Cocoa Touch und Objective C

DesignPatterns: http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaDesignPatterns/CocoaDesignPatterns.html#//apple_ref/doc/uid/TP40002974-CH6-SW6

Benutzer: ASS

Passwort: ass

Cocoa Touch

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

- Frameworkverbund

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

- Frameworkverbund
  - UIKit

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

- Frameworkverbund
  - UIKit
  - Foundation

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

- Frameworkverbund
  - UIKit
  - Foundation
  - CoreData

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

- Frameworkverbund
  - UIKit
  - Foundation
  - CoreData
  - MessageUI

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

Cocoa Touch

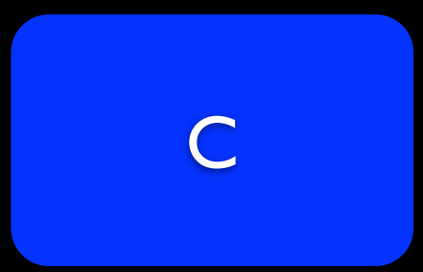
- Frameworkverbund
  - UIKit
  - Foundation
  - CoreData
  - MessageUI
  - ... (über 30 weitere)

CocoaTouch ist ein **Frameworkverbund**, bestehend aus über 30 Einzelframeworks. Zwei davon, das UIKit und die Foundation werden wir innerhalb dieses Kurses ein wenig kennen lernen...

ObjC

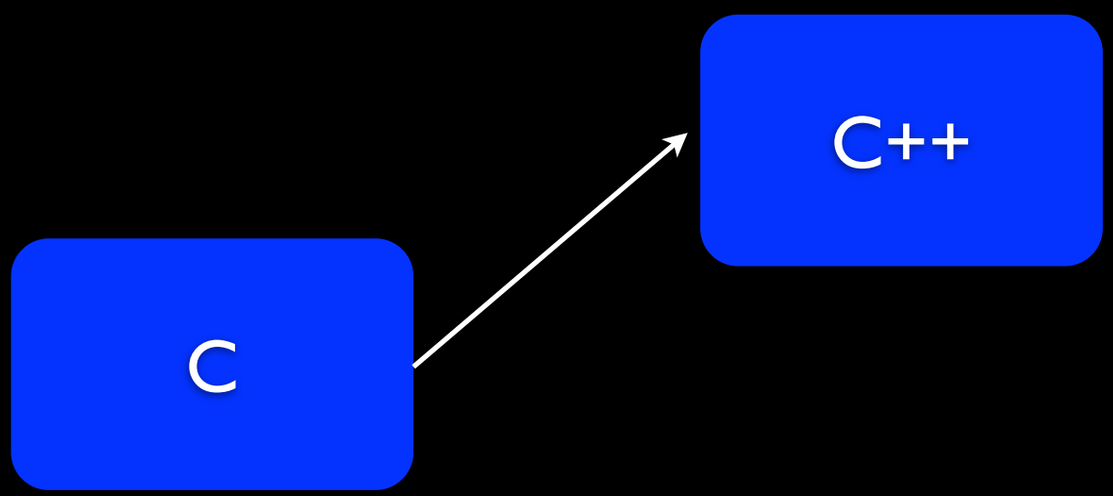
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm
Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



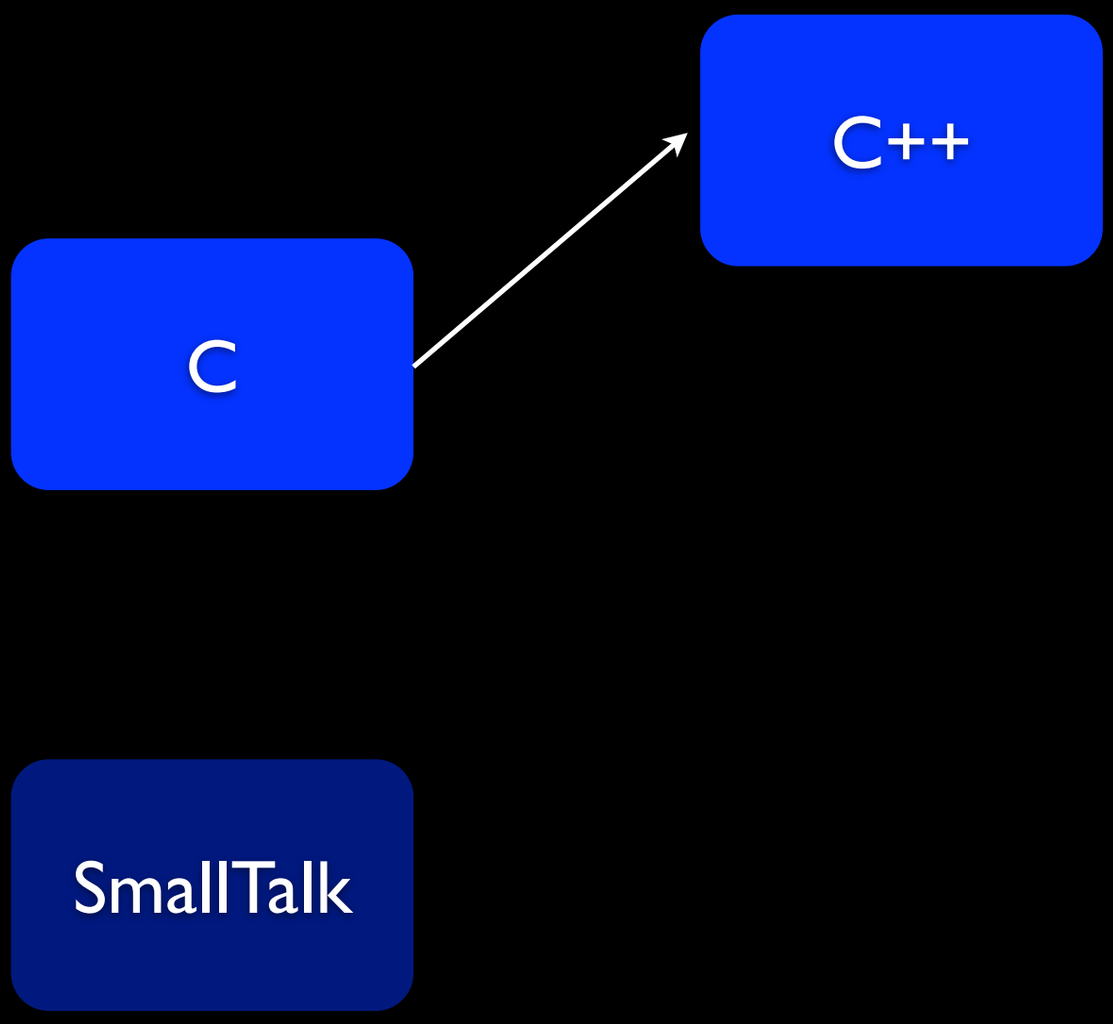
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



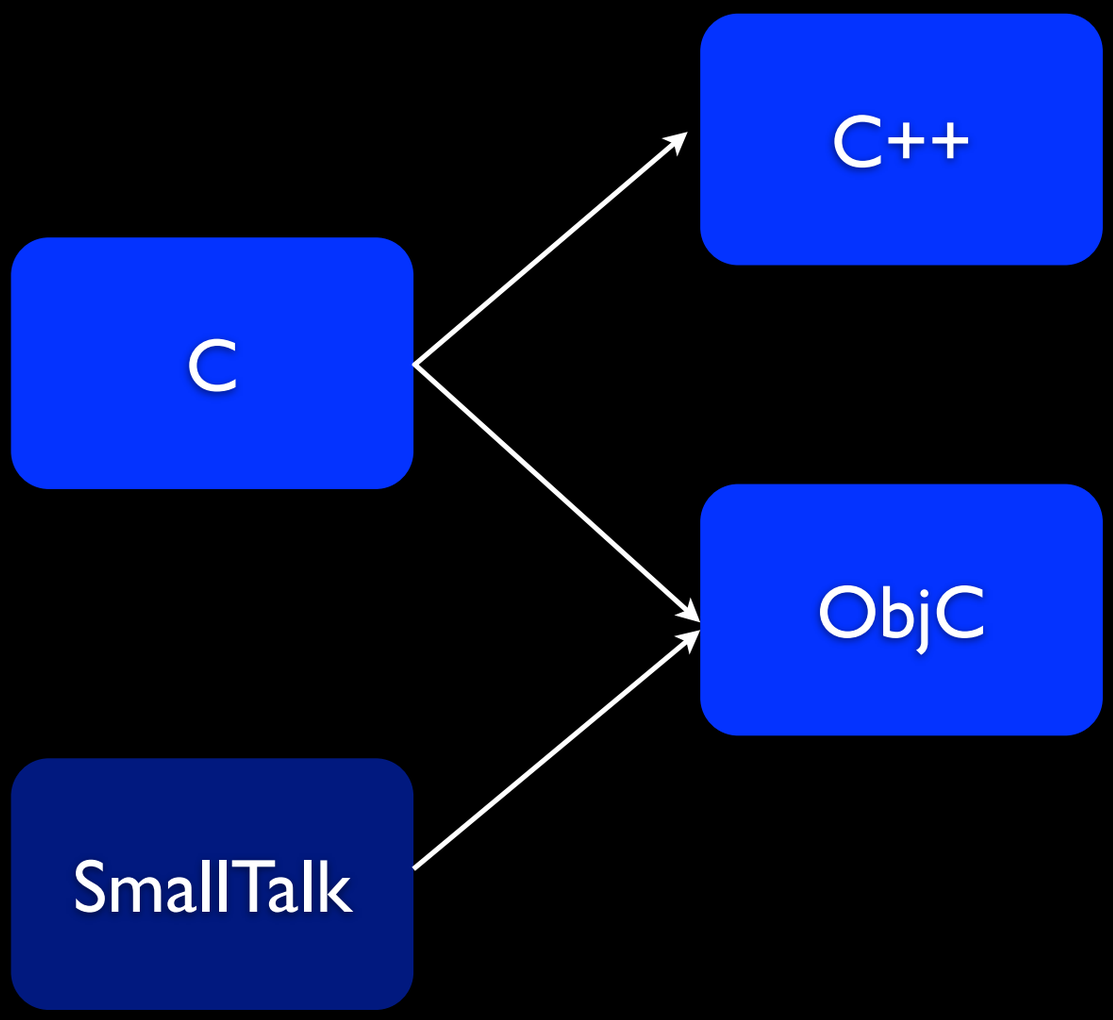
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



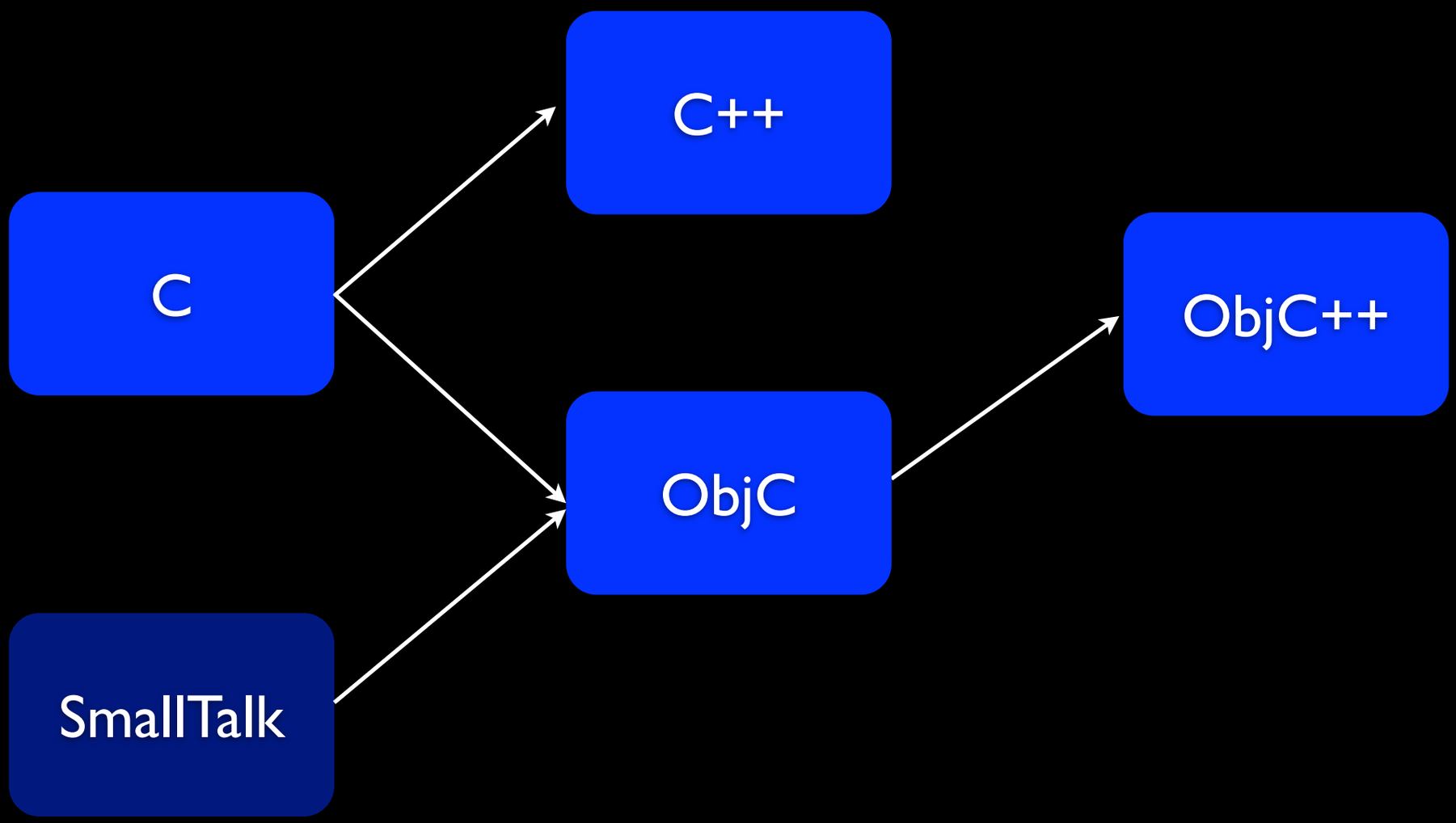
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



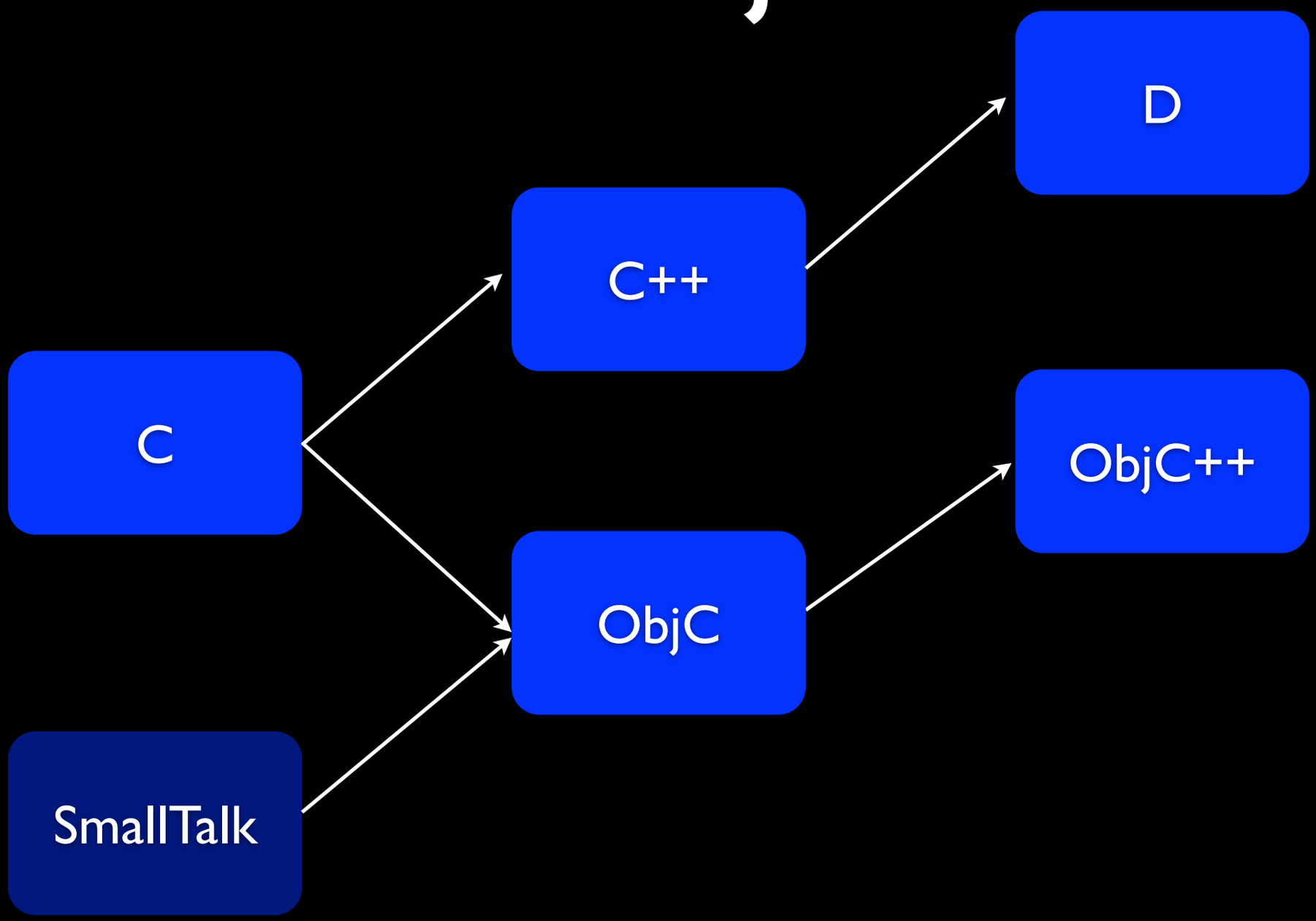
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



