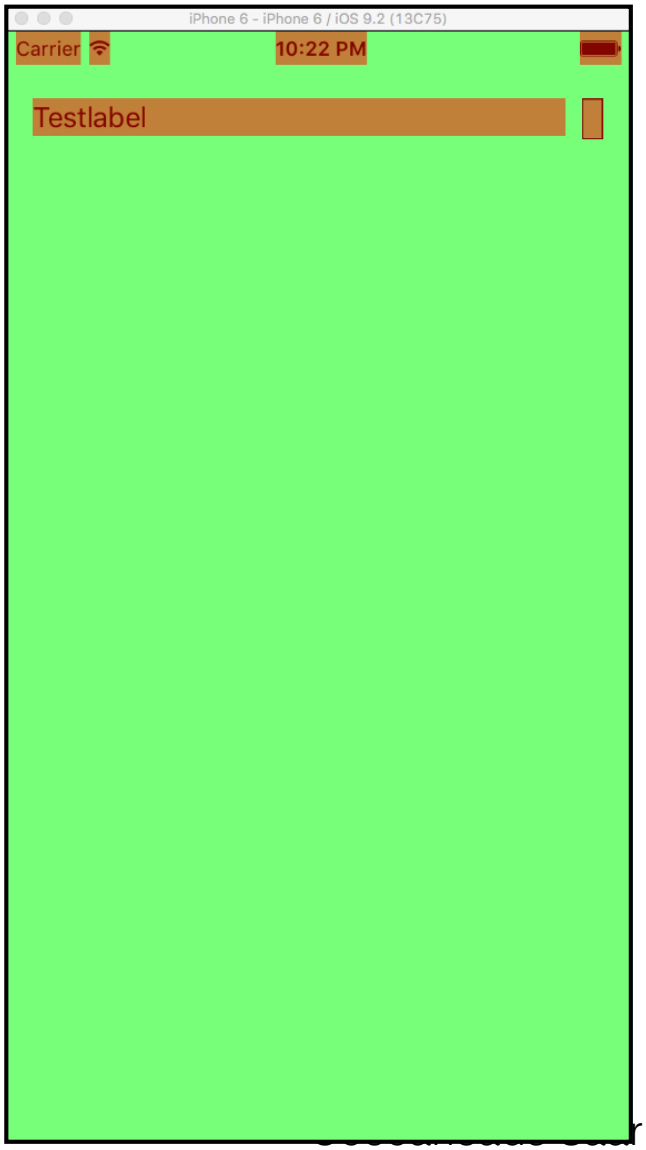
MainView

```
class MainView: UIView {

    var textLabel: UILabel
    var textField: UITextField

    override init(frame: CGRect) {

        // views
        textLabel = UILabel(frame: .zero)
        textLabel.translatesAutoresizingMaskIntoConstraints = false
        textLabel.text = "Testlabel"
        textField = UITextField(frame: .zero)
        textField.borderStyle = .Line
        textField.translatesAutoresizingMaskIntoConstraints = false

        super.init(frame: frame)

        backgroundColor = .whiteColor()

        addSubview(textLabel)
        addSubview(textField)

        self.addConstraint(NSLayoutConstraint(item: textField, attribute: .Baseline, relatedBy: .Equal, toItem: textLabel,
attribute: .Baseline, multiplier: 1.0, constant: 0))
        self.addConstraint(NSLayoutConstraint(item: textField, attribute: .Left, relatedBy: .Equal, toItem: textLabel,
attribute: .Right, multiplier: 1.0, constant: 10.0))
        self.addConstraint(NSLayoutConstraint(item: textLabel, attribute: .Left, relatedBy: .Equal, toItem: self,
attribute: .LeftMargin , multiplier: 1.0, constant: 0))
        self.addConstraint(NSLayoutConstraint(item: textField, attribute: .Right, relatedBy: .Equal, toItem: self,
attribute: .RightMargin , multiplier: 1.0, constant: 0))

   }

    required init(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }
}
```



iPhone 6 - iPhone 6 / iOS 9.2 (13C75)
Carrier
9:09 PM
Testlabel

Wie findet man Fehler?

