Hallo.

Wer sind wir?

- Tim Haussmann
- Daniel Bertram

Wer seid ihr?

Ablauf der Veranstaltung

- Beginn 10 Uhr
- Anwesenheit erforderlich für das Zertifikat
- Wer fragt gewinnt
- Helft uns mal :)

Objective C

!= Cocoa

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

Allgemeine Features

- 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

- 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

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

MyClass.h

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

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

@end
```

MyClass.m

```objc
#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)

Messaging

```
[string length];
[string characterAtIndex:21];
[string getCharacters:aBuffer range:aRange];
[string stringByAppendingString:[string lowercaseString]];
```

VORSICHT: Es können beliebige Nachrichten an jede Instanz gesendet werden. Der Compiler warnt, aber einen Fehler gibt es erst zur Laufzeit.

```
[string thisMessageDoesNotExist];
```

message und andArgument sind beliebig
Selector: „message“
Argument(e):
Message: message:argument

Dot Syntax

- Neu seit ObjectiveC 2.0
- Syntactic sugar

```
int age = [person age];
    age = person.age;
```

```
[person setAge:21];
person.age = 21;
```

```
int age = [[person child] age];
    age = person.child.age;
```

Es ist üblich in ObjectiveC die Getter ohne „get“ zu benennen
Wird vom Compiler in einen „normalen Aufruf“ transformiert

- Vorsicht mit der Dot Syntax!

```
@interface Person : NSObject {
  NSString* name;
}
-(void)someMethod;
-(void)setName:(NSString*)name;
```

```
-(void)someMethod{
  self.name = @"Cocoa"; //The same as [self setName:@"Cocoa"];
  name = @"Cocoa"; // NOT the same as [self setName:@"Cocoa"];
}
```

„Konstruktoren“

- Erzeugen von Instanzen:

```
NSString* s1 = [[NSString alloc] init];
NSString* s2 = [[NSString alloc] initWithContentsOfURL:aURL];

NSString* s3 = [NSString string];
NSString* s4 = @"Hello Cocoa";
```

- Implementierung

```
-(id)init{
    if(self = [super init]){
       //custom initialization
    }
    return self;
}
```

@“string“ definiert ein konstantes NSString Objekt und setzt seinen Inhalt auf „string“

Schnittstellen

- Schnittstellen in ObjC heissen: „Protocol“
- Vgl. Java: Interface
- Nur Definition von Methoden sind erlaubt

MyProtocol.h

```
@protocol MyProtocol
  -(BOOL)doSomethingWithString:(NSString*)string;
@end
```

MyClass.h

```
#import <Foundation/Foundation.h>
@interface MyClass : NSObject <MyProtocol>{
}
@end
```

MyClass.m

```
#import "MyClass.h"

@implementation MyClass
  -(BOOL)doSomethingWithString:(NSString*)string{
     return YES;
  }
@end
```

Null Objekt

- Null object pointer: nil

```
if(father == nil) return;
   if(!father) return;
```

```
[father setChild:nil];
```

!!! nil eats messages !!!

```
father = nil;
[father child];
```

Nil eats messages

Sonstiges

- Datentypen wie in C, C++(und Java)
- Sprachkonstrukte (C, C++)
  - if, if-else, switch, while, while-do, break, continue, goto, return, for, ...

Weitere Datentypen

- Selektoren

```
SEL selector = @selector(myMehod1);
    selector = @selector(myMehod2:);
    selector = @selector(myMehod3:withArg:);
```

- BOOL

```
BOOL boolean;
boolean = YES;
boolean = NO;
if(boolean){
    return @"YES";
}
return @"NO";
```

Selektoren haben 2 Bedeutungen:
1) Der Name einer Methode
2) Die eindeutige ID, die eine Methode nach dem Kompilieren zugewiesen bekommt
Zeigen auf eine Methode mit dem Namen ..., eine spezielle Klasse gehört nicht dazu

Type of BOOL is char
YES := (BOOL)1
NO := (BOOL)0

Dynamic vs. static typing

- Static typing
  - „Überprüfung“ durch Compiler
- Dynamic typing
  - Keine „Überprüfung“ durch Compiler

```
id dynString;                //dynamic typing
NSString* staticString;  //static typing
```

id ist Zeiger auf beliebige Klasse, NSObject ist eine davon, NSObject ist die Basis auf der Cocoa aufbaut (Foundation Classes). Es ist aber auch möglich Klassen zu schreiben, die nicht von NSObject abgeleitet sind, diese sind vom Typ id aber nicht vom Typ NSObject.

id kann gefährlich sein

Logging & FormatStrings

- NSLog()

```
NSLog(@"My message");

NSString* s = @"Hello Cocoa";
NSLog(@"Content: %@   Length: %i", s, s.length);
```