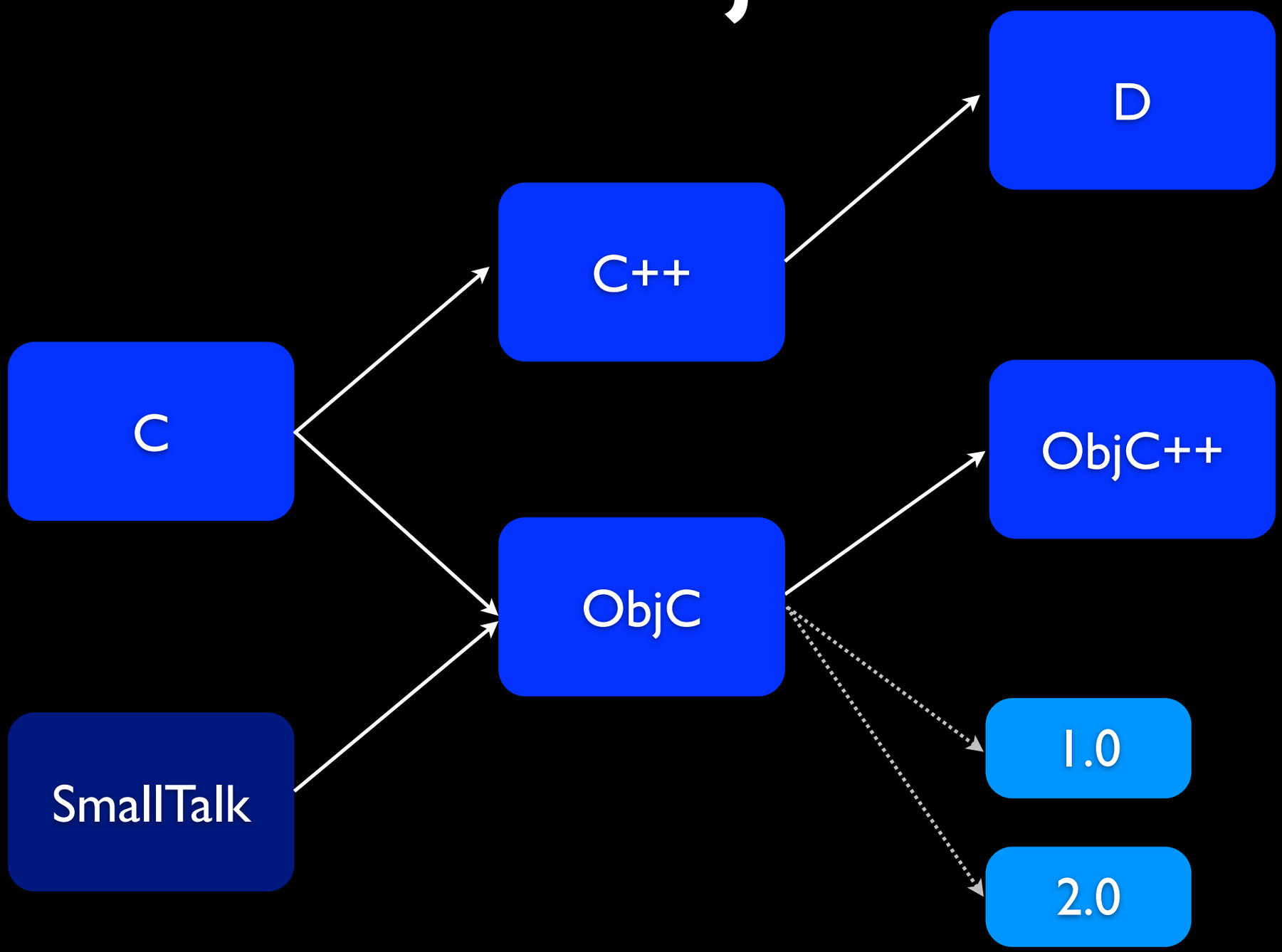
ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC



ObjC entspringt C und Smalltalk, was man an der Syntaxt noch gut erkennen kann. Auch „böse“ Unterformen wie ObjC++ sind möglich. Diese Dateien heißen nicht .m sondern .mm Durch ObjC 2.0 ist die Punkt-Syntax übernommen worden.

ObjC

- OO
- Statische und dynamische Typisierung
- Obermenge von C: Erweiterung des ANSI-C Standart
- OO-Konzepte: Kapselung, Polymorphismus, (einfache) Vererbung, ...

Objects are **dynamically typed**. In source code (at compile time), any object variable can be of type id no matter what the object's class is. The exact class of an id variable (and therefore its particular methods and data structure) isn't determined until the program runs.

Static typing: If a pointer to a class name is used in place of id in an object declaration, the compiler restricts the value of the declared variable to be either an instance of the class named in the declaration or an instance of a class that inherits from the named class.

Statische Typisierung ermöglicht das üBerprüfen von Typen zur Compilezeit (Compiler warnt!)

Klassen in ObjC

- Mechanismus, um Daten und Funktionen auf den Daten zu kapseln
- Alle Klassen erben von NSObject
- Benötigt:
  - Interface (.h) und Implementation(.m)

Fügt Syntax zur Definition von Klassen hinzu
Üblicherweise trennt man Interface von Implementierung --> .h und .m
Es können mehr als eine Klasse in einer Datei definiert/implementiert werden, üblich ist es aber dies streng zu trennen.
Die Definition kann in einer Datei mit beliebiger Endung geschehen.
Die Implementierung muss in einer Datei mit der Endung .m (wg. XCode) geschehen.

Klassen in ObjC

MyClass.h

```
#import <Foundation/Foundation.h>

@interface MyClass : NSObject {

}

@end
```

MyClass.m

```
#import "MyClass.h"

@implementation MyClass

@end
```

If the colon and superclass name are omitted, the new class is declared as a root class, a rival to the `NSObject` class.

#import is identical to `#include`, except that it makes sure that the same file is never included more than once.

Interfaces:

- Beschreiben den Entwicklern die Vererbungshierarchie
- Sagen dem Compiler, welche Variablen eine Instanz besitzt
- Sagen dem Entwickler, welche Variablen geerbt werden können
- Sagen dem Entwickler, welche Methoden er aufrufen kann (andere können aus dem Interface weggelassen werden)

Methoden

- 2 Arten
  - Klassenmethoden

```
+ (id)dictionaryWithCapacity:(NSUInteger)numItems;
```

  - Instanzmethoden

```
- (void)setValue:(id)value forKey:(NSString *)key;
```

Es gibt keine privaten, protected oder public methods! Alle Methoden im Interface sind public!

Method Type Identifier, Return Type, Signature Keyword(s) with Parameter Type(s) and Parameter Name(s)

Man kann einer Klassenmethode und einer Instanzmethode den gleichen Namen geben (unüblich)

Man kann den Rückgabetyp weglassen, es wird dann id angenommen

Nachrichten in ObjC

```
int l = [string lenght];
```

```
UIColor* color = [[self view] backgroundColor];
```

```
string = nil;   !!!
int l2 = [string lenght];
```

Es gibt keine privaten, protected oder public methods! Alle Methoden im Interface sind public!

Method Type Identifier, Return Type, Signature Keyword(s) with Parameter Type(s) and Parameter Name(s)

Man kann einer Klassenmethode und einer Instanzmethode den gleichen Namen geben (unüblich)

Man kann den Rückgabetyp weglassen, es wird dann id angenommen

Eigenschaften in ObjC

```
int l = string.lenght;
```

```
UIColor* color = self.view.backgroundColor;
```

```
string = nil;   !!!
int l2 = string.lenght;
```

Es gibt keine privaten, protected oder public methods! Alle Methoden im Interface sind public!

Method Type Identifier, Return Type, Signature Keyword(s) with Parameter Type(s) and Parameter Name(s)

Man kann einer Klassenmethode und einer Instanzmethode den gleichen Namen geben (unüblich)

Man kann den Rückgabetyp weglassen, es wird dann id angenommen

MyClass.h

```
#import <Foundation/Foundation.h>

@interface MyClass : NSObject {
    double someDoubleValue;
}
-(NSNumber*)characterCountOfString:(NSString*)string;

@end
```

MyClass.m

```
#import "MyClass.h"

@implementation MyClass

-(NSNumber*)characterCountOfString:(NSString*)string{
  return [NSNumber numberWithInt:[string length]];
}
@end
```

characterCountOfString: muss nicht im Header definiert sein, sie ist dann sozusagen private. Wenn jemand den Namen kennt kann er sie aber trotzdem aufrufen.
Der Compiler warnt bei einem Aufruf (wir wollen aber keine einzige Warnungen)

Datentypen

Primitive, Aliase und „NonObjects“

int, float,
double,
BOOL, void
nil, Nil, NULL

SEL,
IBOutlet,
IBAction

id (!)
CGRect

„Objects“

NSArray*
UIButton*
CALayer*
CLLocation*
ABPerson*
...

NS**
UI**

Besonderheiten und Spezialitäten

int*
int**
void*
...
static
const
extern
unsigned
...
id<myDelegate>
oneway, inline, ...

nil, Nil, NULL sind UNTERSCHIEDLICH
SEL, IBOutlet, IBAction die void* Aliase für bestimmte Zwecke
id ist ein Objekt (und zwar das „Urobjekt“ von dem alle erben
CGRect ist ein struct

es gibt noch viel viel mehr Konfigurationen, die hauptsächlich für spezielle Fälle benötigt werden.

Xcode

- War einmal der ProjectBuilder
- Ist in Applescript geschrieben
- Mag keine SCM
- In Version 4 ist alles neu

Xcode kann durch Skripte erweitert und in seiner Funktionalität erweitert werden.
Insgesamt nimmt Xcode sehr viele Aufgaben ab und organisiert recht viel, daher ist eine genauere Betrachtung von Xcode sehr hilfreich.
Im Umgang mit SCMs ist Xcode (leider) sehr anstrengend und fehleranfällig, daher ist hier der Umgang mit der Console zu bevorzugen.

InterfaceBuilder

- Apple‘s kleines Schatzkistchen
- Sehr mächtig! Vor- oder Nachteil?
- Benutzungsoberflächen aus dem InterfaceBuilder werden als nib / xib File gespeichert.
- Objekte aus dem nib-File werden schneller geladen.

Der InterfaceBuilder ist ein sehr mächtiges Werkzeug mit dem sich schnell und bequem Benutzungsoberflächen „bauen“ lassen. Leider ist die Struktur im IB nicht sehr durchsichtig, daher wird der IB öfter benutzt um mit ihm gut umgehen zu können.

Xcode und InterfaceBuilder

Livedemo „Hello Cocoa“

Um die Werkzeuge kennen zu lernen, werden wir sie benutzen :)

Memory Management

Worum geht's

- Wie kann man...
- ... Objekte erzeugen
- ... Objekte zerstören

Da es sich bei Objective-C um eine Programmiersprache handelt, die den Anwendungsentwickler durch einen Garbage Collector vom der Aufgabe des Speichermanagements befreit (zu mindest so lange eine Anwendung für iOS Geräte entwickelt wird), muss sich der Entwickler selber um das Erzeugen UND das Zerstören von Objekten kümmern.

alloc, new, copy

- Verantwortung für Freigabe übernehmen
- Retain Count:
  - “Wie viele Objekte kennen mich?”
  - release (-) und retain (+)

Zur Erzeugung von neuen Objekten wird idR auf alloc bzw. copy zurückgegriffen. Sobald ein Objekt seinen Speicherbereich zugewiesen bekommen hat, ist der Entwickler dafür verantwortlich, dass dieser Bereich auch wieder freigegeben wird, sobald das Objekt nicht mehr benötigt wird. Um diesen Zustand, in dem ein Objekt nicht mehr gebraucht wird, zu erkennen, wird das sogenannte Retain Count verwendet. Dabei handelt es sich um einen Zähler, der für jede Instanz individuell speichert, von wie vielen anderen Objekten bzw. Klassen es referenziert wird. Das Retain Count wird durch den Entwickler nicht direkt gesetzt. Statt dessen verwendet er die Methoden “retain” und “release”, um das Retain Count zu inkrementel zu erhöhen bzw. zu senken.

alloc

- Retain Count startet bei 1
  - Foo* f = [[Foo alloc] init];
- Irgendwann muss das Retain Count wieder gesenkt werden (z.B. in dealloc-Methode)
  - [f release];

Wird ein Objekt durch die alloc Methode erzeugt, so besitzt es bereits ein Retain Count von 1. Wird das Objekt nicht mehr benötigt, soll es zerstört werden. Bei der Zerstörung eines Objekts wird dessen dealloc-Methode automatisch aufgerufen. Statt das Objekt mit einem manuellen dealloc-Aufruf aus dem Speicher zu entfernen, wird daher das Retain Count mit einer Release-Nachricht auf 0 gesenkt. Dadurch wird sichergestellt, dass nur die Objekte zerstört werden, die nicht mehr referenziert werden. Gleichzeitig besteht jedoch das Risiko, dass bei falschen oder fehlenden Release-Aufrufen, ein Objekt nie zerstört wird. Diesen Sachverhalt bezeichnet man dann als Memory Leak bzw. Speicherleck.

eine weitere

Release-Methode

autorelease

- “Sende ein release zu einem späteren Zeitpunkt!”
- Einsatz in Factory-Methoden:

```
+ (Foo*) fooWithBar {
    return [[[Foo alloc] initWithBar] autorelease];
}
```

Neben der release-Methode gibt es zusätzlich noch eine autorelease-Methode. Sie sollte nicht dahingehend missverstanden werden, dass sich der Entwickler nicht mehr um eine Speicherfreigabe kümmern müsste. Die Bedeutung des Autorelease ist eine gänzlich andere: Zu einem verzögerten Zeitpunkt soll eine Release-Nachricht automatisch an das Objekt gesendet werden. Dieses Verhalten ist wichtig, wenn Objekte etwa in einer Methode erzeugt und als Rückgabewert zurückgegeben werden. Würde in dem Beispiel auf der Folie statt eines Autorelease ein “normales” Release verwendet werden, so würde zwar eine Foo-Instanz erzeugt und initialisiert werden, jedoch würde noch bevor das Objekt zurückgegeben werden kann das Retain Count auf 0 gesenkt werden. Als Ergebnis würde ein Zeiger auf eine Speicheradresse zurückgegeben werden, an der kein Objekt mehr existiert. Ein Programmabsturz wäre die Folge!

convenience constructor

- [MYFoo fooByBar:23];
- [NSString stringWithString:@”foobar”];
- erzeugte Objekte in Autorelease Pool
- Autorelease Pools leben nur für einen Runloop-Durchlauf
- Runloop-Ende ist der Zeitpunkt des verzögerten release

Das Implementierungsbeispiel auf der vorherigen Folie zeigt neben der Verwendung des Autorelease auch indirekt, wie sog. Convenience Constructors umgesetzt werden. Hierbei handelt es sich um Klassenmethoden, die eine Instanz der Klasse durch “alloc” und “init” erzeugen, um die Instanz dann mit einem autoreleased als Rückgabewert auszugeben. Ein durch einen Convenience Constructor erhaltenes Objekt besitzt zwar ein Retain Count von 1, jedoch befindet sich das Objekt in einem Autorelease Pool. Sobald dieser Pool zerstört wird, wird eine Release-Nachricht an alle dort enthaltenen Objekte gesendet. Die Zerstörung des Pools erfolgt mit jedem Durchlauf der sog. “Runloop”.