- Man startet die App im Simulator
- Im Menü „Debug“ wählt man „Color Blended Layers“



- width von textLabel muss also noch gesetzt werden

- IntrinsicContentSize ist ein CGSize (width, height), was die so genannte natürliche Größe einer View beinhaltet. Bei UILabels ist das die Größe des Textes des Labels für den Font, den man gewählt hat

- **schlecht**: in MainView

```
self.addConstraint(NSLayoutConstraint(item: textLabel,
attribute: .Width, relatedBy: .Equal, toItem: nil,
attribute: .NotAnAttribute , multiplier: 1.0, constant: 80.0 ))
```

- **gut**: in MainView

```
let textSize = textLabel.intrinsicContentSize().width
self.addConstraint(NSLayoutConstraint(item: textLabel,
attribute: .Width, relatedBy: .Equal, toItem: nil,
attribute: .NotAnAttribute , multiplier: 1.0, constant: textSize ))
```


Constraints mit Anchors

- Die Klasse NSLayoutAnchor stellt eine API zur Verfügung, um Constraints einfacher und leserlicher zu erzeugen
- Um damit zu arbeiten, greift man einfach auf die Anchors der Klasse zu
- Für die Margins gibt es zusätzlich den layoutMarginGuide, der dann wiederum Anchors zur Verfügung stellt
- View haben ebenfalls Anchors, die entsprechend der Attribute
  angeboten werden (height => heightAnchor, …)

```
let margins = layoutMarginsGuide

var layoutConstraints = [NSLayoutConstraint]()

layoutConstraints.append(textLabel.centerYAnchor.constraintEqualToAnchor(textFi
eld.centerYAnchor))

layoutConstraints.append(textField.leftAnchor.constraintEqualToAnchor(textLabel
.rightAnchor, constant: 10.0))

layoutConstraints.append(textLabel.leftAnchor.constraintEqualToAnchor(margins.l
eftAnchor))

layoutConstraints.append(textField.rightAnchor.constraintEqualToAnchor(margins.
rightAnchor))

layoutConstraints.append(textLabel.widthAnchor.constraintEqualToConstant(textSi
ze))

NSLayoutConstraint.activateConstraints(layoutConstraints)
```


Visual Format

- Eine visuelle Art, die Positionen und Größen zu beschreiben.
- Vorteil: Relationen zwischen mehr als zwei Views beschreibbar und die Anzahl der Constraints ist sehr viel geringer
- Zuerst definiert man sich ein Array mit den Views, für die man Constraints beschreiben will
- Dann beschreibt man die Constraints in ASCII Art

```
        let views = ["textLabel" : textLabel, "textField" : textField]
        let metrics = ["textSize" : textSize]

        var layoutConstraints = [NSLayoutConstraint]()

        layoutConstraints +=
NSLayoutConstraint.constraintsWithVisualFormat("H:|-[textLabel(textSize)]-
[textField]-|", options: [], metrics: metrics, views: views)

        NSLayoutConstraint.activateConstraints(layoutConstraints)
```


- options ist eine Bitmask von NSLayoutFormatOptions, mit denen man z.B. Alignment definieren kann
- metrics definiert Abkürzungen für Werte, die man verwenden möchte
- views sind die Views, für die die Constraint gelten soll

Syntax

- [view1]-[view2] : Standardabstand 8
- [view(>=50)] : View mindestens 50 Points breit
- |-10-[view]-10-| : View 10 Points von den Margins
- V:[view1]-10-[view2] : View2 10 Points unter View1
- [view1(==view2)] : View1 genauso breit wie View2
- [view(>=10,<=100] : View zwischen 10 und 100 Points
- [view (100@20)] : View 100 Points, Priorität 20

MainView

```
    override init(frame: CGRect) {

        // views
        textLabel = UILabel(frame: .zero)
        textLabel.translatesAutoresizingMaskIntoConstraints = false
        textLabel.text = "Testlabel"
        let textSize = textLabel.intrinsicContentSize().width
        textField = UITextField(frame: .zero)
        textField.borderStyle = .Line
        textField.translatesAutoresizingMaskIntoConstraints = false

        super.init(frame: frame)

        backgroundColor = .whiteColor()

        addSubview(textLabel)
        addSubview(textField)

        let views = ["textLabel" : textLabel, "textField" : textField]
        let metrics = ["textSize" : textSize]
        var layoutConstraints = [NSLayoutConstraint]()
        layoutConstraints += NSLayoutConstraint.constraintsWithVisualFormat("H:|-
[textLabel(textSize)]-[textField]-|", options: [], metrics: metrics, views: views)
        NSLayoutConstraint.activateConstraints(layoutConstraints)
    }
```