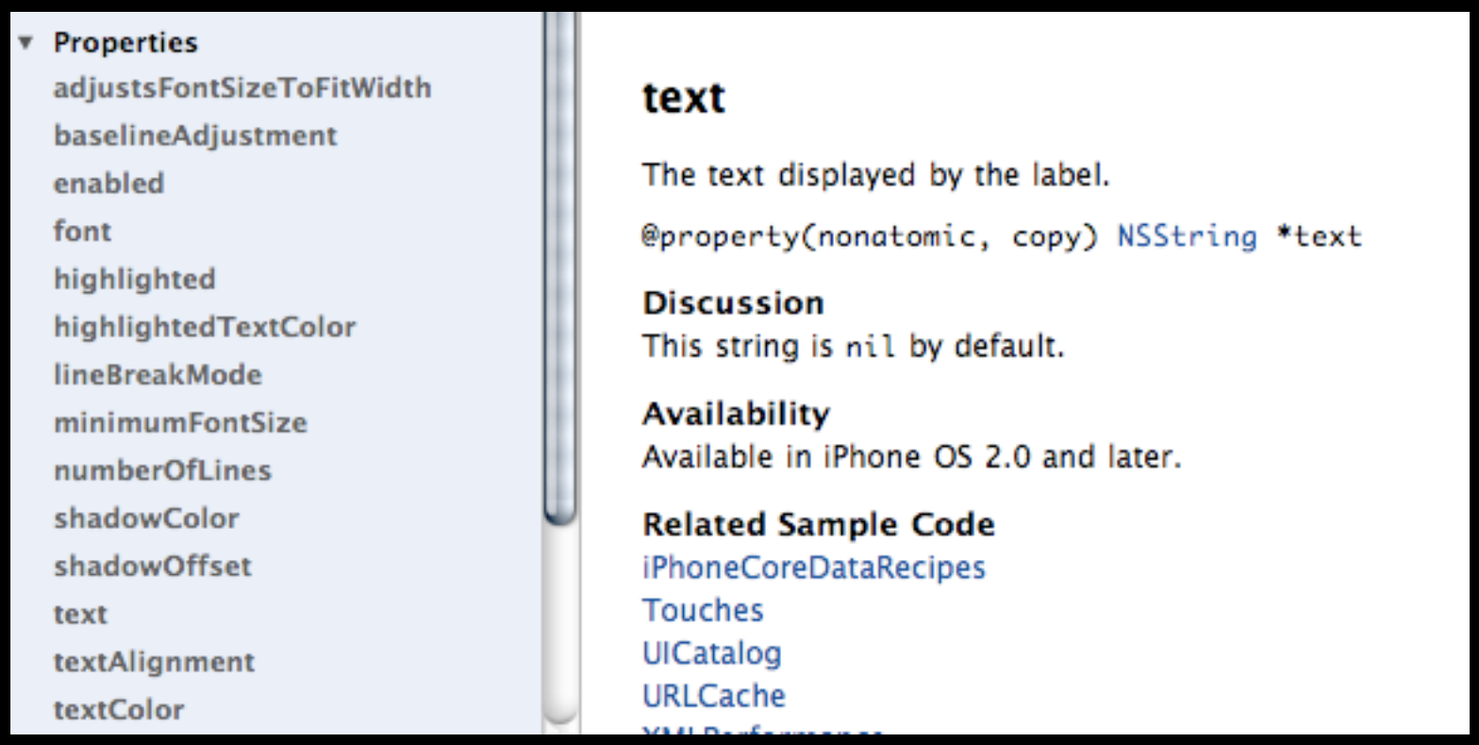
- NSString

```
NSString* s2 = [NSString stringWithFormat:@"Content: %@", s];
```

%@ Objective-C object, printed as the string returned by `descriptionWithLocale:` if available, or `description` otherwise.

Properties

Properties



Properties sind ein wichtiger Bestandteil im Cocoa-Framework. Sie definieren wie auf Eigenschaften zugegriffen werden kann. Weiterhin nehmen sie einem Entwickler auch den Großteil der Speicherverwaltung ab, dazu aber mehr im Advanced-Kurs.

Properties

- Sind Getter und Setter
- Werden automatisch mittels @synthesize generiert oder
- Werden vom Entwickler selbst geschrieben

Properties können als Getter und Setter angesehen werden, nur das sie viel mehr machen als bei Java. In diesem Kurs können wir nicht detailierter darauf eingehen.

Wichtig ist: in der .h Datei werden sie als @propery definiert und in der .m Datei synthetisiert. (@synthesize)

Properties

- Können verschiedene Attribute besitzen:
  - nonatomic
  - retain
  - assign
  - copy
  - readonly
  - ...

```
@property (nonatomic, assign) float redValue;
```

Wichtige Attribute:
* nonatomic: setzt das Objekt oder die Variable in einen Thread-unabhängigen Zustand. Dieses Attribut sollte nur BEWUSST weggelassen werden wenn Thread-bedingte Eigenschaften erwartet werden. Vorteil der Benutzung: Zugriff auch Objekt ~300 mal schneller als ohne „nonatomic“

* retain: der Setter behält das Objekt

* assign: einfacher Zuweisungsoperator, meist für primitive Datentypen oder für den Delegate benutzt

* copy: sollte am Besten mit Strings oder Mutable Objekten verwendet werden.

* readonly: klar :)

Properties

```
- (void) setObject:(id)value{
  // für einfache Datentypen
  object = value;

  // für Objekte
  if(object != value){
     [object release];
     [value retain];
     object = value;
  }
}
- (id) object{
  return object;
}
```

einmal einen Setter von Hand geschrieben. Dieser Setter entspricht einer @propery(nonatomic, retain, readwrite) Beschreibung.

Xcode und InterfaceBuilder

Xcode

- War einmal der ProjectBuilder
- Ist in Applescript geschrieben
- Mag keine SCM

InterfaceBuilder

- Apple‘s kleines Schatzkistchen
- Sehr mächtig! Vor- oder Nachteil?
- Objekte aus dem nib-File werden sehr schneller geladen.

Xcode und InterfaceBuilder

Livedemo „Hello Cocoa“

UIKit
&
Memory Management
BASICS

Agenda

- Memory Management
- Apple‘s MVC
- Memory Warnings

Memory Management

Memory Management

- Es ist kein Garbage Collector verfügbar
- Das Erzeugen von Objekten belegt Speicher
- Der Speicher muss wieder freigegeben werden

Object Ownership

- Wer ein Objekt erzeugt ist Besitzer
- Besitzer sind für Freigabe verantwortlich
- Wer ein Objekt nicht besitzt darf es auch nicht freigeben
- Man besitzt Objekte, die mit `alloc`, `new` oder `copy` erstellt wurden.

```
NSString* s1 = [[NSString alloc] initWithString:@"Hello Cocoa"];
NSString* s2 = [s1 copy];
```

Es ist möglich, dass Instanz A ein Objekt erzeugt und an Instanz B weitergibt --> Wer ist verantwortlich?

Reference Counting

- Jedes Objekt hat einen Zähler: `retainCount`
- Objekte werden mit einem `retainCount` von `1` erzeugt
- Den `retainCount` kann man auch selbst:
  - inkrementieren: `-(id)retain`
  - dekrementieren: `-(id)release` kann
- Sobald `retainCount == 0` gilt wird der Speicher freigegeben

```objc
NSString* s1 = [[NSString alloc] initWithString:@"Hello Cocoa"];
int count = [s1 retainCount]; // (count == 1)

[s1 retain];
count = [s1 retainCount]; // (count == 2)

[s1 release]; // (retainCount == 1)

[s1 release]; // (retainCount == 0)  ==> Memory will be released
```

- Zu `alloc`, `new` und `copy` gehört immer ein Gegengewicht in Form von `release`

-(void)dealloc

```
@interface MyClass : NSObject {
  NSString* myString;
}
@end
```

```
@implementation MyClass
-(id)init{
  if(self = [super init]){
        myString = [[NSString alloc] initWithString:@"Hello Cocoa"];
  }
  return self;
}
- (void)dealloc {
  [myString release];
  [super dealloc];
}
@end
```

dealloc NIE selbst aufrufen!