Vergleich

alloc + release

```
-(void) someMethod {
    Foo* f = [[Foo alloc] initWithBar];
    [f doSomething];
    [f release];
}
```

convenience constructor

```
-(void) someMethod {
    Foo* f = [Foo fooWithBar];
    [f doSomething];
}
```

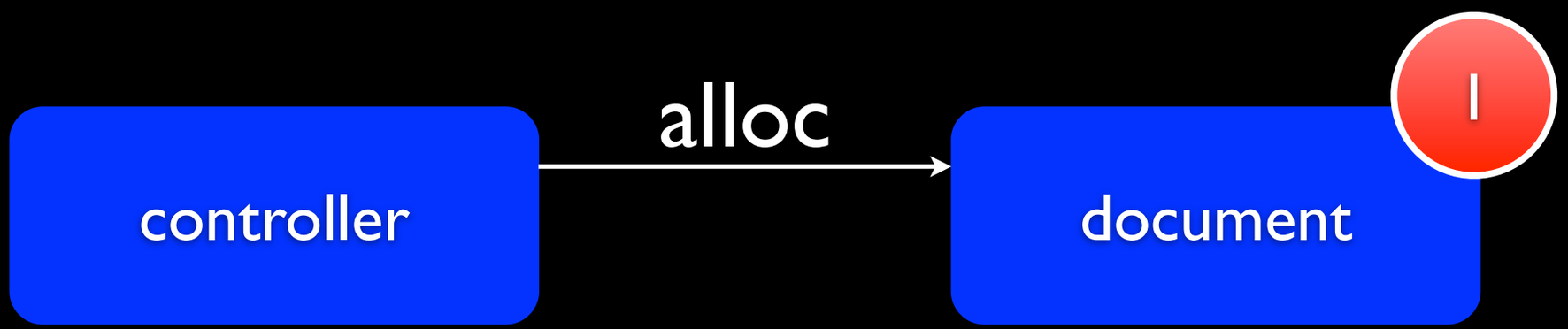
Die beiden Codebeispiele auf dieser Folie zeigen die unterschiedlichen Möglichkeiten, wie Instanzen von Klassen erzeugt und aus dem Speicher entfernt werden können: Entweder durch alloc, init, release und durch einen Convenience Constructor, welcher ein Autorelease auf die erzeugte Instanz anwendet.

eine Problematik

retain cycle

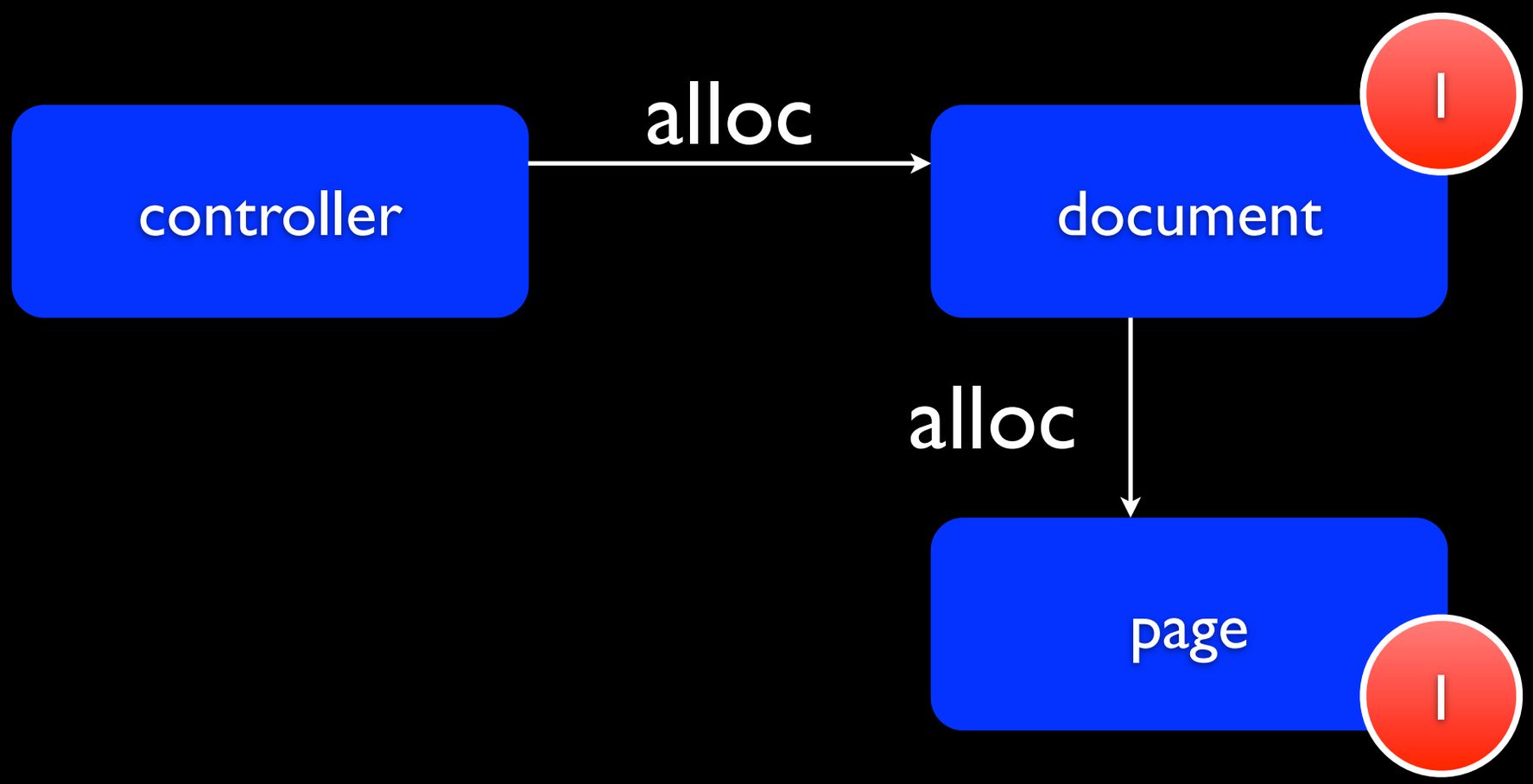
In Bezug auf das Retain Count und die daran gebundene Objektzerstörung gibt es eine Problematik, die zu Speicherlecks führen kann: Retain Cycles. Ein Retain Cycle zeichnet sich dadurch aus, dass zwei Objekte gegenseitig (d.h. zyklisch) ihr Retain Count konstant halten, jedoch nicht mehr von der eigentlichen Anwendung (z.B. einem Controller) referenziert werden. Dadurch bilden im Beispiel auf der Folie die Document- und die Page-Instanz eine Insel im Speicher, die jedoch nicht mehr beim Controller bekannt ist. Ursache für diesen Zustand ist, dass das Kindobjekt (in diesem Fall die Page-Instanz) an sein Elternobjekt eine Retain-Nachricht gesendet hat.

retain cycle



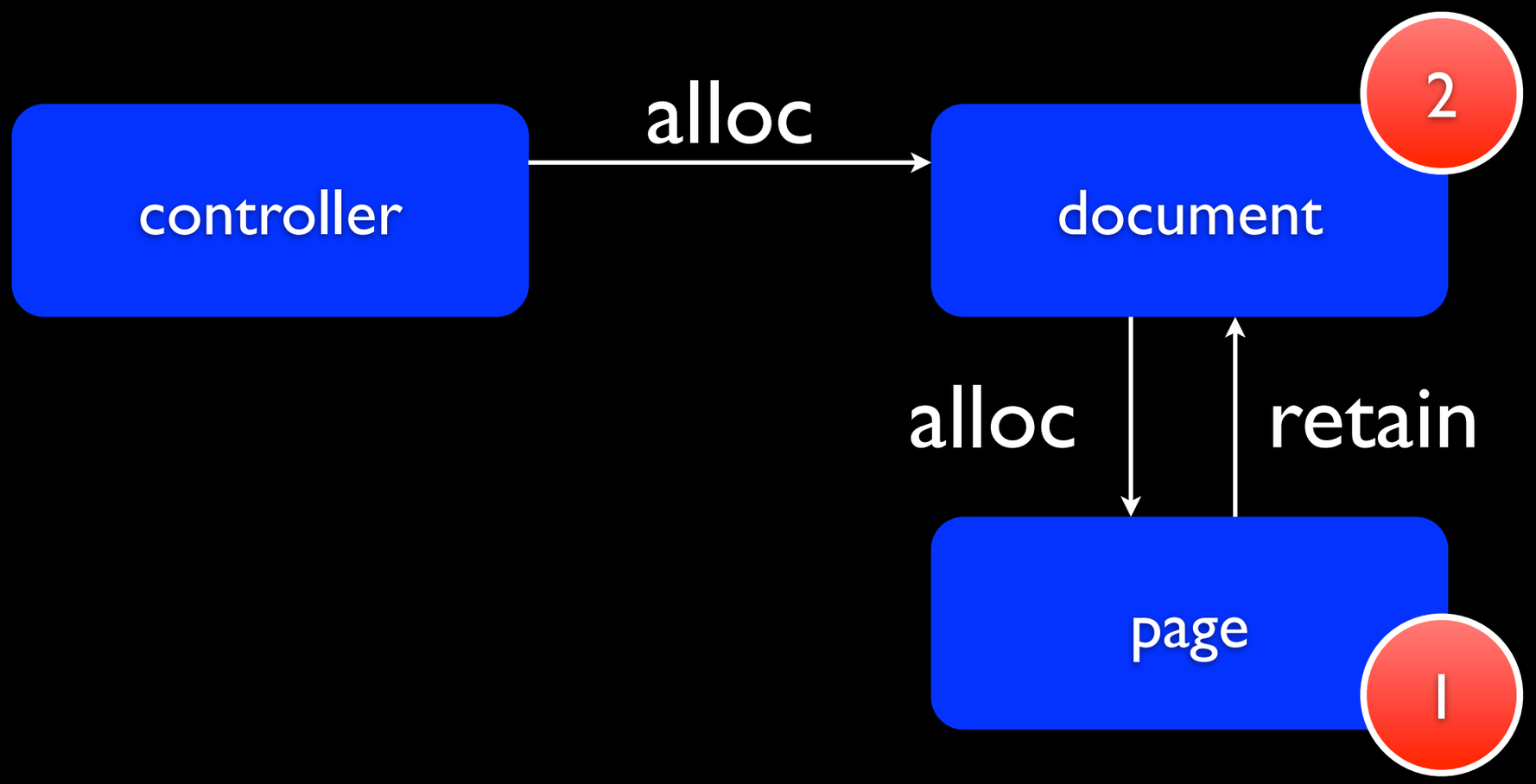
In Bezug auf das Retain Count und die daran gebundene Objektzerstörung gibt es eine Problematik, die zu Speicherlecks führen kann: Retain Cycles. Ein Retain Cycle zeichnet sich dadurch aus, dass zwei Objekte gegenseitig (d.h. zyklisch) ihr Retain Count konstant halten, jedoch nicht mehr von der eigentlichen Anwendung (z.B. einem Controller) referenziert werden. Dadurch bilden im Beispiel auf der Folie die Document- und die Page-Instanz eine Insel im Speicher, die jedoch nicht mehr beim Controller bekannt ist. Ursache für diesen Zustand ist, dass das Kindobjekt (in diesem Fall die Page-Instanz) an sein Elternobjekt eine Retain-Nachricht gesendet hat.

retain cycle



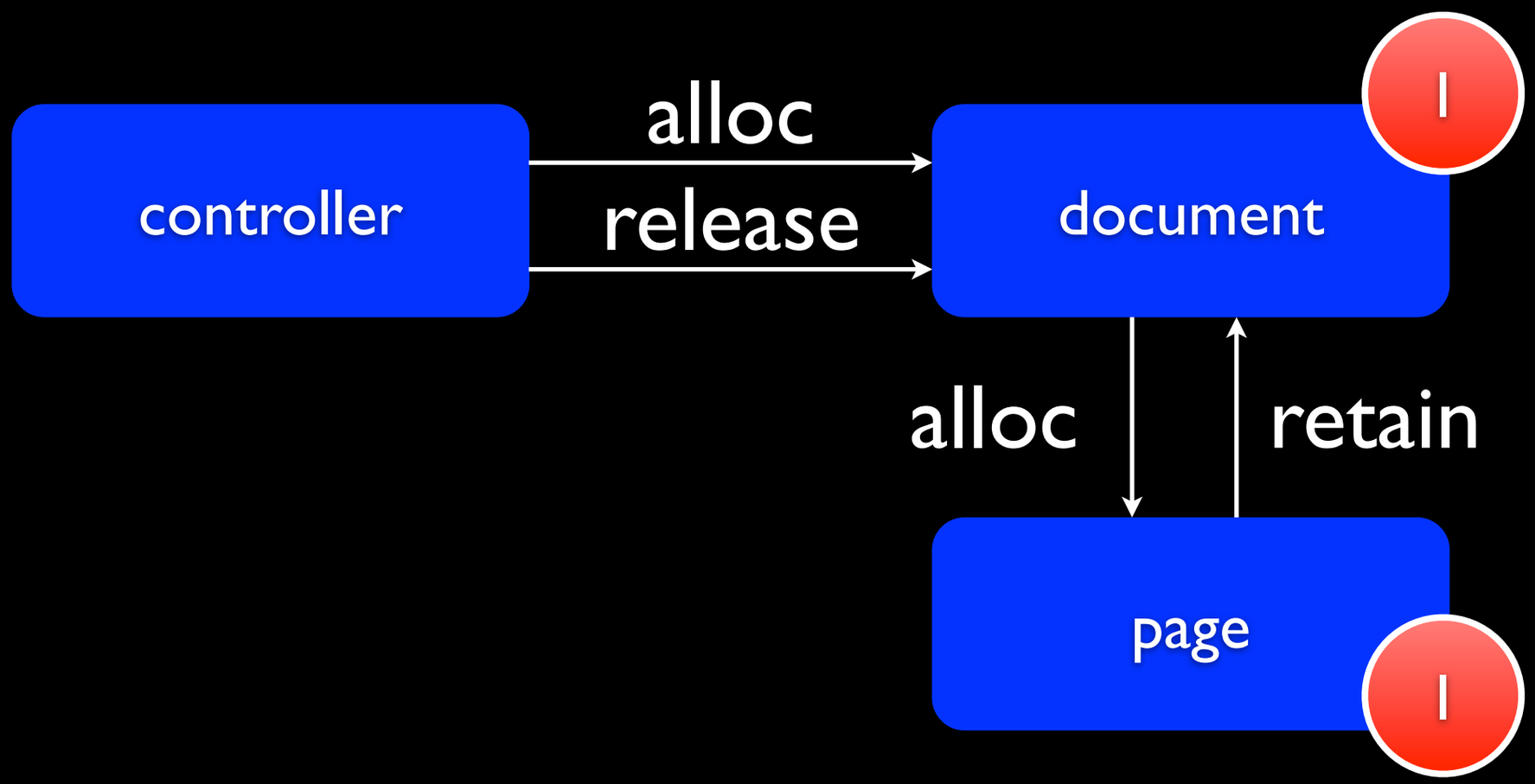
In Bezug auf das Retain Count und die daran gebundene Objektzerstörung gibt es eine Problematik, die zu Speicherlecks führen kann: Retain Cycles. Ein Retain Cycle zeichnet sich dadurch aus, dass zwei Objekte gegenseitig (d.h. zyklisch) ihr Retain Count konstant halten, jedoch nicht mehr von der eigentlichen Anwendung (z.B. einem Controller) referenziert werden. Dadurch bilden im Beispiel auf der Folie die Document- und die Page-Instanz eine Insel im Speicher, die jedoch nicht mehr beim Controller bekannt ist. Ursache für diesen Zustand ist, dass das Kindobjekt (in diesem Fall die Page-Instanz) an sein Elternobjekt eine Retain-Nachricht gesendet hat.

retain cycle



In Bezug auf das Retain Count und die daran gebundene Objektzerstörung gibt es eine Problematik, die zu Speicherlecks führen kann: Retain Cycles. Ein Retain Cycle zeichnet sich dadurch aus, dass zwei Objekte gegenseitig (d.h. zyklisch) ihr Retain Count konstant halten, jedoch nicht mehr von der eigentlichen Anwendung (z.B. einem Controller) referenziert werden. Dadurch bilden im Beispiel auf der Folie die Document- und die Page-Instanz eine Insel im Speicher, die jedoch nicht mehr beim Controller bekannt ist. Ursache für diesen Zustand ist, dass das Kindobjekt (in diesem Fall die Page-Instanz) an sein Elternobjekt eine Retain-Nachricht gesendet hat.

retain cycle



In Bezug auf das Retain Count und die daran gebundene Objektzerstörung gibt es eine Problematik, die zu Speicherlecks führen kann: Retain Cycles. Ein Retain Cycle zeichnet sich dadurch aus, dass zwei Objekte gegenseitig (d.h. zyklisch) ihr Retain Count konstant halten, jedoch nicht mehr von der eigentlichen Anwendung (z.B. einem Controller) referenziert werden. Dadurch bilden im Beispiel auf der Folie die Document- und die Page-Instanz eine Insel im Speicher, die jedoch nicht mehr beim Controller bekannt ist. Ursache für diesen Zustand ist, dass das Kindobjekt (in diesem Fall die Page-Instanz) an sein Elternobjekt eine Retain-Nachricht gesendet hat.

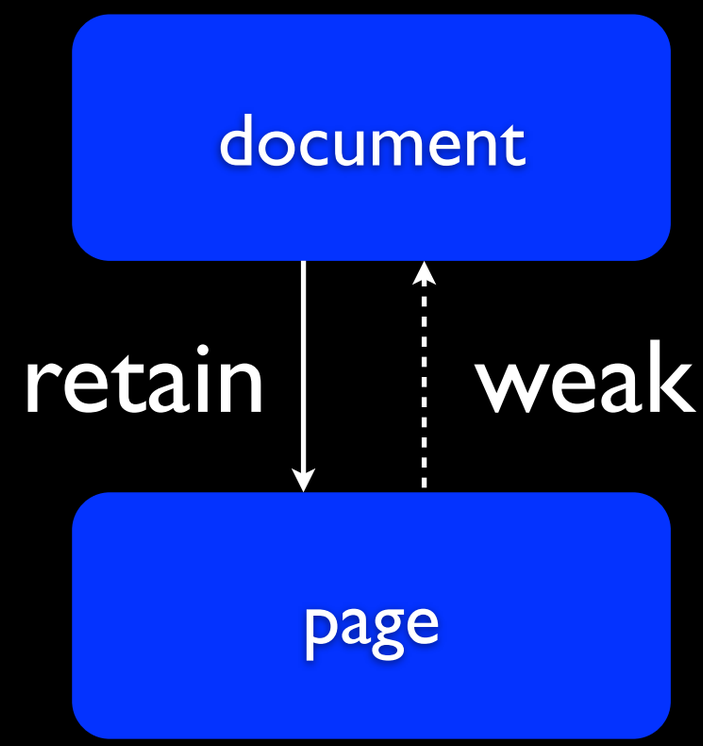
weak references

- nicht sichergestellt, dass das Objekt existiert
- nicht immer will man retain an ein Objekt senden
- Delegate Objekte sind idR. weak referenziert

Um dieser Problematik aus dem Weg zu gehen, werden sog. “weak references” verwendet. Dabei handelt es sich um schwache Referenzen, die nicht sicherstellen können, dass das referenzierte Objekt existiert. D.h. statt durch eine Retain-Nachricht die Existenz des referenzierten Objekts sicherzustellen, wird lediglich die Annahme getroffen, dass das Elternobjekt seine Kindobjekte “überlebt”.
Ein häufiger Anwendungsfall für schwache Referenzen ist die Referenzierung von Delegate-Objekten.

retain cycles

- Besitzer “retained” von ihm instanziierte Objekte
- Die instanziierten Objekte “retained” den Besitzer idR. nicht!