MainViewController

```
override func viewWillLayoutSubviews() {

        let topLayoutGuide = self.topLayoutGuide

        let views = ["topLayoutGuide": topLayoutGuide, "textLabel":
mainVisualView.textLabel, "textField": mainVisualView.textField] as
[String:AnyObject]

NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisual
Format("V:[topLayoutGuide]-[textLabel]", options: [], metrics: nil, views:
views))

NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisual
Format("V:[topLayoutGuide]-[textField]", options: [], metrics: nil, views:
views))

    }
```

Cocoaheads Saar

Stackviews

- StackViews sind Container für Views, für die man Constraints definieren kann.
- Beispiel: textField und textLabel sollen in einen Stackview. Nötige Constraints für das obige sind dann:

1. Stackview geht von rechten bis linken Margin
2. Stackview hat Abstand 20 von oben

- Im Stackview sollen dann textField und textLabel horizontal ausgerichtet sein, textLabel hat Breite textSize und den Rest macht Auto Layout
- besonders wichtig bei mehr als zwei Views im Stackview
- ein Stackview kann auch aus anderen Stackviews bestehen
- **Achtung**: geht erst ab iOS 9

MainStackView

```
    var textLabel: UILabel
    var textField: UITextField
    var textStackView: UIStackView

    override init(frame: CGRect) {
       // views
        textLabel = UILabel(frame: .zero)
        textLabel.translatesAutoresizingMaskIntoConstraints = false
        textLabel.text = "Testlabel"
        let textSize = textLabel.intrinsicContentSize().width
        textField = UITextField(frame: .zero)
        textField.borderStyle = .Line
        textField.translatesAutoresizingMaskIntoConstraints = false
        textStackView = UIStackView(arrangedSubviews: [textLabel, textField])
        textStackView.translatesAutoresizingMaskIntoConstraints = false
        textStackView.axis = .Horizontal
        textStackView.spacing = 10
        textStackView.distribution = .Fill

        super.init(frame: frame)
        backgroundColor = .whiteColor()
        addSubview(textStackView)

        let views = ["textStackView" : textStackView]
        var layoutConstraints = [NSLayoutConstraint]()
        layoutConstraints += NSLayoutConstraint.constraintsWithVisualFormat("H:|-[textStackView]-|",
options: [], metrics: nil, views: views)
        layoutConstraints.append(textLabel.widthAnchor.constraintEqualToConstant(textSize))
        NSLayoutConstraint.activateConstraints(layoutConstraints)
   }
```

Cocoaheads Saar

MainStackViewController

```
    override func viewWillLayoutSubviews() {

        let topLayoutGuide = self.topLayoutGuide

        let views = ["topLayoutGuide": topLayoutGuide, "textStackView":
mainStackView.textStackView] as [String:AnyObject]

NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisual
Format("V:[topLayoutGuide]-[textStackView]", options: [], metrics: nil, views:
views))

    }
```