-(id)autorelease

- Verzögertes `release`
- Objekte können Rückgabewert einer Methode sein, ohne das Prinzip der Ownership zu verletzen
- ==> Convenience Constructors

```
//NSString
+ (id)string;

//NSArray
+ (id)arrayWithArray:(NSArray *)anArray;
```

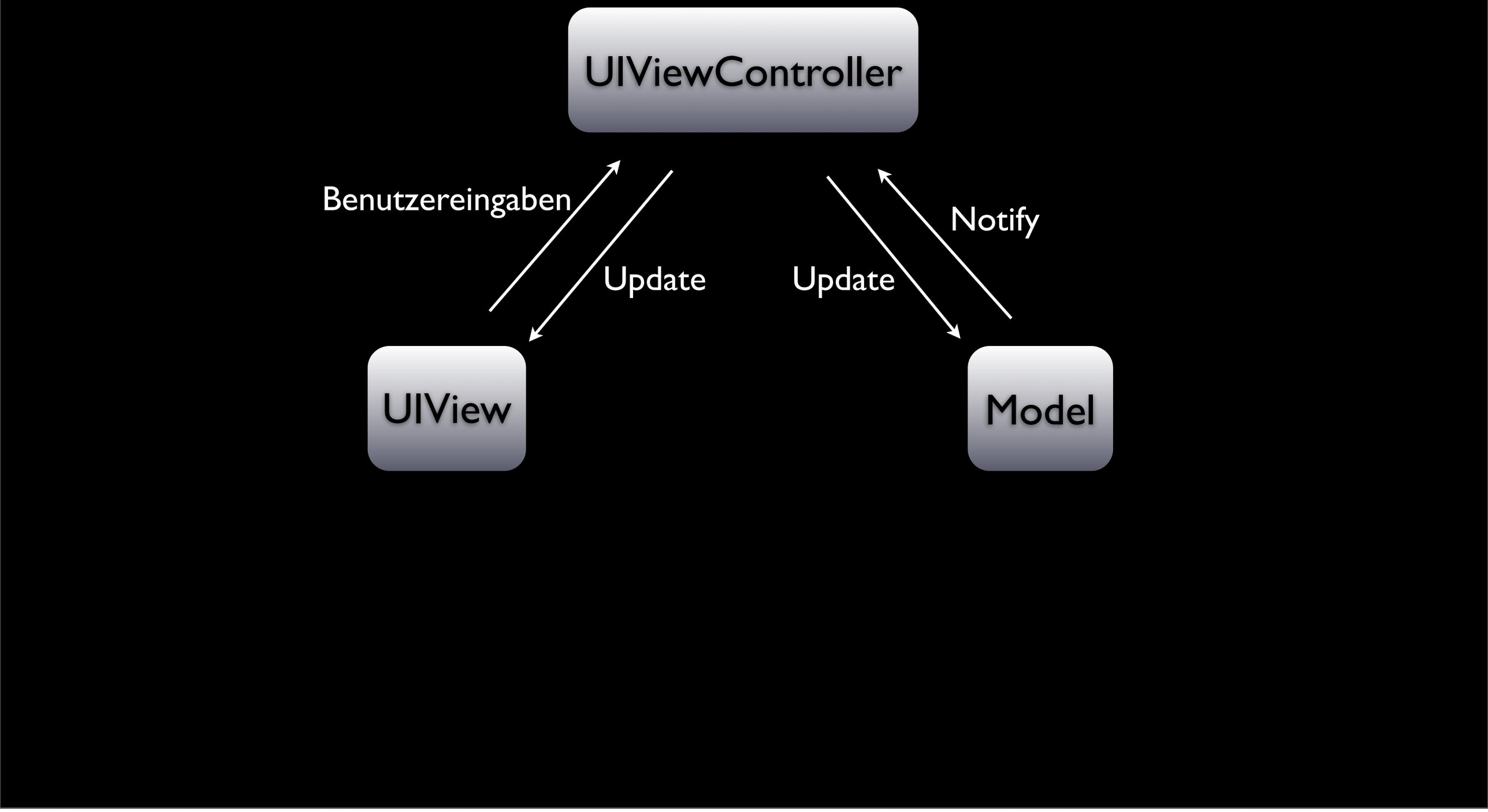
```objc
NSString* s1 = [[[NSString alloc] init] autorelease];
int count = [s1 retainCount]; // (count == 1)

NSString* s2 = [NSString string];
count = [s2 retainCount]; // (count == 1)
```

```objc
//Convenience constructor
+(id)string{
  return [[[NSString alloc] init] autorelease];
}
```

Apple‘s MVC

MVC



MVC is central to a good designed Cocoa Application, Cocoa Technologies and architectures are based on MVC and require that your custom classes play one of the MVC roles.

Bessere Wiederverwendbarkeit (insbesondere Model, View weniger, Controller selten)
Kein riesiges Monster, dass alles kann
Klare Verantwortlichkeiten, einfacher zu Verwalten
Verringert Abhängigkeiten