Diese Folie zeigt nochmals wie das Elternobjekt (“Besitzer”) die Existenz seiner Kinder (d.h. die von ihm erzeugten Objekte) durch jeweils eine Retain-Nachricht sicherstellt. Die Kinder hingegen kennen zwar ihr Elternobjekt, müssen jedoch damit rechnen, dass das Elternobjekt nicht mehr existiert.

noch was Praktisches

Properties

- ermöglichen es Getter und Setter zu generieren
- Deklaration des Property gibt Retain-verhalten an
  - @property (nonatomic, **retain**)
  - @property (nonatomic, **assign**)

Bei der Behandlung von Speichermanagement, soll auch das Objective-C Sprachfeature der sog. Properties betrachtet werden. Hierbei handelt es sich um ein Konstrukt, welches es ermöglicht Getter- und Setter-Methoden für Variablen auszuzeichnen und durch den Compiler generieren zu lassen. Die Deklaration eines Properties erfolgt im Header-File und ist durch ein @property am Zeilenanfang gegenzeichnet. Anschließend folgt in Klammern eine Konfiguration, wie das Property verwaltet werden soll, falls auf dieses mittels der Getter und Setter zugegriffen wird. Beim Thema Speichermanagement sind hier die Schlüsselwörter “retain” und “assign” wichtig. Retain sagt aus, dass übergebene Objekte bei einem Setter-Aufruf retained und ggf. vorhandene alte Objekte released werden. Assign hingegen setzt das Konzept der schwachen Referenzen um (weak references). Hierbei findet demnach kein Retain und Release statt.

Objektzerstörung

dealloc

- Methode in einer Klasse, die bei Erreichen eines Retain Count von 0 aufgerufen wird
- keine manuellen dealloc Aufrufe!!!
- dealloc-Methode muss zur Freigabe nicht mehr benötigter Resourcen verwendet werden!

Auf den vorherigen Folien ist bereits angesprochen worden, dass die dealloc-Methode niemals direkt im eigenen Code aufgerufen wird. Statt dessen wird das Konzept des Retain Counts verwendet, um durch die Laufzeitumgebung die Objektzerstörung durchführen zu lassen. Dies passiert immer dann, sobald das Retain Count einer Instanz bei 0 angekommen ist.
Wichtig bei der Implementierung eigener Klassen ist es, dass in einer eigenen Implementation der dealloc-Methode alle Resourcen freigegeben werden müssen, die nach der Objektzerstörung nicht mehr existieren sollten. Dies kann im einfachsten Fall durch Senden von Release-Nachrichten erfolgen. Ggf. sind aber auch andere oder zusätzliche Schritte notwendig (z.B. dann, wenn C-Bibliotheken verwendet werden, die nicht auf Retain Counter zurückgreifen können).

dealloc

```
- (void)dealloc {
    [foo release];
    [bar release];
    [pirates release];

    [super dealloc];
}
```

Dieses Codebeispiel zeigt den häufigsten Fall der Resourcenfreigabe durch Release-Nachrichten. Ein Setzen der Variablen bzw. Pointer auf “nil” ist nicht notwendig!

Zusammenfassung

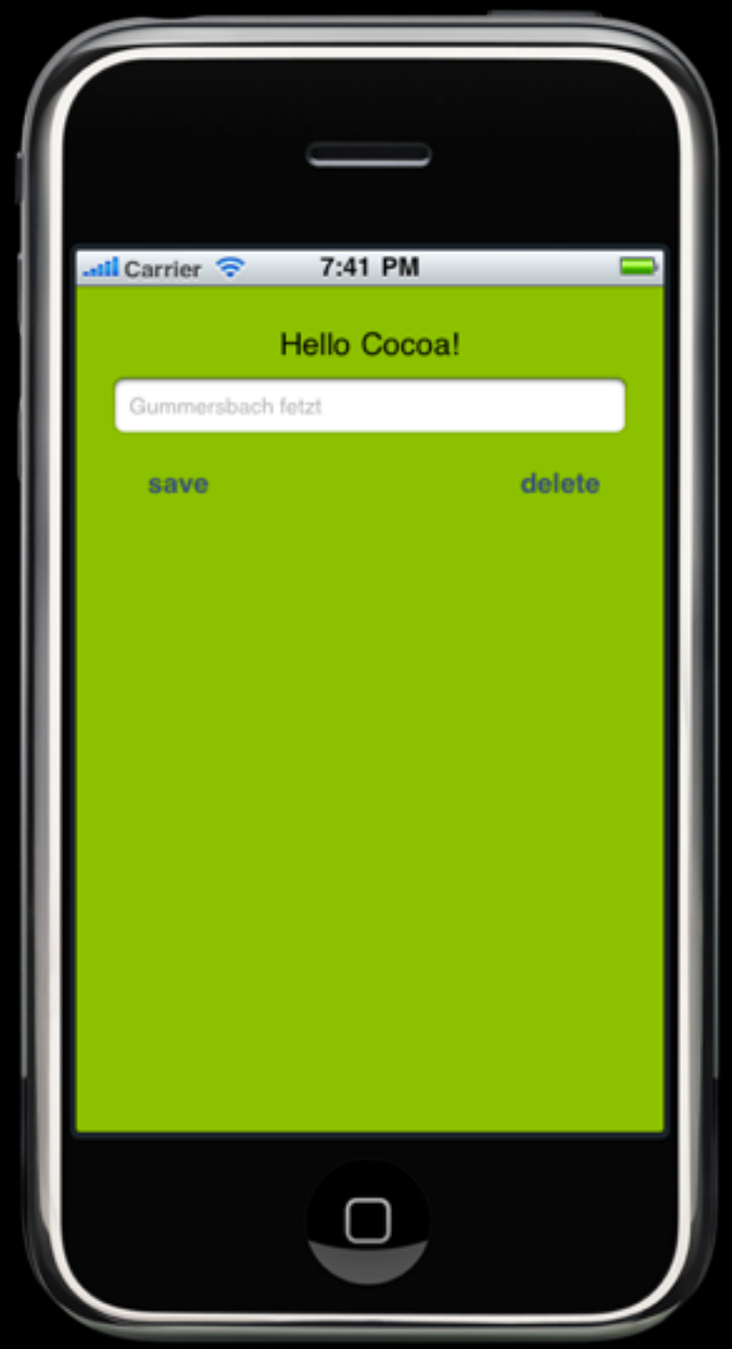
- Objekterzeugung durch **alloc** und **convenience constructor**
- Objektzerstörung durch **release** und **autorelease**
- **Retain Cycles** und **weak** References
- Bedeutung von **dealloc**

Auf den letzten Folien ist gezeigt worden, dass die Objekterzeugung entweder durch direkte alloc-Aufrufe aber auch durch sog. Convenience Constructor erfolgen kann. Die Objektzerstörung ist zwar Aufgabe des Anwendungsentwicklers, jedoch wird diese nicht direkt vorgenommen, sondern erfolgt durch Senken des Retain Counts. Hierzu werden die Methoden Release und Autorelease verwendet. Durch das Konzept des Retain Counts entsteht ein mögliches Speicherleck durch die Erzeugung von Retain Cycles. Um diese zu vermeiden, sollten Kindobjekte ihre Erzeuger nur schwach referenzieren. Abschließend ist dargestellt worden, wie die dealloc-Methode verwendet wird, um bei der Objektzerstörung den Speicher von nicht mehr benötigten Resourcen zu befreien.

- https://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.html

Aufgabe

- Baut “Hello Cocoa” nur mit Code d.h. **OHNE** den Interface Builder nach!



Delegate und Methoden

Methoden

- Gute Methoden ersparen Dokumentation
- Methodentyp (Rückgabewert)Methodenname: (TypDesParameters*)parameterName wasJetztPassiert:(TypDesParameters*)parameterName
- - (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
- Wenn Methoden der Oberklasse überschrieben werden sollte [super]; aufgerufen werden.

Wichtig ist sich mit den Methoden zu beschäftigen. Methodendeklarationen können in aller Regel so geschrieben werden das sie die Dokumentation ersetzen.

Wichtig ist: das Aufrufen von [super ....] ist entscheidend für den Ablauf eines Programms. Super immer weiterleiten außer ihr wollt erreichen das die Eventkette bei euch endet.

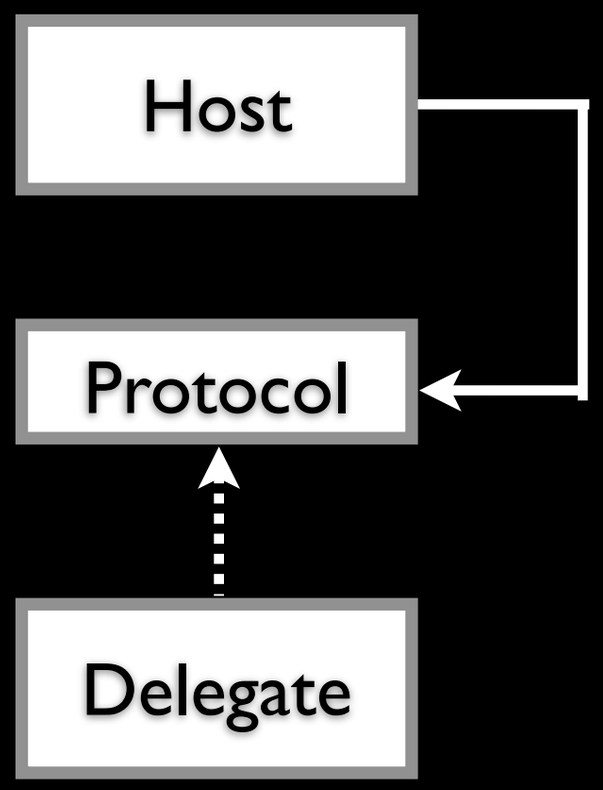
Keyboard of Doom

- Die Tastatur wird automatisch angezeigt, sobald ein Textfeld/-view angetippt wird
- Der Return-Button ist (vorerst) ohne Funktion

in der vorigen Übung haben wir das „Keyboard of Doom“ erzeugt. Wir können die Tastatur nicht schließen.

Delegation Pattern

- Aufgaben und Ereignisse werden an weitere Klassen delegiert
- Z.B. um auf Ereignisse zu reagieren
- Intensive Nutzung in Cocoa



Oft geteilt in formales und informelles Protokoll
Host testet vorher, ob methode implementiert wird
In algorithmen können so allgemeiner formuliert werden und einzelne Schritte speziell implementiert werden
Verhalten kann ohne ableiten geändert/erweitert

Delegates

```
@class GPSController;
@protocol GPSControllerDelegate
- (void) gpsController:(GPSControllerr*)controller hasNewLocation:(CLLocation* )loc;
@end
```

```
@interface MapNavigationViewController < GPSControllerDelegate > {...} @end

@implementation MapNavigationViewController
-(id)init{
  if(self = [super init]){
      gpsCon = [GPSController sharedInstance];
      gpsCon.delegate = self;
      [gpsCon start];
... } }
```

```
#pragma mark GPSController delegate methods
- (void) gpsController:(GPSControllerr*)controller hasNewLocation:(CLLocation* )
loc{
   NSLog(@"%f %f", loc.coordinate.latitude, loc.coordinate.longitude);
}
```

Beispiel

Aufgabe

- Implementiert den entsprechenden Delegate um die Tastatur des UITextFields wieder zu schließen.
- Schaut euch die weiteren Methoden an, die vom Delegate angeboten werden.

Rotation und Scaling

Rotation

- Behandelt den Umgang der Anwendung mit den Elementen einer Benuztungsoberfläche
- Vier Ausrichtungen möglich:

UIDeviceOrientationPortrait

UIDeviceOrientationLandscapeLeft UIDeviceOrientationLandscapeRight

UIDeviceOrientationPortraitUpsideDown

- Die Rotation ist Abhängig von der Orientierung der Statusbar

Rotation

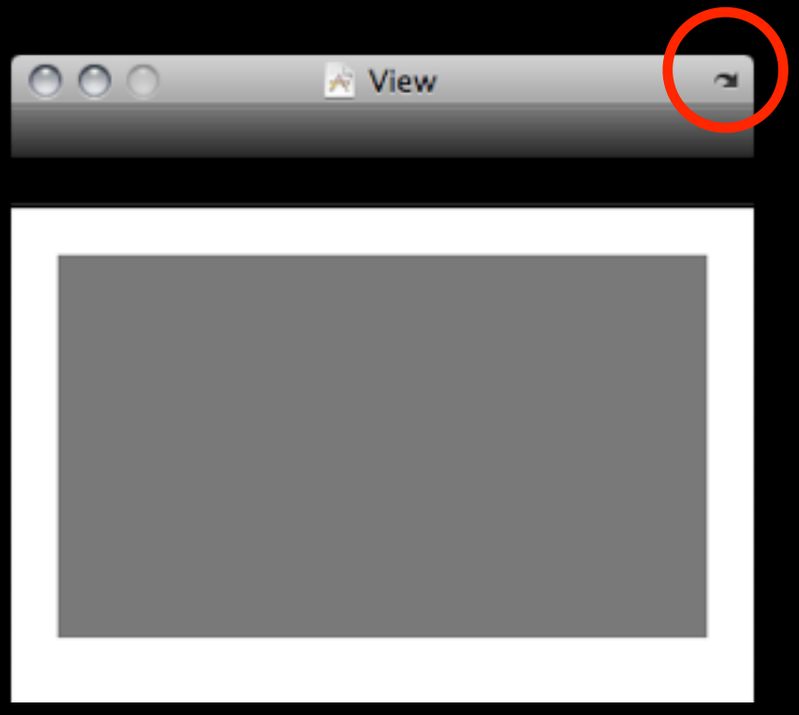
- Zahlreiche Zustände verfolgbar:
  - willRotateToInterfaceOrientation:duration
  - willAnimateSecondHalfOfRotationFromInterfaceOrientation
  - didRotateFromInterfaceOrientation
  - ...

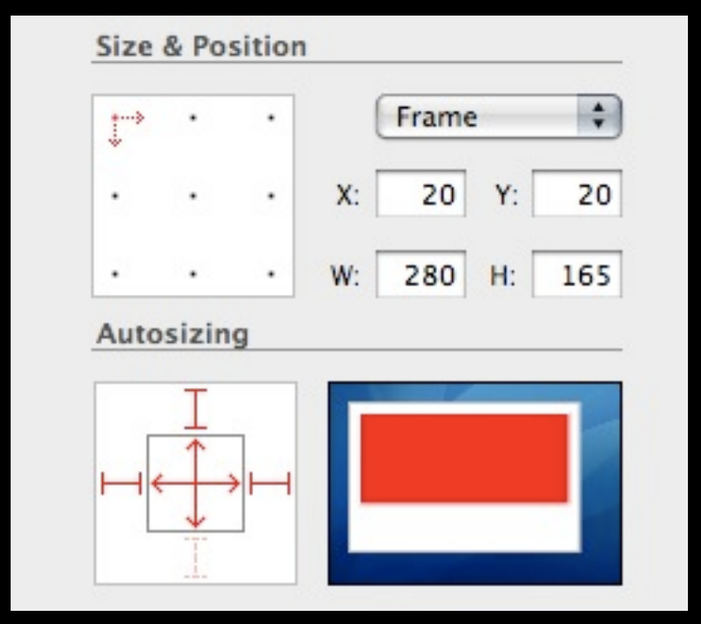
beim verfolgen der Zustände ist es wichtig [super ...] aufzurufen damit der Delegate die Animation zu ende fahren kann.s

Scaling

- Wird vorausgesetzt:
  Skaliert sich die Anwendung richtig, fällt es dem Benutzer nicht auf;
  Skaliert sie sich nicht richtig gibt es negative Bewertungen
- Abhängig von der Anzahl der Elemente auf dem sichtbaren Screen
- Abhängig vom Status des Gerätes

Rotation und Scaling





die Rotation im Interface Builder kann mit einem Click auf den kleinen Pfeil getan werden. Der Simulator läßt sich mit Apfel+Pfeil links / rechts drehen.