Cocoaheads Saar

MainStackButtonView

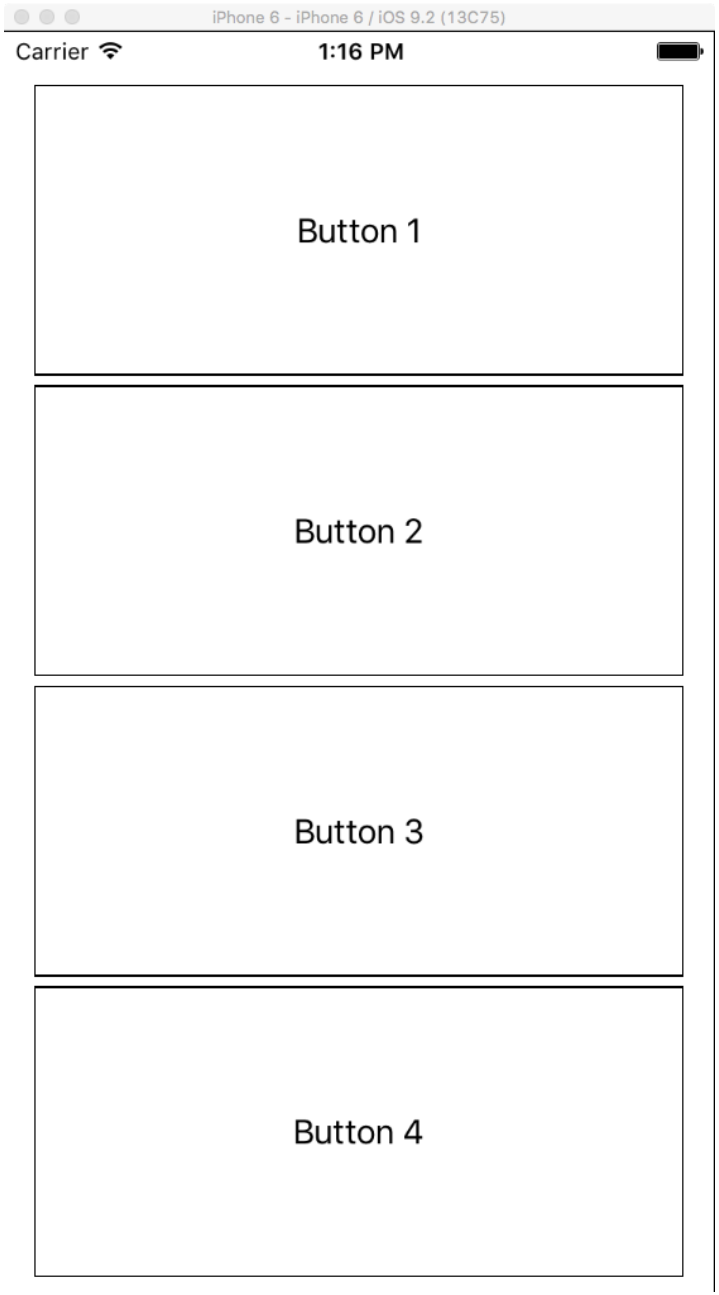
```
    var buttonStackView: UIStackView
    var buttonArray: [UIButton]
    var createNewButton: UIButton?

    override init(frame: CGRect) {
       buttonArray = []
        for buttonID in 1...4 {
            createNewButton = UIButton(frame: .zero)
            createNewButton!.setTitle( "Button \(buttonID)", forState: .Normal)
            createNewButton?.setTitleColor(UIColor.blackColor(), forState: .Normal)
            createNewButton!.translatesAutoresizingMaskIntoConstraints = false
            createNewButton?.layer.borderWidth = 1.0
            createNewButton!.tag = buttonID
            buttonArray.append(createNewButton!)
        }
        buttonStackView = UIStackView(arrangedSubviews: buttonArray)
        buttonStackView.translatesAutoresizingMaskIntoConstraints = false
        buttonStackView.axis = .Vertical
        buttonStackView.spacing = 5
        buttonStackView.distribution = .FillEqually

        super.init(frame: frame)
        addSubview(buttonStackView)

        let views = ["buttonStackView" : buttonStackView]
        var layoutConstraints = [NSLayoutConstraint]()
       layoutConstraints += NSLayoutConstraint.constraintsWithVisualFormat("H:|-[buttonStackView]-|",
options: [], metrics: nil, views: views)
        NSLayoutConstraint.activateConstraints(layoutConstraints)

    }
```