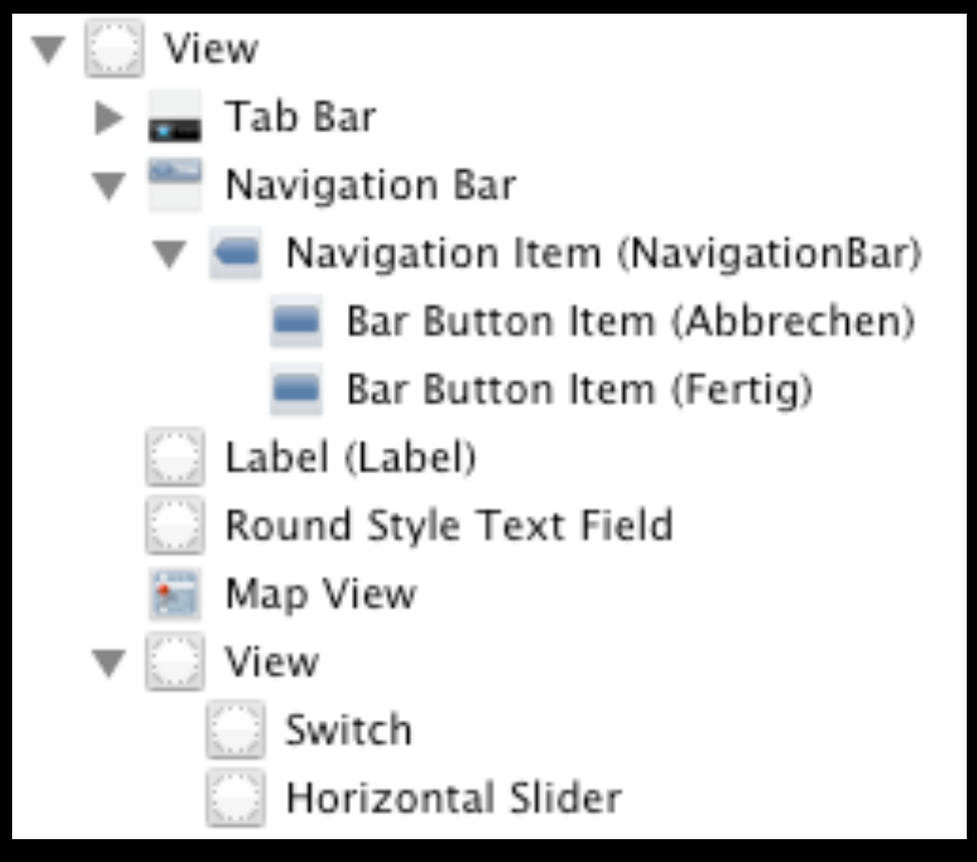
UIViewController

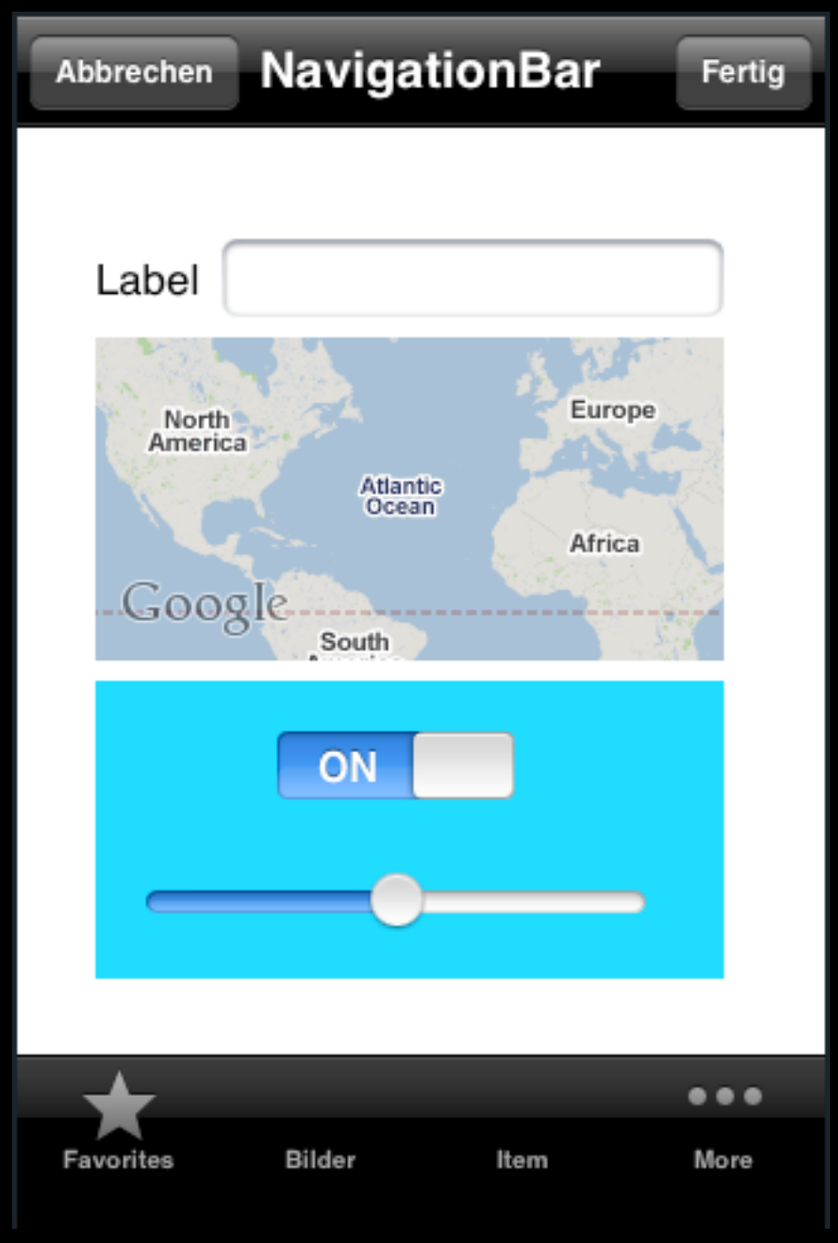
- Zentrale Elemente in jeder Anwendung
- Typisch ist ein Controller pro Bildschirm
- Verwaltet einen `UIView` (und dessen Subviews) sowie das Model
- Ableiten, um eigene Anwendungslogik zu implementieren
- Verbinden Bildschirme innerhalb einer Anwendung
  - ==> `UINavigationController`, `UITabbarController`

Kennt seine(n) View(s) und Model(s)
Anwendungslogik
Anwendungsspezifisch

UIView

- UIViews sind sichtbar
- Werden ineinander verschachtelt
  - ==> Subviews
- Eigene Views durch Ableitung





Kennt keinen bestimmten Controller
Kommunikation: Target-Action, Delegation

Views im Controller

- Der View eines Controllers:

```
@property (retain) UIView* view;
```

- Wird erst erzeugt, wenn er benötigt wird (lazy loading)
- Kann jederzeit zerstört werden (durch OS)

Schlecht ist es, wenn man zu Anwendungsbeginn alles(Views,Models) erzeugt. Wahrscheinlich wird der Anwender nie X Bildschirme in die Tiefe navigieren

Memory Warnings

Memory Warnings (iPhone OS >=3.0)

- Memory Warnings können jederzeit auftreten, auch bei sehr kleinen Apps
- Alle nicht sichtbaren Views werden automatisch freigegeben (release) und es wird `viewDidUnload()` auf allen betroffenen UIViewControllern aufgerufen.
- Die App muss reagieren bzw. korrekt implementiert sein, sonst wird sie evtl. einfach beendet

iPhone OS 3.0
Speicher ist knapp, insbes. auf dem iPhone mit Safari
Jeder ViewController im Speicher wird einzeln benachrichtigt

Memory Warnings (iPhone OS >=3.0)

- Um korrekt mit Memory Warnings umzugehen müssen die ViewController ihre Views in ...
  - `-(void)viewDidLoad` erstellen
  - `-(void)viewDidUnload` freigegeben
- Views, die im InterfaceBuilder erstellt wurden müssen nicht freigegeben werden
  - !!! Outlets müssen allerdings in `-(void)viewDidUnload` freigegeben werden

viewDidLoad kann auch als Konstruktor verwendet werden

Zusammenfassung

- Wer Objekte mit `alloc`, `new` oder `release` erstellt ist für deren Freigabe (mit `-release` bzw. `-autorelease`) verantwortlich
- Immer die `-dealloc` Methode jeder Klasse implementieren!
- In `-(void)viewDidLoad` erzeugte (View-) Instanzen in `-(void)viewDidUnLoad` wieder freigeben
- Alle Outlets in `-(void)viewDidUnLoad` freigeben

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.