Rotation

- Kann auch „von Hand“ erstellt werden.
- Abgeleitet von UIViewAutoresizing

```
enum {
   UIViewAutoresizingNone                 = 0,
●  UIViewAutoresizingFlexibleLeftMargin   = 1 << 0,
   UIViewAutoresizingFlexibleWidth        = 1 << 1,
●  UIViewAutoresizingFlexibleRightMargin  = 1 << 2,
●  UIViewAutoresizingFlexibleTopMargin    = 1 << 3,
   UIViewAutoresizingFlexibleHeight       = 1 << 4,
●  UIViewAutoresizingFlexibleBottomMargin = 1 << 5
};
typedef NSUInteger UIViewAutoresizing;
```

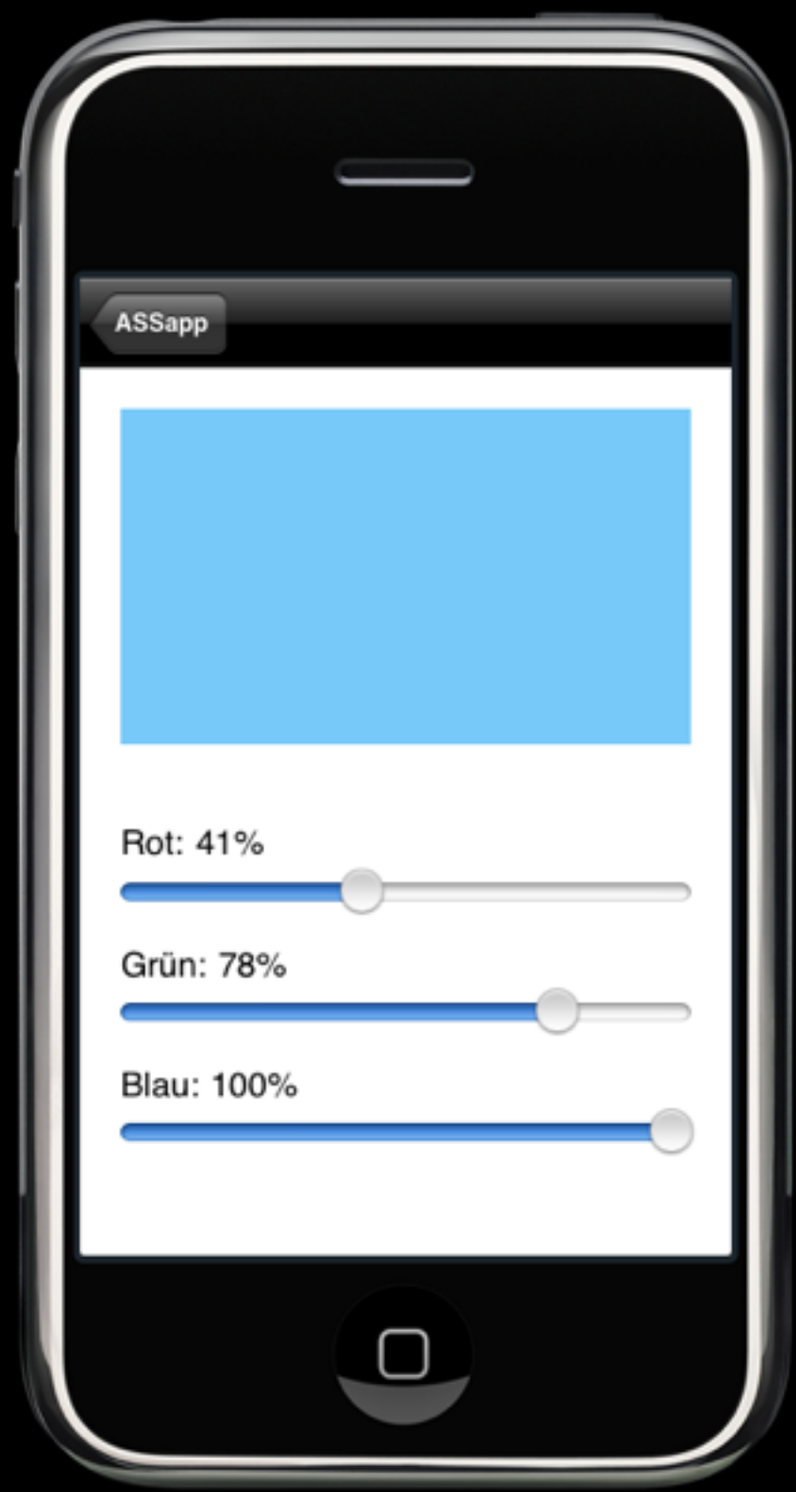
die rot markieren flags entsprechen genau dem Gegenteil dessen was im Interface Builder ausgewählt wird. Im Code muß man also genau darauf achten was für flags man setzt

Rotation

- Live Demo mit dem Interface Builder

Aufgabe

- Schreibt eine Anwendung die folgende Eigenschaften besitzt:
  - Die Slider verändern die Farbwerte des Views
  - Alle Komponenten sollen auf Rotation reagieren
  - Zusatzaufgabe: Label mit Prozentangaben einfügen



UIViewController

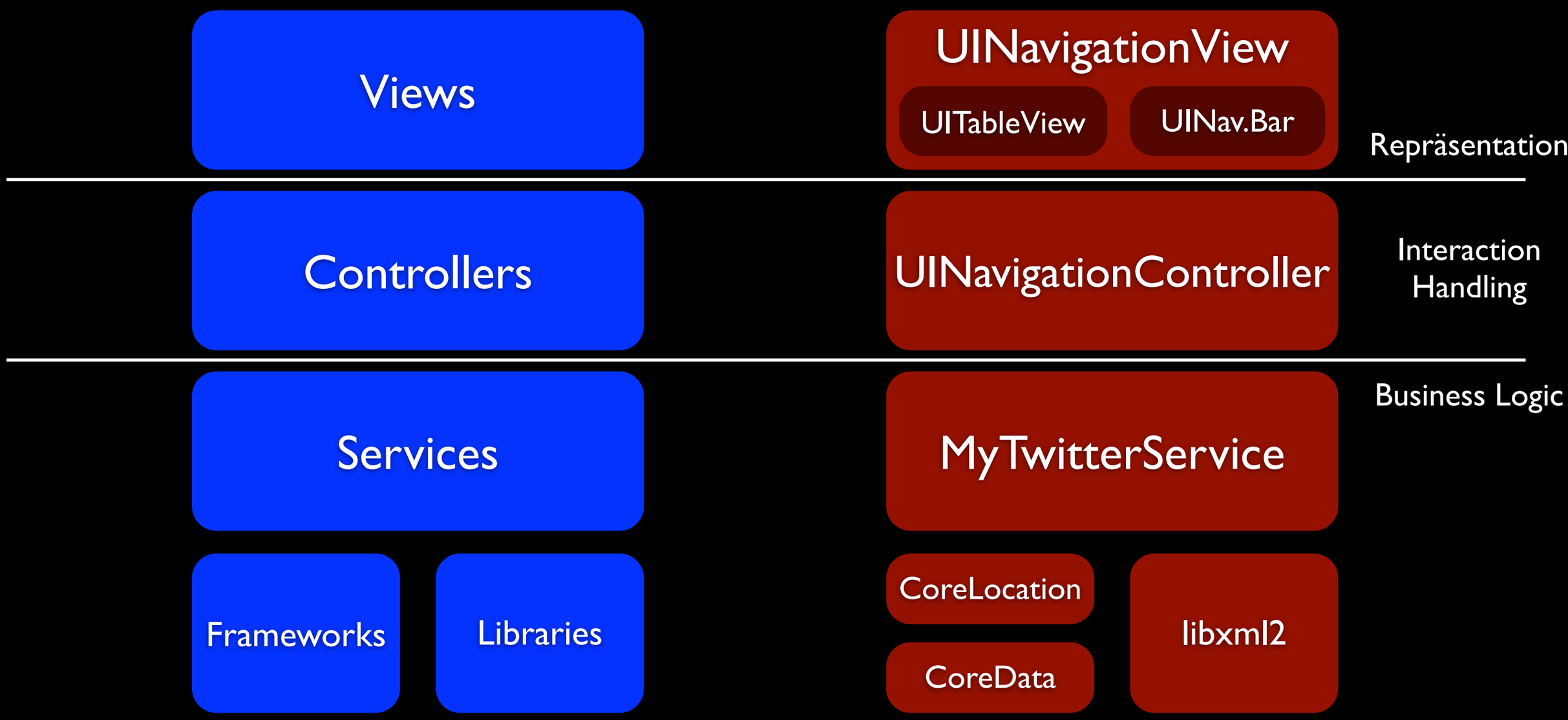
ViewController stellen bei der Entwicklung von iPhone bzw. iPad-Anwendungen eine der zentralsten Komponenten dar. Daher sollen sie im Folgenden detailierter betrachtet werden.

MVC

- **M**odel **V**iew **C**ontroller
- ViewController:
  - Bindeglied zwischen View und Business Logic
  - pro Screen üblicherweise ein ViewController
  - Ausnahme: NavigationBar oder TabBar

Der Name “ViewController” macht einen Zusammenhang mit dem MVC-Enwurfsmuster erkennbar. Dieses Entwurfsmuster legt es einem Anwendungsentwickler nahe, seine Komponenten in drei Arten von Komponenten zu entkoppeln: Model, View und Controller. Das Model enthält das Datenmodell, welches vom Nutzer bearbeitet werden soll. In einem einfachen Fall ließe sich sagen, dass es sich hierbei um die Daten handelt, die man mittels eines ER-Diagramms beschreiben würde. Der View dient ausschließlich der Repräsentation von Daten, die aus dem Model stammen. Der Controller hat die Aufgabe Aktionen am User Interface (d.h. den View-Komponenten) in Veränderungen am Modell zu übersetzen bzw. solche Veränderungen etwa an einer Service-Klasse zu veranlassen.
Im iPhone SDK stellt ein UIViewController daher ebenfalls das Bindeglied zwishen View und Geschäftslogik dar. Unter Geschäftslogik ist in diesem Fall die Logik zu verstehen, die den Sinn und Zweck der Anwendung abbildet.
Pro Screen d.h. pro gefülltem Bildschirm wird üblicherweise genau ein UIViewController verwendet. Diese Regel wird aufgehoben, sobald UINavigationBar oder UITabBar im User Interface verwendet werden.

Anwendungsarchitektur



Eine generische iPhone-Anwendung lässt sich mit der abgebildeten Architektur beschreiben. In der Repräsentationsschicht befinden sich View-Komponenten. D.h. hier würde z.B. ein NavigationView mit seinen SubView-Elementen (TableView & NavigationBar) eingeordnet werden. Da es sich dabei um Elemente handelt, die auf Nutzeraktion reagieren können, werden sie verwendet, um in einer Interaktionsschicht Controller-Objekte von Ereignissen zu benachrichtigen (z.B. “Der Benutzer hat Zelle 3 in Sektion 2 ausgewählt”).
Die UIViewController-Kindklassen sollten nun jedoch nicht direkt die Veränderung von Daten vornehmen, sondern über Service-Klassen eine Abstraktionsschicht verwenden, um die Geschäftslogik ablaufen zu lassen. Dies bietet den Vorteil, dass die ViewController-Kindklassen auf bei komplexen Interaktionsabläufen noch möglichst übersichtlich bleiben und eine Unabhängigkeit der oberen beiden Schichten von verwendeten Frameworks und Softwarebibliotheken gewonnen wird.

Eigenschaften

- UIViewController:
  - kennt genau einen View (darüber auch alle Subviews)
  - definiert, ob ein View drehbar ist
  - Initialisierung eines Views sollte durch die viewDidLoad im Controller erfolgen
  - besitzt ein navigationItem (siehe später NavigationController)

Da nun die Bedeutung der ViewController verdeutlicht worden ist, kann auf einige wesentliche Eigenschaften der ViewController eingegangen werden.
Ein UIViewController kennt maximal einen View. Über diesen View ist es jedoch möglich auch an dessen SubViews zu gelangen.
Der ViewController entscheidet darüber, ob ein View drehbar ist oder nicht. D.h. er definiert die Antwort auf Anfragen, ob eine bestimmte Interface-Orientierung unterstützt wird.
Falls eine Screen nicht komplett mittels des InterfaceBuilders erstellt werden kann, weil etwa die Inhalte erst zur Laufzeit geladen werden können, so werden derartige Initialisierungen innerhalb der viewDidLoad-Methode der ViewController angestoßen.
Jeder ViewController besitzt ein navigationItem-Property, welches die Darstellung des ViewControllers innerhalb innerhalb einer NavigationBar definiert.

ein Screen ist nicht genug

- komplexe Datenstrukturen vs. eingeschränktes User Interface
- Problem: ViewController soll ein Foto an die Business Logic weitergeben

In der Regel sind Anwendungen so komplex, dass ihre gesamte Funktionalität nicht innerhalb eines einzigen Screens umgesetzt werden kann, da die zur Verfügung stehenden Resourcen eingeschränkt sind (kleines Display, fehlende Haptik etc.).
Eine häufig auftretende Situation ist die Auswahl eines Fotos. Hierbei handelt es sich um einen komplexen Vorgang, da etwa zwischen der Fotosammlung und der Kamera gewählt werden kann und da die Fotosammlung hierarchisch gegliedert sowie sehr groß sein kann.

Ein Screen ist nicht genug

- Lösung:
  - ein eigener ViewController mit zugehörigem View wird zur Auswahl des Fotos verwendet
  - ein Delegate Protocol definiert eine Schnittstelle über die das Foto an den Aufrufer zurückgegeben wird (lose Kopplung)

Daher werden Einzelschritte von komplexen Anwendungsfällen mittels eigener unabhängiger ViewController, Views und Delegate-Protokolle umgesetzt.
In Bezug auf das vorhergehende Beispiel bedeutet dies:
Innerhalb eines bzw. mehrerer Views erhält der Benutzer die Möglichkeit ein Foto aus seiner Sammlung auszuwählen und die Auswahl zu bestätigen. Dabei kommen unterschiedliche Views zum Einsatz (z.B. TableView oder ScrollView). Sobald die Auswahl bestätigt wurde, wird mittels einer im Delegate-Protokoll spezifizierten Methode die ursprünglich aufrufende Instanz (idR. ein ViewController) benachrichtigt und das Foto übergeben. Da ein Protokoll und nicht die Klassennamen selbst verwendet werden, wird eine lose Kopplung der Komponenten und damit eine hohe Wiederverwendbarkeit erreicht.

modale ViewController

- ViewController kann einen weiteren ViewController einblenden lassen
- zwei wichtige Methoden:
  - presentModalViewController:animated:
  - dismissModalViewControllerAnimated:
- Beispiel: Wähle ein Foto aus

Um zwischen verschiedenen Views bzw. ViewControllern zu wechseln, gibt es verschiedene Ansätze. Eine Möglichkeit ist die Verwendung der modalen ViewController. Diese werden mittels presentModalViewController und dismissModalViewController angezeigt. Dabei ist zu beachten, dass der presentModalViewController-Methode der anzuzeigende ViewController als Parameter übergeben werden muss. Die dismissModalViewController-Methode wird beim eingeblendeten modalen ViewController selbst aufgerufen und benötigt daher den zu entfernenden ViewController **nicht** erneut als Parameter.

Demo

Durch klicken auf “existing photo” wird presentModalViewController mit einem UIImagePickerViewController als Parameter aufgerufen. Der ImagePickerViewController besitzt ein Delegate-Protokoll, welches vom ViewController des ersten Screens implementiert wird. Sobald der ViewController über die in diesem Protokoll spezifizierte Methode angesprochen wird, sendet er eine dismissModalViewController-Nachricht an den ImagePickerViewController und zeigt das Bild an.

Zusammenfassung

- **keine** Anwendung kommt ohne ViewController aus!
- pro Screen wird **ein** ViewController verwendet
- Controller stellen **Verbindung** zwischen View und der Business Logic her

Es sollte nun klar sein, dass keine Anwendung ohne ViewController auskommt, da jeder Screen mindestens einen ViewController benötigt. Dabei stellen die ViewController eine Verbindung zwischen Geschäftslogik und der Repräsentation am User Interface dar.

Navigation Controller

Neben den modalen ViewControllern gibt es auch eine zweite Möglichkeit wie zwischen mehreren ViewControllern gewechselt werden kann: mit der Hilfe von Navigation Controllern.

Zweck

- Navigation durch hierarchischen Inhalt

Ein Navigation Controller besitzt gegenüber den modalen ViewControllern jedoch eine konkretere Aufgabe, da er zur Navigation in hierarchischem Inhalt konzipiert ist. Diese Navigationsstrukturen sind in vielen Anwendungen vorhanden. Ein Beispiel wären z.B. die Systemeinstellungen.

Funktionsweise

- NavigationController baut einen Stack aus ViewControllern auf
- zwei wichtige Methoden:
  - pushViewController:animated:
  - popViewControllerAnimated:

Um durch den hierarchischen Inhalt zu navigieren, baut der NavigationController einen Stak von ViewControllern auf. Dabei wird der oberste ViewController aktuell durch den NavigationController unterhalb der NavigationBar angezeigt.
Zum Aufbau und Abbau des Stacks bzw. zur Navigation in die Tiefe und wieder zurück zum Ursprung werden die Methoden pushViewController und popViewController verwendet. Die Parametrisierung erfolgt analog zu den Methoden der modalen ViewController. Der einzige Unterschied ist, dass die pushViewController-Methode am NavigationController und nicht am aktuell dargestellten ViewController (d.h. derjenige, der den eigentlichen Inhalt darstellt) aufgerufen wird.

Funktionsweise

```
ProtocolSummaryViewController *protocolSummaryViewController =
[[ProtocolSummaryViewController alloc] initWithNibName:@"ProtocolSummaryViewController"
bundle:nil];

/* do some initialization work */

[self.navigationController pushViewController:protocolSummaryViewController animated:YES];
[protocolSummaryViewController release];
```

Dieses Beispiel erzeugt einen ViewController per Code, lädt dessen View jedoch aus einem XIB-File. Nach der Erzeugung könnten weitere Initialisierungen erfolgen. Anschließend legt der aktuell dargestellte ViewController (self) den neuen ViewController auf den Stack seines NavigationController.
Exkurs: Damit keine Speicherlecks auftreten wird an den per “alloc and init” erzeugten ViewController nach dem Ablegen auf dem Stack eine release-Nachricht geschickt.