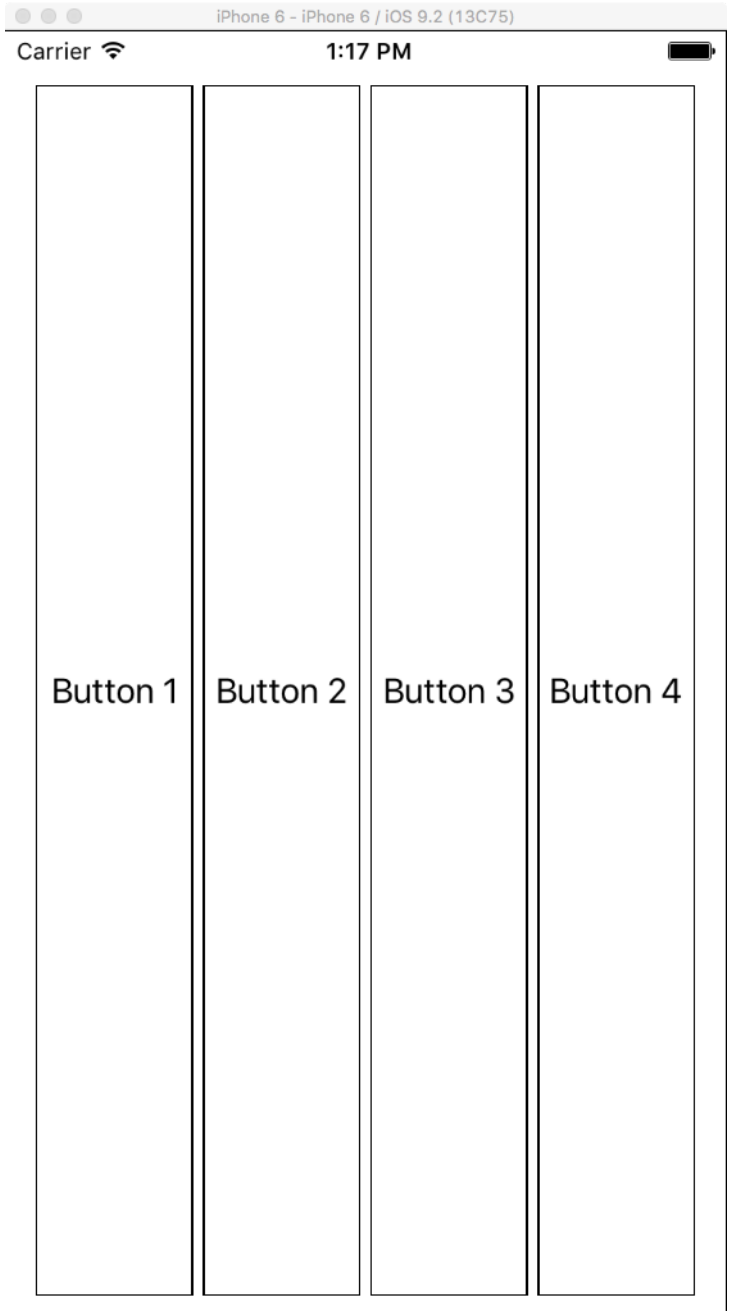
Cocoaheads Saar

MainStackButton ViewController

```
    override func viewDidLoad() {
        super.viewDidLoad()
        for newButton in mainStackView.buttonArray {
            newButton.addTarget(self, action: Selector("evaluateButton:"),
forControlEvents: .TouchDown)
        }
   }

    override func loadView() {
       let contentView = MainStackButtonsView(frame: .zero)
       view = contentView
    }

    override func viewWillLayoutSubviews() {
        let topLayoutGuide = self.topLayoutGuide
        let bottomLayoutGuide = self.bottomLayoutGuide
        let views = ["topLayoutGuide": topLayoutGuide, "buttonStackView" :
mainStackView.buttonStackView, "bottomLayoutGuide" : bottomLayoutGuide] as [String:AnyObject]

NSLayoutConstraint.activateConstraints(NSLayoutConstraint.constraintsWithVisualFormat("V:
[topLayoutGuide]-[buttonStackView]-[bottomLayoutGuide]", options: [], metrics: nil, views:
views))
   }

    func evaluateButton(sender: UIButton) {
        print("Button : \(sender.tag)")
    }
```