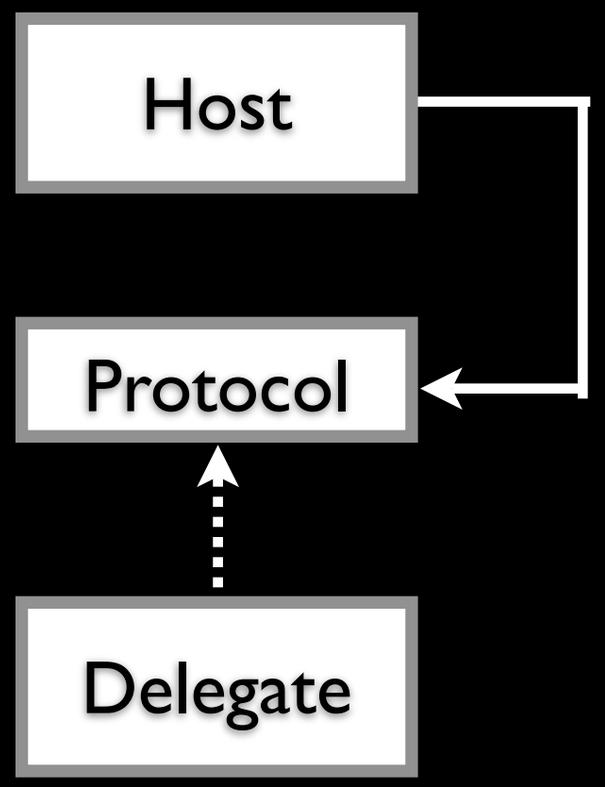
Keyboard of Doom

- 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

Rotation

Rotation

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

UIDeviceOrientationPortrait

UIDeviceOrientationLandscapeLeft UIDeviceOrientationLandscapeRight

UIDeviceOrientationPortraitUpsideDown

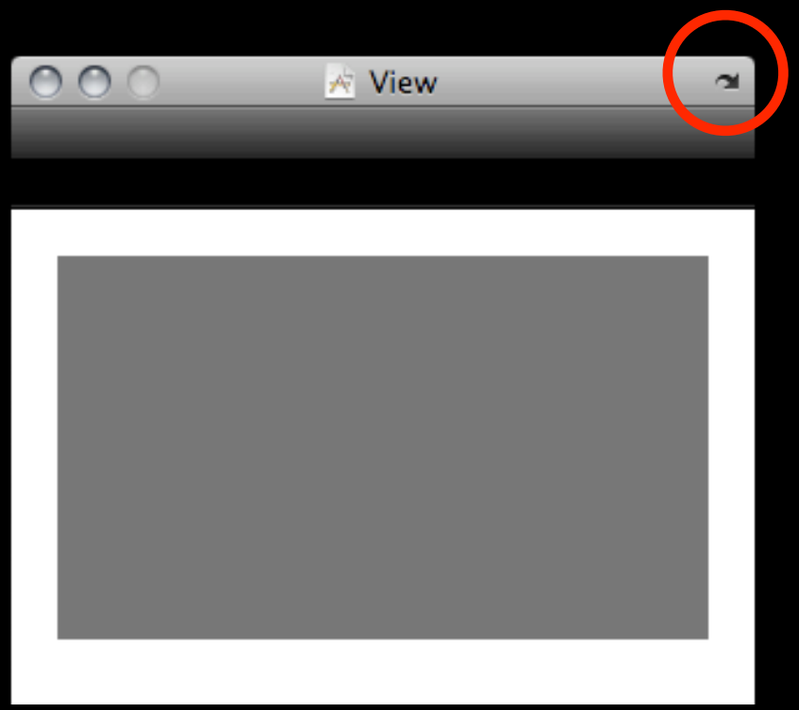
- Die Rotation ist Abhängig von der Orientierung der Statusbar

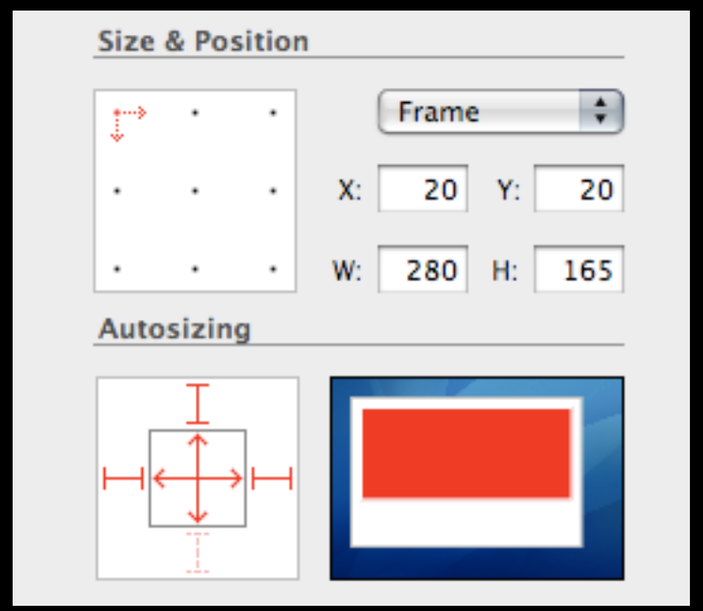
Rotation

- 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

Rotation





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“ mittels binärem Shifting 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