NavigationBar

- kann mit Buttons und komplexen aggregierten Views bestückt werden
- kann Titel darstellen

```
self.navigationItem.leftBarButtonItem = self.editButtonItem;   // kommt aus der Elternklasse UIViewController

UIBarButtonItem *rightBarButton = [[UIBarButtonItem alloc] initWithCustomView:segmentedControl];
self.navigationItem.rightBarButtonItem = rightBarButton;
[rightBarButton release];

self.navigationItem.title = @"protocols";
```

Damit ein ViewController in der NavigationBar sinnvoll dargestellt wird, muss er ein NavigationItem mit einem Titel und Steuerungselementen für die NavigationBar befüllen. Dabei kann zum einen auf bereits vorgefertigte Buttons (siehe leftBarButtonItem im Beispiel) oder komplexer gestaltetere UI-Elemente zurückgreifen (siehe rightBarButtonItem im Beispiel).

NavigationBar

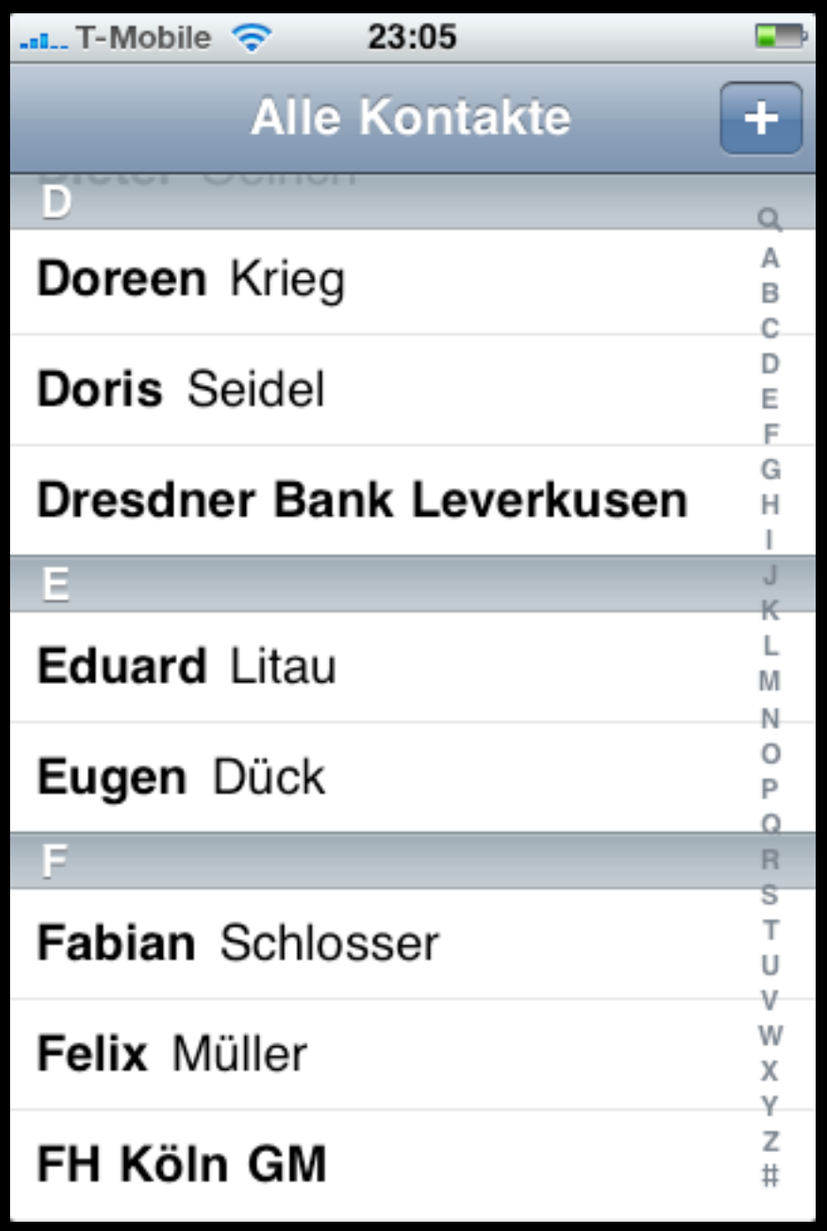
leftBarButtonItem

rightBarButtonItem

Das vorhergehende Beispiel erzeugt in der NavigationBar das auf der Folie dargestellte Bild. Links befindet sich ein standardisierter Edit-Button, in der Mitte der Titel und rechts ein UISegmentedControl, welches im InterfaceBuilder erstellt und dem ViewController injeziert wurde (zur Erinnerung: IBOutlet).

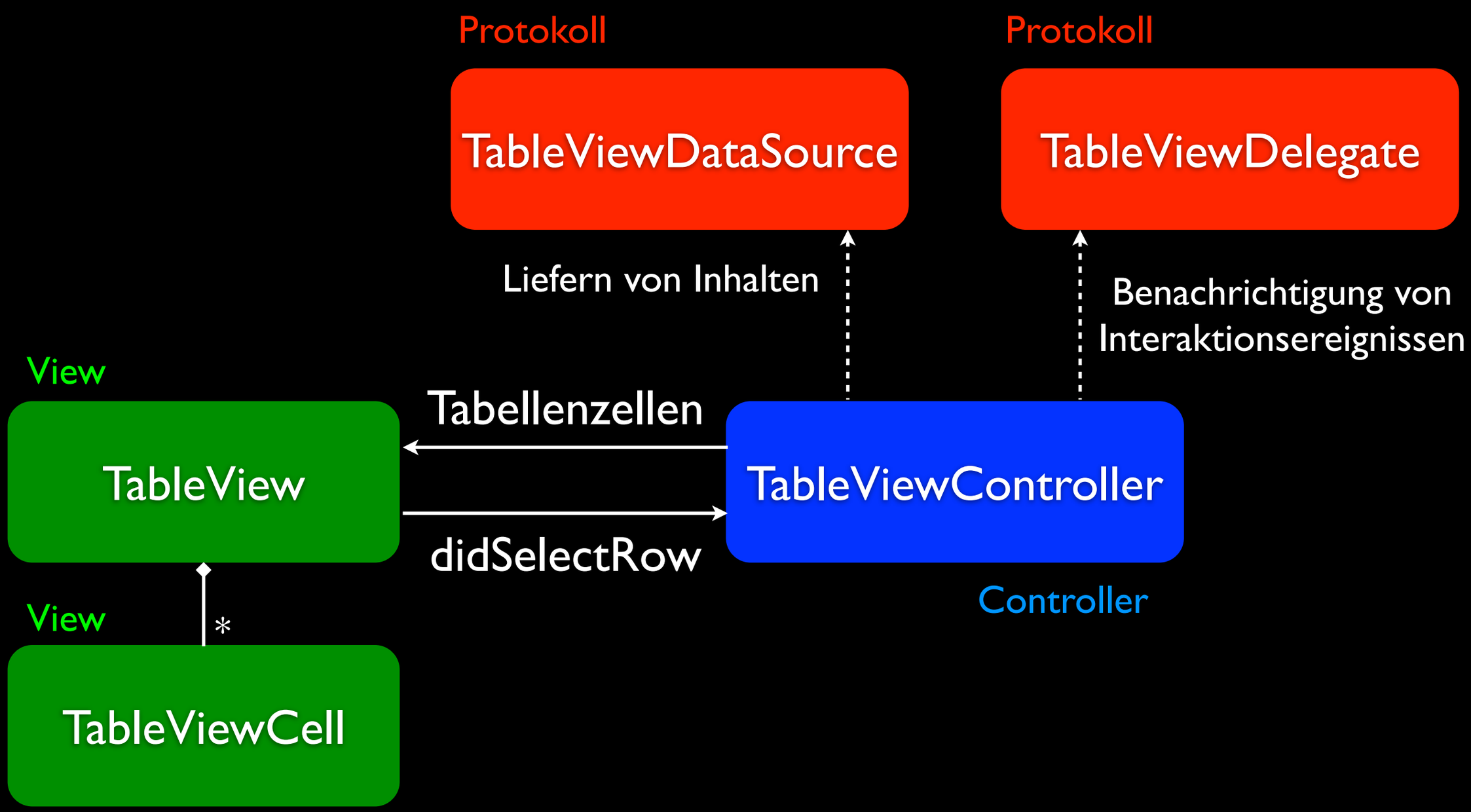
UITableView

Wo begegnen sie einem?



Neben der NavigationBar bzw. dem NavigationController stellt der TableView bzw. der TableViewController eine der meistverwendeten Komponenten in iPhone-Anwendungen dar. Tabellen werden dazu verwendet, um z.B. Kontaktdaten in einem Adressbuch darzustellen oder um Navigationsmöglichkeiten anzubieten, die nicht direkt auf eine Entität abgebildet werden.

Wie funktionieren sie?



Da der TableView ein sehr vielseitige Komponente ist, ist seine Implementierung ebenso komplex. Daher sind in der Abbildung die einzelnen Teilkomponenten farbig gruppiert, falls sie in einem engeren Zusammenhang zueinander stehen.
Die Repräsentation eine Tabelle erfolgt durch einen TableView. Dabei setzt sich der TableView als Komposition von TableViewCell-Instanzen zusammen. Der TableView wird zu Beginn seiner Initialisierung mit einem TableViewController bekanntgemacht. Hierbei handelt es sich um eine Kindklasse des UIViewController, jedoch werden zwei zusätzliche Protokolle implementiert: TableViewDataSource und TableViewDelegate. Das TableViewDataSource-Protokoll definiert eine Schnittstelle von Methoden, die im TableViewController implementiert werden, um eine Tabelle mit Daten zu füllen. Eine zentrale Aufgabe für den TableViewController ist es daher die TableViewCell-Instanzen zur erzeugen und mit Inhalten zu füllen. Das TableViewDelegate-Protokoll definiert Methoden, die vom TableView bei seinem Delegate-Objekt aufgerufen werden, sobald ein bestimmtes Ereignis eintritt. Ein solches Ereignis könnte die Auswahl einer Zeile (bzw. Zelle) sein. In Folge dessen könnte der TableViewController einen weiteren ViewController auf den Navigations-Stack legen, um eine Navigationshierarchie aufzubauen.

some TableViewDataSource

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section {
      if(section == 0) {
            return 3;
      }
    return 0;
}
```

Eine Tabelle wird im TableView in sog. Sections und Rows gegliedert. Dabei stellt eine Section das Oberelement zu einer Row dar. In der Adressbuchanwendung gibt es für jeden vorhandenen unterschiedlichen Anfangsbuchstaben eines Kontaktnamens eine eigene Section. Innerhalb einer solchen Section wird die Anzahl der Rows dadurch bestimmt, wie viele Namen mit dem zur Section gehörenden Buchstaben beginnen.
Je nach Anwendungsfall kann die Implementierung der numberOfSections bzw. numberOfRowsInSection-Methoden unterschiedlich komplex sein: entweder werden die Rückgabewerte fest in den Programmcode geschrieben (wie auf der Folie) oder durch Abfragen einer Datenbank errechnet (wie z.B. bei der Adressbuchanwendung).

some TableViewDelegate

```
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:
(NSIndexPath *)indexPath {
    NSLog(@"did select row %i", indexPath.row);
}
```

Das TableViewDelegate Protokoll ist für die Handhabung der Interaktion mit der Tabelle verantwortlich. Dabei delegiert der TableView das Ereignis “Zelle ausgewählt” durch einen didSelectRowAtIndexPath-Methodenaufruf an den TableViewController. Dieser gibt im Beispielcode der Folie die Nummer der ausgewählten Zeile in den Log-Daten aus.

wie sieht's dann aus?

plain

grouped

TableViews besitzen zwei verschiedene optische Stile: "plain" und "grouped". Der "plain"-Stil zeichnet sich dadurch aus, dass die Tabellenzellen rechteckig sind und die vollständige Fläche des TableView einnehmen. Der "grouped"-Stil besitzt abgerundete Zellen, wobei die Rundungen immer um eine einzelne "Section" gelegt sind. Daher lassen sich zusammenhängende Zellen leichter erkennen.

Einsatzgebiete?

- kurz: überall!
- Kontakte, Einstellungen, Mail, iPod, Kalender, ...
- Kombination mit NavigationController
- zur Implementierung von Navigationshierarchien geeignet

Da TableViews sehr vielseitig sind, werden sie überall eingesetzt. Nahezu jede der Standardanwendungen, die mit dem iPhone ausgeliefert werden, verwendet TableViews entweder zur Darstellung von Entitäten bzw. Daten oder als Menu. Beim Einsatz als Menu tauchen die TableViews bzw. TableViewController häufig in Zusammenspiel mit dem NavigationController auf.

Einsatzgebiete?

- Beispiel: Mensa App
- TableView: Wähle den Tag
- TableView: Wähle das Gericht
- TableView: Betrachte die E-Zusatzstoffe, Brennwert und ein Foto des Gerichts

Dieses Zusammenspiel von TableView und NavigationController lässt sich an einer Mensa-App verdeutlichen. Beim Anwendungsstart wird in einer Tabelle mit 5 Zeilen eine Auswahl des Tages ermöglicht, für den man den Speiseplan abrufen möchte. Wird eine Zeile ausgewählt, wird ein weiterer TableViewController mit zugehörigem TableView auf den Navigationsstapel gelegt. In der neuen Ansicht kann eines der Gerichte ausgewählt werden. Nach Auswahl eines Gerichts, wird in einem weiteren TableView das Gericht mit Foto und detailierten Informationen vorgestellt.
Dabei fällt auf, dass etwa die Anzahl der Tage konstant ist und somit fest kodiert werden kann. Die Anzahl der verfügbaren Gerichte kann jedoch schwanken. Daher ist es sinnvoll in der Gerichttabelle eine dynamische Implementierung der numberOfRowsInSection-Methode zu verwenden. Bei der Detailansicht des Gerichts bietet sich die Verwendung eines grouped-Stils an, da sich so zusammenhängende Informationen wie etwa Zutaten von anderen Daten wie etwa Preisen abgrenzen lassen.

Look of TableView

- Plain & Grouped
- Style kann nicht zur Laufzeit geändert werden!
- TableViewCells können mit Subviews “aufgemotzt” werden

Bei der Gestaltung eines TableView ist es wichtig zu beachten, dass der Stil nach der Instanziierung des TableView nicht mehr verändert werden kann. D.h. wird ein TableView im InterfaceBuilder erzeugt, so muss auch dort bereits der später zu verwendende Stil gewählt sein. Die einzige Veränderung der TableViews kann nach der Instanziierung nur noch durch Anpassung der TableViewCells erfolgen. Diesen Views können weitere Views mit der addSubview-Methode angehangen werden.

Customizing Cells

- verschiedene Möglichkeiten:
  - im TableViewController Code Subviews erstellen und anhängen
  - eigene XIB Resource im Interface Builder erstellen
  - eigene Klasse schreiben

Die Gestaltung der Tabellenzellen ist jedoch nicht auf eine Herangehensweise limitiert. So können die TableViewCells wie bereits auf der vorigen Folie erwähnt mittels Programmcode erzeugt und erweitert werden. Eine weitere Möglichkeit ist durch den InterfaceBuilder gegeben. Hier können für die Tabellenzellen eigene XIB-Dateien angelegt werden und die dort platzierten Views über IBOutlets anderen Instanzen (idR. den ViewControllern) injeziert werden. Eine alternative auf Programmcode basierende Variante wäre das Implementieren einer eigenen TableViewCell-Kindklasse, die die zusätzlichen Views und Eigenschaften bereits bei der Instanziierung mitbringt.
Keine der aufgelisteten Varianten ist als “die Beste” zu bezeichnen. Eine Auswahl sollte an Hand der Wartbarkeit, des Aufwands und der Anforderungen erfolgen.

individuelle TableViewCells

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";

    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault reuseIdentifier:CellIdentifier]
autorelease];
    }

        cell.textLabel.text = [NSString stringWithFormat:@"row %i in section %i", indexPath.row, indexPath.section];
        int s = 20;
        UIView* someView = [[UIView alloc] initWithFrame:CGRectMake(tableView.frame.size.width - s - 20,
(tableView.rowHeight-s)/2, s, s)];
        if (indexPath.row == 0) {
                someView.backgroundColor = [UIColor redColor];
        } else if (indexPath.row == 1) {
                someView.backgroundColor = [UIColor greenColor];
        } else if (indexPath.row == 2) {
                someView.backgroundColor = [UIColor blueColor];
        }

        [cell addSubview:someView];
        [someView release];

    return cell;
}
```

Diese Folie zeigt beispielhaft die erste Möglichkeit individuelle TableViewCells zu erzeugen (d.h. durch Programmcode nach der Instanziierung zusätzliche Subviews anhängen). Bei komplexen Individualisierungen macht es Sinn, den Programmcode der “Gestaltung” in eine eigene Klasse auszulagern, um sog. “Gottklassen” zu vermeiden. Bei einer Gottklasse handelt es sich um eine Klasse, die zu viele Funktionen auf ein Mal erfüllt und besser in einzelne voneinander losgelöste Klassen aufgeteilt werden sollte.

individuelle TableViewCells

Auf dieser Folie ist das Ergebnis des vorangehenden Beispielcodes zu sehen.