Cocoaheads Saar





Cocoaheads Saar

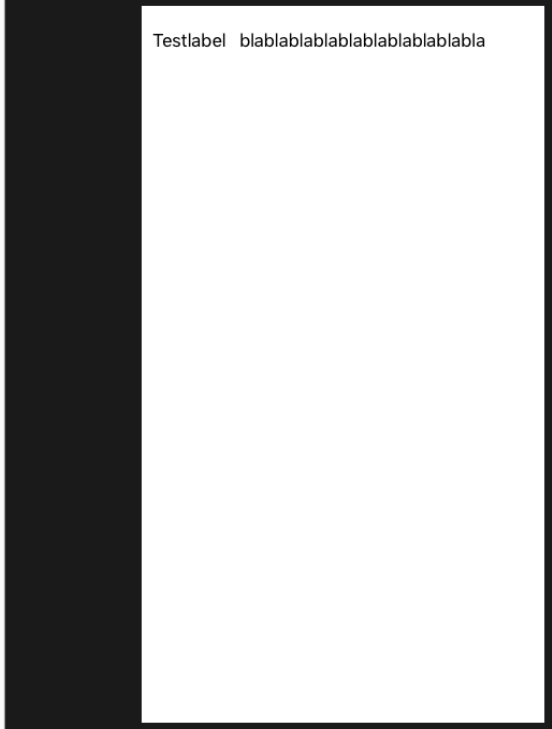
Layouten per Playground

- Für Swift kann man im Playground Layouten
- Es ist zwar nicht der fertige Code, den man dann im Projekt verwendet, aber zum Austesten, ob die Constraints passen, reicht es vollkommen aus
- Da man kein Device hat, muss man sich als erstes eine UIView erzeugen, die der Größe eines Devices entspricht (http://www.paintcodeapp.com/news/ultimate-guide-to-iphone-resolutions)
- Wichtig: das ist keine Simulation eines Devices, es fehlt also z.B. die StatusBar
- Natürlich kann man auch mit den Klassen arbeiten, die wir vorher definiert haben
- man muss den AssistentEditor aktivieren, um das Resultat zu sehen

AutolayoutTests 〉 AutolayoutTests 〉 SwiftUITests.playground

```swift
//: Playground - noun: a place where people can play

import UIKit
import XCPlayground

let mainFrame = UIView(frame: CGRect(x: 0, y: 0, width: 375, height: 667)) // iPhone 6

mainFrame.backgroundColor = UIColor.whiteColor()
XCPlaygroundPage.currentPage.liveView = mainFrame

var textLabel: UILabel
var textField: UITextField

textLabel = UILabel(frame: .zero)
textLabel.translatesAutoresizingMaskIntoConstraints = false
textLabel.text = "Testlabel"
let textSize = textLabel.intrinsicContentSize().width

textField = UITextField(frame: .zero)
textField.borderStyle = .Line
textField.translatesAutoresizingMaskIntoConstraints = false
textField.text = "blablablablablablablablabla"

mainFrame.addSubview(textField)
mainFrame.addSubview(textLabel)

let margins = mainFrame.layoutMarginsGuide

var layoutConstraints = [NSLayoutConstraint]()
layoutConstraints.append(textLabel.centerYAnchor.constraintEqualToAnchor(textField.centerYAnchor))
layoutConstraints.append(textField.leftAnchor.constraintEqualToAnchor(textLabel.rightAnchor, constant: 10.0))
layoutConstraints.append(textLabel.leftAnchor.constraintEqualToAnchor(mainFrame.leftAnchor, constant: 10.0))
layoutConstraints.append(textField.rightAnchor.constraintEqualToAnchor(mainFrame.rightAnchor, constant: 10.0))
layoutConstraints.append(textLabel.widthAnchor.constraintEqualToConstant(textSize))
layoutConstraints.append(textLabel.topAnchor.constraintEqualToAnchor(mainFrame.topAnchor, constant: 20))
layoutConstraints.append(textField.topAnchor.constraintEqualToAnchor(mainFrame.topAnchor, constant: 20))

NSLayoutConstraint.activateConstraints(layoutConstraints)
```

Timeline 〉 SwiftUITests.playground (Timeline)



Cocoaheads Saar

Was ist möglich?

- Komplexe Apps komplett im Code
- Problem der Organisation des Codes dabei
- Grundlagen sind geschaffen. Der Rest ist „Wie erzeuge ich diese oder jene View mit Code und was muss ich da setzen?“

Noch viel lernen du musst :)

–Yoda