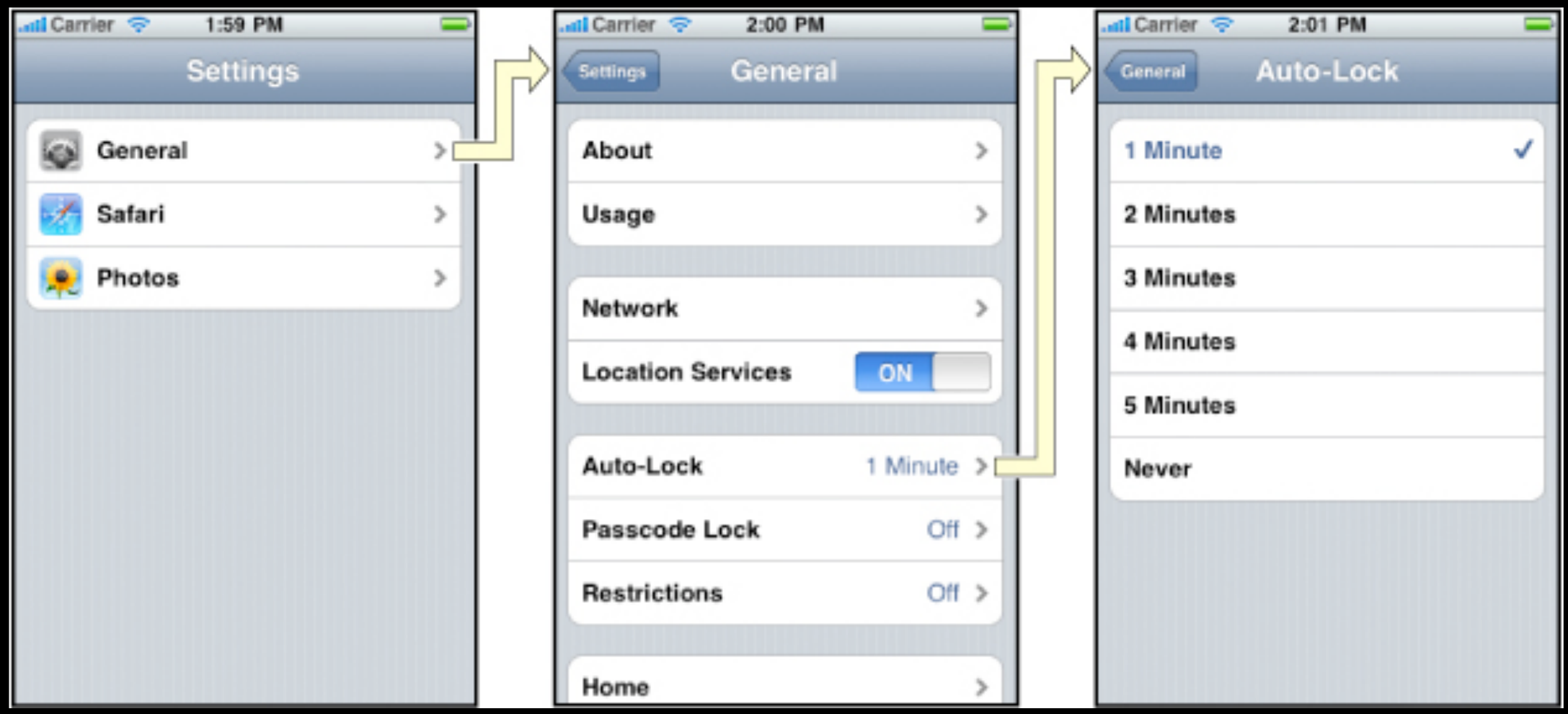
NavigationController
und
TableViews

UINavigationController

- Ist von UIViewController abgeleitet
- Verwaltet einen Stapel mit ViewControllern (Display Stack)
- Der oberste ViewController
  - ist sichtbar und
  - nimmt Benutzereingaben entgegen

Kann auch als Viewcontroller eingesetzt werden
Aber eigentlich nur für Verwaltung des Display Stacks
Keine Ableitung nötig/sinnvoll

UINavigationController



Automatisch: NavigationBar ,Titel, DrillDown- und PoP-Animationen, BackButton,

Erstellung

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
  //Create first PinkViewController
  PinkViewController* viewCon = [[[PinkViewController alloc]
                initWithNibName:@"PinkViewController" bundle:nil] autorelease];
  viewCon.title = @"PinkViewCon";

  //Create the NavigationController
  navigationController = [[UINavigationController alloc]
                                        initWithRootViewController:viewCon];

  [window addSubview:navigationController.view];
  [window makeKeyAndVisible];
}
```

Pushen (Drill-Down)

```
@implementation PinkViewController
-(void)buttonPressed:(id)sender{
  //Create a new ViewController instance
  PurpleViewController* viewCon = [[[PurpleViewController alloc]
           initWithNibName:@"PurpleViewController" bundle:nil] autorelease];

  viewCon.title = @"PurpleViewCon";