Zusammenfassung

- **DataSource** und **TableViewDelegate** werden vom TableViewController implementiert
- TableView ist **die** Komponente
- **TableViewCell** lässt sich erweitern mit allen UIView-Subklassen

Bei der Verwendung von TableViews ist zu beachten, dass die DataSource- und TableViewDelegate-Protokolle in der eigenen TableViewController-Klasse implementiert werden müssen.
Da der TableView sehr flexibel und vielseitig (Skalierbarkeit bei Großen Datenmengen, visuelle Darstellung, Erweiterbarkeit der Tabellenzellen) ist, stellt er eine der zentralsten Komponenten des SDK dar.
Zur Anpassung von TableViews an eigene Anforderungen sollte auf eine Individualisierung der TableViewCells zurückgegriffen werden.

Aufgabe

- Baut eine Anwendung, die...
  - in einem TableView im Laufe des Workshops eure erstellten ViewController auflistet und
  - bei Auswahl einer Zelle mittels eines NavigationController einblendet

HTTP(S) Verbindung

HTTP(S) Verbindung

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren
2. Verbindung aufbauen

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren
2. Verbindung aufbauen
3. Verbindungsdaten lesen

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren

NSURLRequest

2. Verbindung aufbauen

3. Verbindungsdaten lesen

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren

NSURLRequest

2. Verbindung aufbauen

NSURLConnection

3. Verbindungsdaten lesen

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

1. Verbindung konfigurieren — **NSURLRequest**
2. Verbindung aufbauen — **NSURLConnection**
3. Verbindungsdaten lesen — **NSData**

Das Erstellen und die Handhabung einer HTTP(S) Verbindung ist erfreulich einfach...

HTTP(S) Verbindung

```
url = [NSURL URLWithString:@"www.wasIhrWünscht.de"];

request = [NSURLRequest requestWithURL:url];

connection = [NSURLConnection connectionWithRequest:request delegate:self];

[connection start]; // Für Objekte die euch gehören notwendig
```

das ist auch schon alles :)

NSURL erstellen, diese URL kann verschiedene Ziel haben, http:// , mail:// , local:// ...
Die URL wird jetzt einen einen NSURLRequest übergeben, der in die NSURLConnection eingefügt wird.
WICHTIG: beim erstellen der Connection wird der Delegate gesetzt! Eine Connection bleibt im Speicher aktiv und „findet zur Klasse zurück“ auch wenn die Connection NICHT retaint wird!
-> hohes Fehlerpotenzial (Absturz wenn die Klasse nicht mehr im Speicher liegt)

(die Connection muss nicht mehr expliziet gestartet werden, war in älteren SDK-Versionen noch üblich)

HTTP(S) Verbindung

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data_{
    if(data == nil){
        data = [[NSMutableData alloc] initWithData:data_];
    }
    else{
        [data appendData:data_];
    }
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection_{ ... }
```

Beim Empfangen der Daten ist es wichtig zu wissen das nicht immer alle Daten auf einmal zur Verfügung stehen, sondern sie können „reintröpfeln“. Daher hängen wir Daten an.

HTTP(S) Verbindung

- Wichtig:
  Verbindung **abbrechen** wenn ihr den ViewController verlasst oder wenn ein Fehler eintritt.

```
[connection cancel];
connection = nil;
```

Wichtig wichtig: Abbrechen im Fehlerfall oder beim Verlassen des ViewControllers! (siehe Problem Folie-- -- :)

Aufgabe

- Erstellt eine Twitter-App mit der ihr die Nachrichten aus verschiedenen Accounts lesen könnt.
- http://api.twitter.com/1/statuses/user_timeline/%@.xml

iOS 4.0 Features

iOS 4.0 Features

- Modern Runtime
- App Switching
- Gestures
- (einige mehr)

Modern Runtime – inteligentes Konzept zur Speicherverwaltung
App Switching – wird als „multitasking“ verkauft... (Ein Prozessor kann NUR eine Sache berechnen!)
Gestures – Erlauben den Zugriff auf die Touchgesten von Apple

Modern Runtime

Modern Runtime

- „Intelligentes“ Speichermanagmentkonzept
- Garbage Collector = foo (und langsam)
- Idee: kein Garbage entstehen lassen

  ➡ Es gibt keine Objekte mehr!

Die Idee hinter der modern Runtime ist die, das der Entwickler einfach ohne Objekte entwickelt! Ohne Objekte kann ein Entwickler auch keine Objekte Speicher „leaken“ lassen. Der gesamte Zugriff wird nur noch durch getter und setter durchgeführt, die NICHT selbst geschrieben werden sollten. (Das Schreiben von gettern und settern ist NICHT TRIVIAL unter ObjC)

Modern Runtime

- Zugriff wird nur noch über Getter und Setter gestattet
- Getter und Setter „wissen“ was sie machen
- Lassen sich klar und einfach definieren
- Lassen sich klar und einfach überschreiben

Modern Runtime

- @property (x, ...) id obj
  - nonatomic
  - retain
  - assign
  - copy
  - ...
  - readonly
  - readwrite
- @synthesize obj

nonatomic – der Zugriff auf die Variable wird als nicht-atomarer Prozess betrachtet, dieses macht den Zugriff ~100 mal schneller als ein atomarer Zugriff
retain – Das erstellte Objekt wird mit einem „retain“ versehen
assign – Das erstellte Objekt wird ohne „retain“ versehen
copy – wie retain, der getter erstellt immer Kopien. (Das ist zum Beispiel bei NSMutableString sehr hilfreich)

freundliche Beschreiber...
readonly – nur Lesezugriff
readwrite – Lese- Schreibzugriff

Modern Runtime

- @property (nonatomic, assign, readwrite) id obj;
- @synthesize obj;

Und nun?

Erstellen einer property und deren Umsetzung. Wir definieren eine schwache Referenz auf den Datentyp id.

Modern Runtime

➡ self.obj

Von nun an arbeiten wir nur noch auf den gettern und settern mit **self**

Modern Runtime

- Beispiel

Modern Runtime

- Ausblick
  - Sehr gut geeignet um das „M“ im MVC Modell umzusetzen
  - Zugriff auf properties läßt sich beobachten mittels Key-Value-Observing

Durch die Bestimmbarkeit der properties lassen sich sehr gut Komponenten abtrennen. So sollten nur Elementare Modeldaten im .h gesetzt werden können, die von der View unabhängig sind. Perfekt für MVC!

Sollte auf die Eigenschaften der properties gelauscht werden wollen muss Key-Value-Observing betrieben werden.

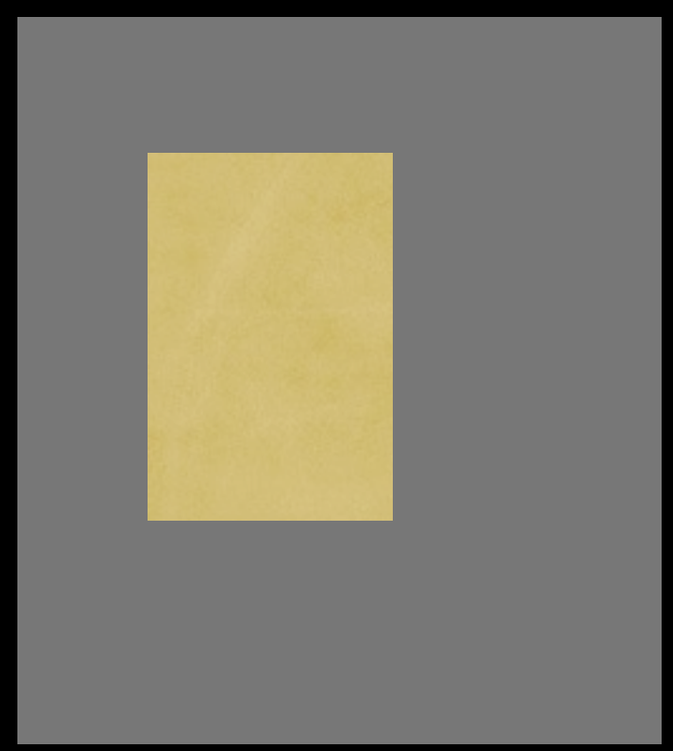
App Swiching

- Erlaubt es euere Anwendung im Hintergrund weiter zu laufen. (Wirklich ?)
- Erlaubt schnellen Wechsel zwischen den Apps.
- App Switching macht Apps schreiben schwer.

Gestures + UIScrollView

UIScrollView

- Die am häufigsten Verwendete Klasse bei Apple
- View + Scrolling
- View + Content



Die UIScrollView ist die Klasse, die für die meisten Anwendungen benötigt wird. Das Springboard, jeder TableView, iTunes, die PhotoApp, fast jede Anwendung basiert auf dem UIScrollview.

Der gelbe Bereich stellt den Bildschirminhalt dar, der graue den Inhalt. Über den frame (CGRect) wird der gelbe Bereich konfiguriert, über die contentSize (CGSize) der graue

Gestures

- Dienen dazu vorhandene, gelernte Interaktionsparadigmen wieder zu verwenden
- Ersparen das Handling mit UITouch

Benutzer erwarten das sich eine Anwendung verhält wie der Rest. Dieses kann nur gewährleistet werden, wenn die Eigenschaften der Erkennung für alle Anwendungen auf einem Gerät identisch sind.

Gestures

- Leiten von UIGestureRecognizer ab.
- Es gibt für jede Geste einen „Recognizer“
  - UITapGestureRecognizer
  - UISwipeGestureRecognizer
  - UIRotationGestureRecognizer
  - UILongTapGestureRecognizer
  - ...
- Eigene Klassen die von UIGestureRecognozer ableiten sind erlaubt und erwünscht!

Die Klasse UIGestureRecognizer ist abstrakt, diese Klasse NIE instanziieren. Es gibt eine Reihe von Recognizern die fast alle Fälle abdecken, sollte ein eigene Recognizer geschrieben werden wollen bitte in der Apple Doku nachschauen welche Methoden alles überschrieben werden müssen aus UIResponder, UITouch, UIView.

WICHTIG: der Recognizer wird intern als Moore-Automat abgebildet, diesen vorher modellieren und gedanklich durchtesten!!!

Gestures

- Werden immer an eine View gebunden
- Mehrere Recognizer können an eine View gebunden werden
- Jeder Recognizer kann mehrere Targets besitzen
- Sind erfreulich angenehm zu benutzen :)

Benutzung: siehe nächste Folie

Gestures

```
UIPinchGestureRecognizer* pincher =
[[UIPinchGestureRecognizer alloc] initWithTarget:self
action:@selector(zoomGreenView:)];