  //Push it onto the NavigationController
  [self.navigationController pushViewController:viewCon animated:YES];
}
@end
```

UITableView

- Zeigt Daten in einer Liste an
- Bezieht seine Daten von 2 Delegates
  - UITableViewDataSource
  - UITableViewDelegate

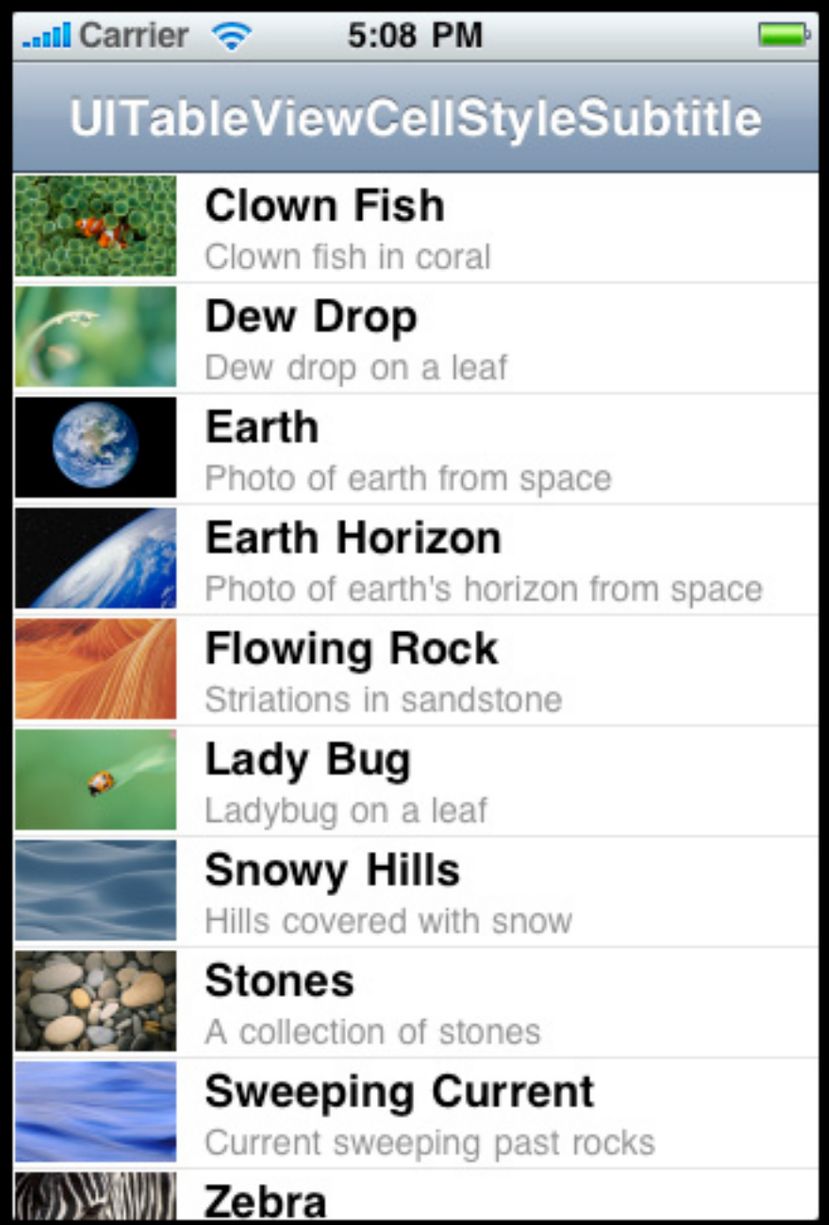
UITableViewController

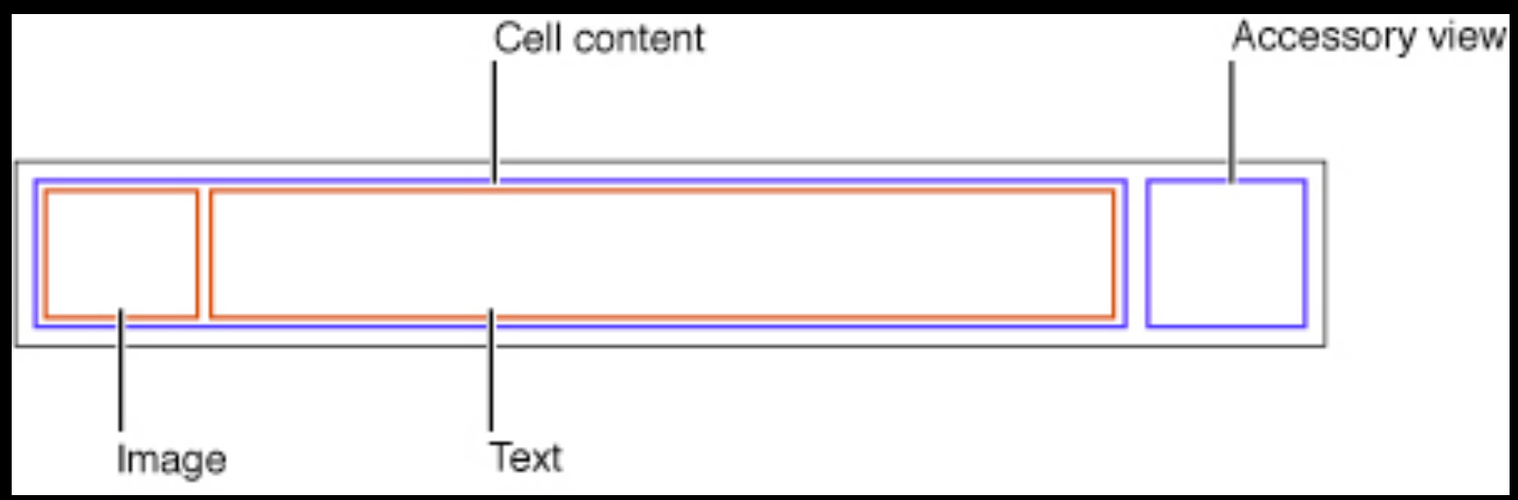
- Abgeleitet von UIViewController
- Bietet einen TableView von der Stange
- Im zugehörigen Template sind einige Methoden der TableView-Delegates schon implementiert

Sections, Grouped/Plain

UITableViewCell

- Sind die Zeilen im TableView
- 4 verschiedene Styles
- Es können auch neue Zellen implementiert werden





UITableViewDataSource

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return 1;
}
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)sec {
   return [dataArray count];
}
```

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

  static NSString *CellIdentifier = @"EinZellenIdentifier";

  UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:CellIdentifier];
  if (cell == nil) {
         cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                                                reuseIdentifier:CellIdentifier] autorelease];
     }
     cell.textLabel.text = [dataArray objectAtIndex:indexPath.row];

     return cell;
}
```

CoreData

CoreData

- Datenframework zur Handhabung von Daten und Datenstrukturen
- Wird oft zur Persistierung genutzt
- Hat Vor- und Nachteile gegenüber SQLite

Core Data ist KEINE Datenbank. Man kann CoreData benutzen ohne Daten auf „die Platte“ zu schreiben.

CoreData

- Arbeitet ausschließlich im Speicher während SQLite auf der „Platte“ arbeitet
- Erstellt Objekte und verknüpft diese (Objektgraphen)
- Ist beim Lesen und Schreiben von Daten ist CoreData langsamer als SQLite

Jedes Objekt muss bei CoreData erstellt werden um darauf zugreifen zu können. Aus diesem Grunde ist CoreData beim Zugriff auf viele Datensätzen langsamer als SQLite.

CoreData

- SQLite „DROP table...“ löscht (sofort)
- CoreData muß jedes Objekt vor dem löschen öffnen
- Große Datenmengen: "UPDATE table SET vorname = Tim WHERE nachname = Haussmann"

CoreData

- NSManagedObject: NSObject für CoreData
- NSManagedObjectContext: arbeitet mit dem Objektgraphen (sucht, löscht, erstellt)
- NSPersistentStoreCoordinator: speichert und lädt das Objektgraphen

Touches und Animation

Touches

- Verarbeiten die Touches
- Werden definiert in der Klasse UIResponder
- Erkennt „Touches“ und „Motion“

Touches

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

   UITouch *touch = [[event allTouches] anyObject];
   // UITouch* touch = (UITouch*)[touches anyObject];

   if([touch tapCount] == 2){
         // Doubletap
   }

   else{
         CGPoint point = [touch locationInView:self.view];
         // position of the tap
        NSLog(@“ x: @f y: @f“, point.x, point.y);

   }
}
```

Beim Umgang mit Touches ist die Dokumentation immer sehr hilfreich!

Ein Touch...

```
<UITouch: 0xd568c0>
phase: Began tap
count: 4
window: <UIWindow: 0xd19c00; frame = (0 0; 320 480); opaque = NO;
autoresize = RM+BM; layer = <CALayer: 0xd1a990>>
view: <UIView: 0xd55eb0; frame = (0 0; 320 436); autoresize = RM
+BM; layer = <CALayer: 0xd55be0>>
location in window: {139, 118}
previous location in window: {139, 118}
location in view: {139, 74}
previous location in view: {139, 74}
```

Details eines Touches, sie Beschreiben in verschiedenen Kontexten wo der Touch statt gefunden hat.

Animationen

- Einige Properties von UIView können sehr einfach animiert werden.
- Das UIKit übernimmt die Berechnung der Zwischenbilder
- Animierbare Properties sind:
  - frame
  - bounds
  - center
  - transform
  - alpha

Den Großteil der Animationsarbeit übernimmt Apple bereits. Fast alle Animationen sind implizit.

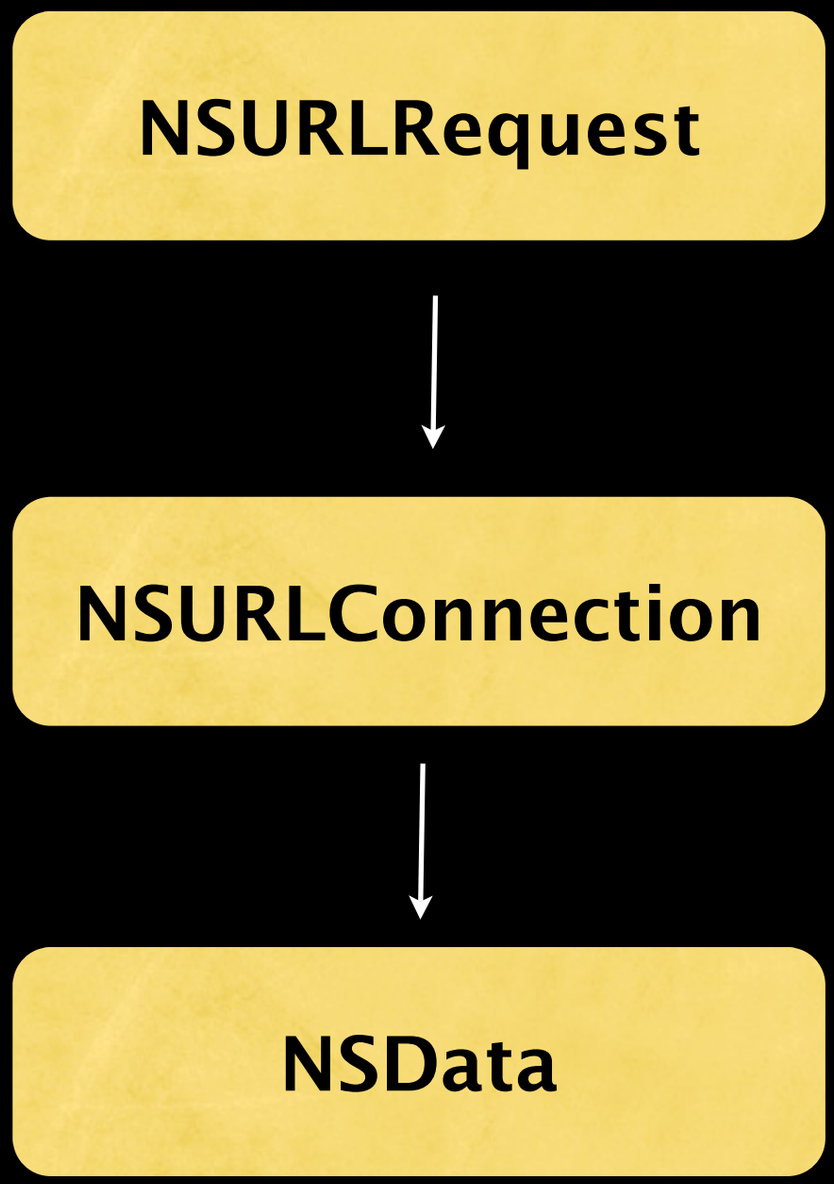
```objc
[UIView beginAnimations:@"SimpleAnimation" context:nil];
[UIView setAnimationDelegate:self];
[UIView setAnimationDidStopSelector:@selector(myDidStop)];
monsterImageView.center = CGPointMake(1,1);
[UIView commitAnimations];
```

```objc
[UIView beginAnimations:@"DelegationAnimation" context:nil];
[UIView setAnimationDuration:0.8f];
monsterImageView.alpha = 0.0;
[UIView commitAnimations];
```

```objc
-(void)myDidStop{
        //do something i.e. a further animation
}
```

HTTP Verbindung

HTTP Verbindung



Das Erstellen einer Verbindung ist recht leicht, eine NSURLConnection wird mit einem Request gestartet und liefert als Ergebnis Daten. Mehr das zu auf der nächsten Folie

HTTP Verbindung

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

request = [NSURLRequest requestWithURL:url];

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

[connection start];
```

das ist auch schon alles :)

HTTP 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 Verbindung

- Wichtig: **Abbrechen!**

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

Wichtig wichtig: Abbrechen im Fehlerfall oder beim Verlassen des ViewControllers!

Twitter

http://api.twitter.com/1/statuses/user_timeline/%@.xml

UIPicker und UserDefaults

UIPicker

- Besteht aus:
  - UIPickerView
- Und folgenden Protokollen:
  - UIPickerViewDelegate (!)
  - UIPickerViewDataSource (!)

    (! == Notwendig um den Picker sichtbar zu machen)

Der UIPicker verhält sich wie ein UITableView.

UIPicker

Interface

```
@interface PickerController : UIViewController <UIPickerViewDelegate, UIPickerViewDataSource>
```

Klasse

```
- (NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)row forComponent:(NSInteger)component{

   if(component == 0){ // die erste Komponente
         return [NSString stringWithFormat:@"%d", (row + 1)];
   }
   if(component == 1){ // die zweite Komponente
         if(row == 0){ // die erste Reihe der zweiten Komponente
             return @"Fleischbergsalat";
         }
   }
   return @"";
}
```

einfaches Implementationsbeispiel

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.

Instruments

Instruments

- nicht zu vergessen: Shark (!)
- 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

Tipps...!

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

- MoviePlayer: kann (bis jetzt) nur **Vollbild**

- Speichermanagement: erst **Apple**, 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.

- 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 Kalendar

- 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**