[viewToZoom addGestureRecognizer:pincher];
```

```
- (void)zoomGreenView:(UIPinchGestureRecognizer*)gesture{
   ...
   float value = gesture.scale;
   ...
}
```

1) Erstellen eines Recognizers
2) Recognizer hinzufügen
3) Werte vom Recognizer auslesen

Aufgabe

1. Setzt einen GestureRecognizer und gebt dessen Werte als NSLog() in der Konsole aus.
2. Fügt ein Bild (UIIMageView mittels IB) und eine UIScrollView in die Anwendung ein
3. Implementiert die UIScrollViewDelegate Methode um das Bild zu skalieren.

Instruments

Instruments

- dynamische Betrachtung des Kompilates
- mächtiges Werkzeug (gut wie auch schlecht)
- Fertige Instrumente
- Eigene Instrumente
- Schlecht dokumentiert

* Shark dient zur Betrachtung des Laufzeitverhaltens, der Umgang mit Shark ist nicht trivial, darauf wird in diesem Kurs nicht eingegangen.

Instruments

- Instrumente
  - Verändern nicht den Code
  - Müssen nicht in den Code eingefügt werden
  - Verändern die Laufzeit des Systems
  - Laufen direkt im Kernel ab (!)
  - Ausnahme: PL_NOATTACH

es können alle Programme beobachtet werden mit Ausnahme von iTunes. (Sicherheitsgründe)

Instruments

- Fertige Instrumente:
  - Speicherverhalten
  - Laufzeitverhalten
  - Datenzugriffe
  - OpenGL
  - ...

Instruments

- Eigene Instrumente
  - „DTrace“ probes
  - Werden in D geschrieben
  - provider:module:funktion:name
  - Beispiel 1: pid42:::enrty
  - Beispiel 2: objC:NSArray::entry

das Auslassen von Elementen entspricht dem „*“, sprich alles wird mitgenommen.

Instruments

- live Demo

On Device Testing

Problem

- Entwickler darf nicht “out of the box” auf einem Gerät eigenen Code ausführen!
- Apple will...
  - ... wissen, wer der Entwickler ist.
  - ... am Developer Program verdienen.
  - ... Nutzer vor schlechter Software schützen.

Wenn man eine Anwendung für das iPhone entwickelt, will man diese auch früher oder später auf dem Gerät testen. Dies ist vor allem dann sinnvoll, wenn es um Laufzeitverhalten geht oder wenn Komponenten oder Programmbibliotheken verwendet werden, die nicht durch den iPhone Simulator unterstützt werden (wie z.B. GameKit oder die Fotokamera).
Um Anwender und die eigene Plattform zu schützen, erlaubt Apple es den Entwicklern nicht eigenen Anwendungen ohne Weiteres auf einem iPhone installieren. Apple möchte wissen, wer die Entwickler sind, und gleichzeitig an den Entwicklern verdienen. Eine Mitgliedschaft im Developer Program ist idR. kostenpflichtig. Ein Developer Account hingegen ist kostenlos und ermöglicht es lediglich das SDK herunterzuladen und Zugriff auf die Dokumentation zu bekommen.

Anforderungen

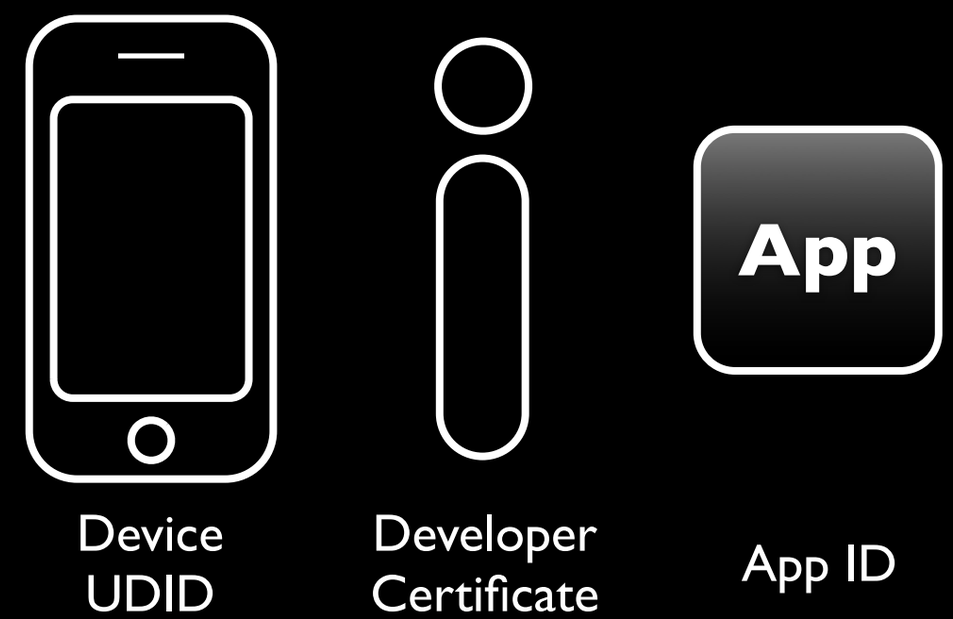
- ein iPhone, einen iPod touch oder ein iPad
- Apple Developer Account
- Mitglied im iPhone Developer Program (z.B. über die FH)

Um eine Anwendung auf einem iPhone (oder iPod touch bzw. iPad) zu testen, muss zunächst ein solches Gerät vorhanden sein. Der Simulator deckt nicht alle Funktionen und Verhaltensweisen eines echten Geräts ab!
Ferner muss der Entwickler einen Developer Account besitzen. Zwar lässt sich das iPhone SDK auch ohne einen solchen Account installieren, aber sobald eine eigens entwickelte Anwendung auf ein Gerät installiert werden soll, muss dieser Account Mitglied des iPhone Developer Program sein. Für Studenten der FH Köln Campus Gummersbach kann dies kostenlos über die FH erfolgen.

Ablauf

Der Ablauf des Testens auf einem Gerät lässt sich in drei Blöcke unterteilen. Zunächst erfolgt in Apple’s Developer Portal eine Zuordnung von Geräten, Entwicklern und Anwendungen. Hierbei sind sowohl die Entwickler als auch ein Administrator der FH involviert. Der Entwickler sammelt die UDIDs der Geräte, auf denen die Anwendung installiert werden soll. Außerdem stellt der Entwickler eine Zertifikatanfrage. Das später erhaltene Zertifikat dient zur Identifikation des Entwicklers. Schlussendlich muss von der Hochschule eine App ID im Developer Portal eingetragen und dieser die Entwickler und Geräte zugeordnet werden. Ein Entwickler kann nun im Developer Portal ein sog. Provisioning Profile herunterladen und mit XCode auf ein Gerät kopieren. Um nun die Anwendung auf einem Gerät zu installieren wird in XCode als Zielgerät nicht der Simulator sondern das iPhone ausgewählt. XCode erkennt das Entwicklerzertifikat und signiert den Code. Da auf dem Gerät ein Provisioning Profile vorhanden ist, kann das Gerät sicherstellen, dass der Entwickler die Anwendungen installiert hat und diese ausführen darf.

Ablauf

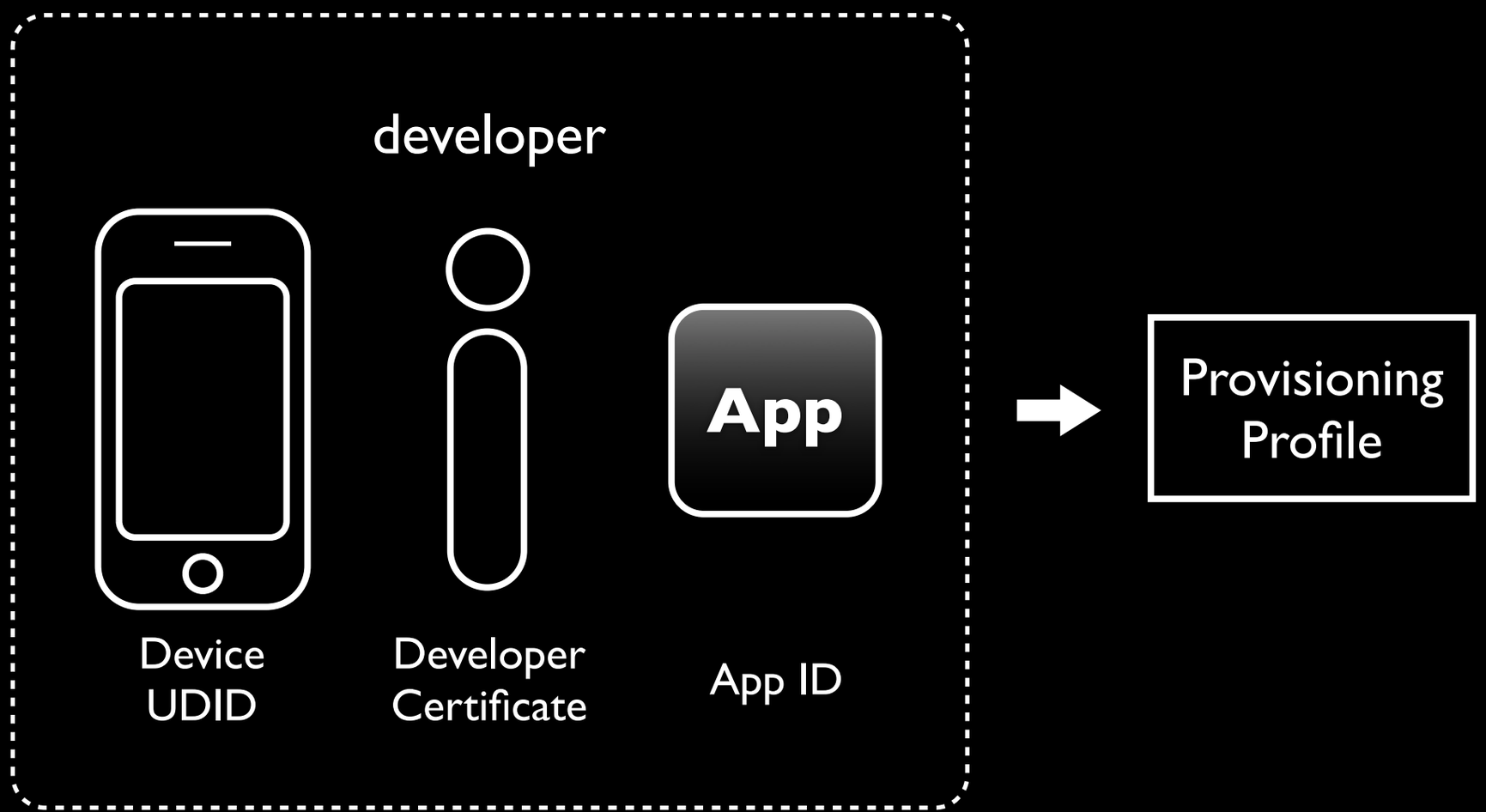


Der Ablauf des Testens auf einem Gerät lässt sich in drei Blöcke unterteilen. Zunächst erfolgt in Apple's Developer Portal eine Zuordnung von Geräten, Entwicklern und Anwendungen. Hierbei sind sowohl die Entwickler als auch ein Administrator der FH involviert. Der Entwickler sammelt die UDIDs der Geräte, auf denen die Anwendung installiert werden soll. Außerdem stellt der Entwickler eine Zertifikatanfrage. Das später erhaltene Zertifikat dient zur Identifikation des Entwicklers. Schlussendlich muss von der Hochschule eine App ID im Developer Portal eingetragen und dieser die Entwickler und Geräte zugeordnet werden. Ein Entwickler kann nun im Developer Portal ein sog. Provisioning Profile herunterladen und mit XCode auf ein Gerät kopieren. Um nun die Anwendung auf einem Gerät zu installieren wird in XCode als Zielgerät nicht der Simulator sondern das iPhone ausgewählt. XCode erkennt das Entwicklerzertifikat und signiert den Code. Da auf dem Gerät ein Provisioning Profile vorhanden ist, kann das Gerät sicherstellen, dass der Entwickler die Anwendungen installiert hat und diese ausführen darf.

Ablauf

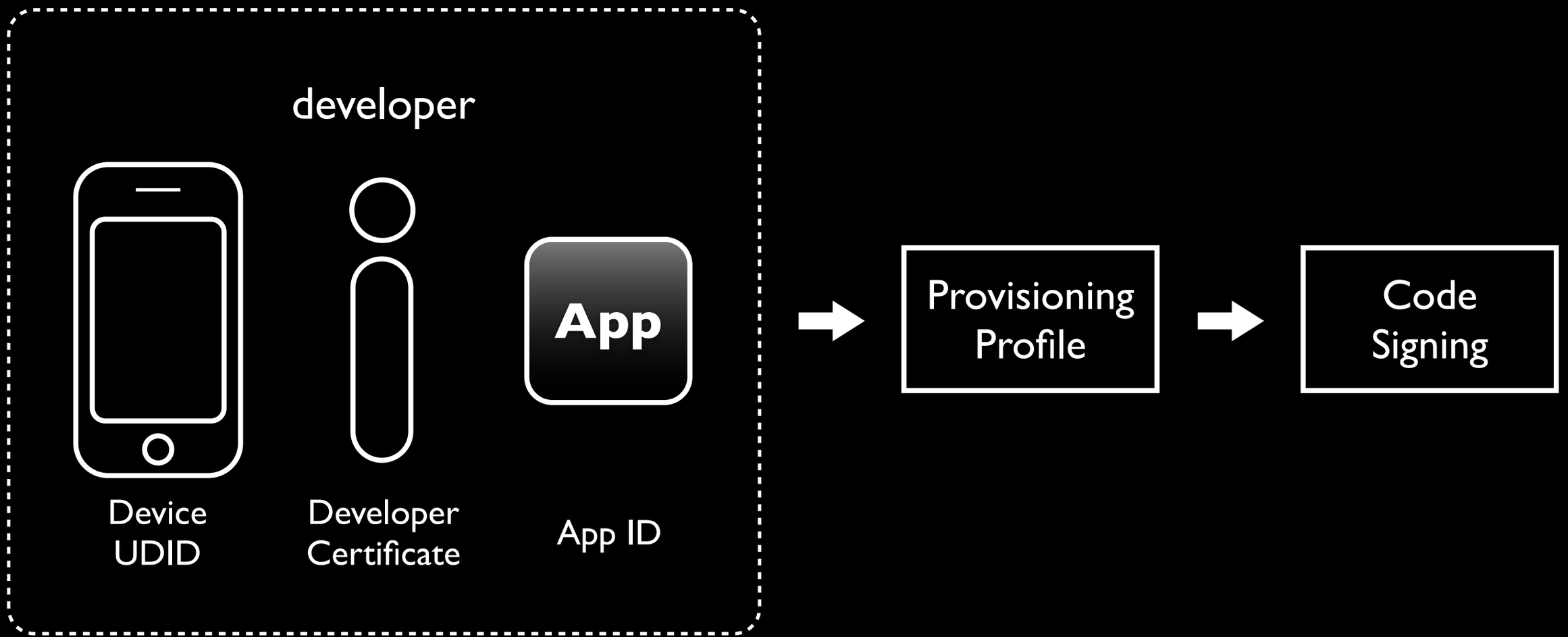
Der Ablauf des Testens auf einem Gerät lässt sich in drei Blöcke unterteilen. Zunächst erfolgt in Apple's Developer Portal eine Zuordnung von Geräten, Entwicklern und Anwendungen. Hierbei sind sowohl die Entwickler als auch ein Administrator der FH involviert. Der Entwickler sammelt die UDIDs der Geräte, auf denen die Anwendung installiert werden soll. Außerdem stellt der Entwickler eine Zertifikatanfrage. Das später erhaltene Zertifikat dient zur Identifikation des Entwicklers. Schlussendlich muss von der Hochschule eine App ID im Developer Portal eingetragen und dieser die Entwickler und Geräte zugeordnet werden. Ein Entwickler kann nun im Developer Portal ein sog. Provisioning Profile herunterladen und mit XCode auf ein Gerät kopieren. Um nun die Anwendung auf einem Gerät zu installieren wird in XCode als Zielgerät nicht der Simulator sondern das iPhone ausgewählt. XCode erkennt das Entwicklerzertifikat und signiert den Code. Da auf dem Gerät ein Provisioning Profile vorhanden ist, kann das Gerät sicherstellen, dass der Entwickler die Anwendungen installiert hat und diese ausführen darf.

Ablauf



Der Ablauf des Testens auf einem Gerät lässt sich in drei Blöcke unterteilen. Zunächst erfolgt in Apple's Developer Portal eine Zuordnung von Geräten, Entwicklern und Anwendungen. Hierbei sind sowohl die Entwickler als auch ein Administrator der FH involviert. Der Entwickler sammelt die UDIDs der Geräte, auf denen die Anwendung installiert werden soll. Außerdem stellt der Entwickler eine Zertifikatanfrage. Das später erhaltene Zertifikat dient zur Identifikation des Entwicklers. Schlussendlich muss von der Hochschule eine App ID im Developer Portal eingetragen und dieser die Entwickler und Geräte zugeordnet werden. Ein Entwickler kann nun im Developer Portal ein sog. Provisioning Profile herunterladen und mit XCode auf ein Gerät kopieren. Um nun die Anwendung auf einem Gerät zu installieren wird in XCode als Zielgerät nicht der Simulator sondern das iPhone ausgewählt. XCode erkennt das Entwicklerzertifikat und signiert den Code. Da auf dem Gerät ein Provisioning Profile vorhanden ist, kann das Gerät sicherstellen, dass der Entwickler die Anwendungen installiert hat und diese ausführen darf.

Ablauf



Der Ablauf des Testens auf einem Gerät lässt sich in drei Blöcke unterteilen. Zunächst erfolgt in Apple's Developer Portal eine Zuordnung von Geräten, Entwicklern und Anwendungen. Hierbei sind sowohl die Entwickler als auch ein Administrator der FH involviert. Der Entwickler sammelt die UDIDs der Geräte, auf denen die Anwendung installiert werden soll. Außerdem stellt der Entwickler eine Zertifikatanfrage. Das später erhaltene Zertifikat dient zur Identifikation des Entwicklers. Schlussendlich muss von der Hochschule eine App ID im Developer Portal eingetragen und dieser die Entwickler und Geräte zugeordnet werden. Ein Entwickler kann nun im Developer Portal ein sog. Provisioning Profile herunterladen und mit XCode auf ein Gerät kopieren. Um nun die Anwendung auf einem Gerät zu installieren wird in XCode als Zielgerät nicht der Simulator sondern das iPhone ausgewählt. XCode erkennt das Entwicklerzertifikat und signiert den Code. Da auf dem Gerät ein Provisioning Profile vorhanden ist, kann das Gerät sicherstellen, dass der Entwickler die Anwendungen installiert hat und diese ausführen darf.

Was müsstet ihr tun?

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***
5. *Entwickler (Zertifikat), Geräte (UDID) in die App (App ID) aufnehmen lassen **(FH)***

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***
5. *Entwickler (Zertifikat), Geräte (UDID) in die App (App ID) aufnehmen lassen **(FH)***
6. generiertes Provisioning Profile herunterladen

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***
5. *Entwickler (Zertifikat), Geräte (UDID) in die App (App ID) aufnehmen lassen **(FH)***
6. generiertes Provisioning Profile herunterladen
7. Profile auf das Gerät installieren

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***
5. *Entwickler (Zertifikat), Geräte (UDID) in die App (App ID) aufnehmen lassen **(FH)***
6. generiertes Provisioning Profile herunterladen
7. Profile auf das Gerät installieren
8. im Xcode Projekt den App Identifier für das Build Target setzen

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

Was müsstet ihr tun?

1. Entwicklerzertifikat anfordern
2. Zertifikat in die “KeyChain” des Entwicklerrechners aufnehmen
3. *Gerät UDID im Developer Program eintragen lassen **(FH)***
4. *App ID im Developer Program anlegen lassen **(FH)***
5. *Entwickler (Zertifikat), Geräte (UDID) in die App (App ID) aufnehmen lassen **(FH)***
6. generiertes Provisioning Profile herunterladen
7. Profile auf das Gerät installieren
8. im Xcode Projekt den App Identifier für das Build Target setzen
9. hoffentlich und endlich auf dem Gerät testen

Diese Liste soll als Übersicht über die notwendigen Schritte für das Testen einer Anwendung auf einem Gerät dienen. In kursiver Formatierung sind die Schritte hervorgehoben, in denen durch die FH von Administratoren Aktionen durchgeführt werden müssen.

UIViewController + NSUserDefaults

UIViewController View States

- Dienen zum überwachen des Views
- Vier Zustände:
  - viewWillAppear
  - viewDidAppear
  - viewWillDisappear
  - viewDidDisappear
- Ideal als Einstiegspunkt für Anpassungen an der Benutzungspberfläche

UserDefaults

- Dient zum Laden und Speichern von Daten
- Dient zur Steigerung des Benutzungserlebnisses
- Jedes Programm besitzt „eigene“ User Defaults

UserDefaults

- Speichert:
  - Float, Double, Int, Bool und URL‘s
  - Objekte
    (Die vom Typ NSString, NSNumber, NSData, NSDate,
    NSArray oder NSDictonionary sind oder davon ableiten)
  - Objekte sind „immutable“

Objekte aus den UserDefaults sind immutable, auch wenn man Mutable-Objekte reingesteckt hat. Einfachster Weg das zu lösen: z.B. bei einem Array:
NSMutableArray* array = [NSMutableArray arrayWithArray:arrayAusDefaults];

UserDefaults

```objc
// NSUserDefaults referenzieren
NSUserDefaults* defaults = [NSUserDefaults standardUserDefaults];

// schreiben
[defaults setDouble:42.0 forKey:@"AntwortFürAlles"];

// lesen
double value = [defaults doubleForKey:@"AntwortFürAlles"];
```

die Werte werden alle 10 Sekunden in die Defaults geschrieben, wenn die Anwendung vorher ausgeschalten wird sind diese Werte weg.
Ein Speichern kann jedoch mich [defaults synchronize] erzwungen werden.

Aufgabe

- Erweitert die Rotationsaufgabe

  1. Implementiert die viewWillAppear Methode (überprüft diese mit einem NSLog())
  2. Schreibt die Werte für Rot, Grün und Blau in die NSUserDefaults
  3. Lest die Defaults wieder aus wenn die Methode viewWillAppear geladen wird

Tipps...!

(... und **Wissen** von uns an euch)

- Speichermanagement 1: erst bekommt **Apple** Speicher, dann ihr

Ihr bekommt eine Speicherwarnung, auch wenn z.B. der Safari 80 MB Ram belegt. Erste Speicherwarnungen können euer Programm bereits nach 2 MB erreichen.

- Speichermanagement II:
„Multitasking“ heißt, die Anwendung die mehr Speicher verbraucht fliegt raus.

Ihr bekommt eine Speicherwarnung, auch wenn z.B. der Safari 80 MB Ram belegt. Erste Speicherwarnungen können euer Programm bereits nach 2 MB erreichen.

- Threadmonster: NSXMLParser und der UITableView ->

One Thread eats all

Das Benutzen dieser Objekte kann Timeouts produzieren. Wenn Sie aktiv sind „steht“ das System.

- Eigene Schriftarten: gehen, aber nicht von Apple aus... z.B. ***FontLabel***

- Rotation: ist leicht, sie **erzwingen** schwer

es ist nicht leicht während der Navigation eine andere Ausrichtung zu erzwingen. (Es geht, aber dazu muss man den AppDelegate informieren, recht aufwendig)

- InterfaceBuilder **fetzt**!

- Absturz ohne jede Meldung:

Häufig **Formatierungsfehler**

also %i %f %@ %d usw werden falsch verwendet

- Keine Änderungen im UI:

Clean or **CleanAll**

- Man kann auf das Adressbuch **zugreifen**

- Man hat **keinen** Zugriff auf den Kalender

- Photos: werden immer **fertig** geliefert

es ist nicht (ohne Aufwand) möglich auf die Pixeldaten zuzugreifen.

- Accelerometer: nicht langsamer als **10Hz** (und nicht schneller als **300hz**)

- Simulator und Gerät sind **unterschiedlich**

- **nil eats messages**
  Die Umgebungsvariable NSZombiesEnabled kann beim Finden solcher Fehler helfen

NSZombieEnabled should not be left in place permanently, as by default no objects will ever be truly deallocated, making your application use tremendous amounts of memory.

- NSLog() machts **langsam**