

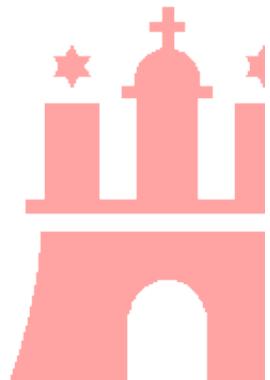


iPhone Summerschool 2011

Einführung in die Entwicklung von iOS-Applikationen
mit Objective-C und Xcode

Kai Meyer, Felix Zwingenberger, Jarig Richter-Peill

24. – 26. August, 9:00h – 17:00h



Begrüßung und Vorstellung



C1 WPS

Universität Hamburg
In Kooperation mit der C1 WPS GmbH

Kai Meyer: **kai.meyer@c1-wps.de**

- ◆ Software Entwickler
- ◆ Seit 4 Jahren Angestellter der C1 WPS
- ◆ Hat 2010 mehrere iPhone-Workshops mit der Uni Hamburg durchgeführt
- ◆ Seit Anfang 2009 mit iPhone Entwicklung beschäftigt

Felix Zwingenberger: **felix.zwingenberger@uni-hamburg.de**

- ◆ Software Entwickler
- ◆ Universität Hamburg
- ◆ Hat seit 2010 mehrere iPhone- und iPad-Workshops am Lehrstuhl SWT der Uni Hamburg durchgeführt
- ◆ Seit 2009 mit iPhone Entwicklung beschäftigt

Jarig Richter-Peill: **jarig@gmx.de**

- ◆ Software Entwickler
- ◆ Universität Hamburg
- ◆ Seit 2010 mit iPhone Entwicklung beschäftigt



Organisatorisches

- ◆ Vormittag
 - Treffen in D 125
 - Übungen in D 011 – D 013
- ◆ Nachmittags
 - Treffen in D 125
 - Übungen in D 011 – D 013
- ◆ Geplantes Ende jeweils um 17:00 Uhr



Scheinkriterien

- ◆ Teilnahme an allen 3 Tagen
- ◆ Bearbeitung der Übungsaufgaben
- ◆ Bearbeitung und Präsentation der Wochenaufgabe
 - Lauffähiger Code
 - Gewünschte Funktionen umgesetzt
 - Kaum Speicherlecks



Tag 1: **Einführung in Objective-C**
Einführung in iOS-Entwicklung

Tag 2: iOS Schichten
 Cocoa in a Nutshell

Tag 3: Objective-C Teil 2
 Testen
 Deployment
 Wrap-Up und Ausblick



- | | |
|-------|---------------------------------|
| 09:00 | → Einführung in Objective-C |
| | Aufgabe: Erstellen einer Klasse |
| 12:00 | Pause |
| 13:00 | Einführung in iOS-Entwicklung |
| | Aufgabe: Erstellen einer View |
| 17:00 | Ende |



Einführung in Objective-C

Motivation

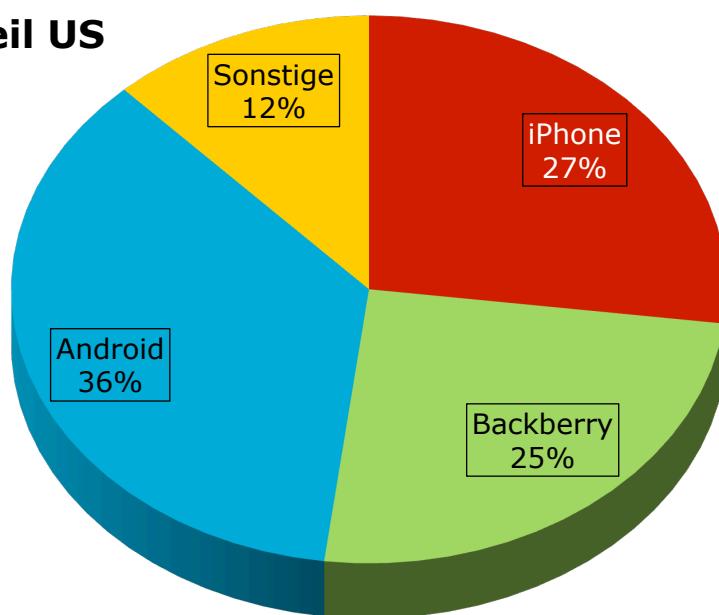


Das mobile Büro

- ◆ Smartphones sind ständige Begleiter
- ◆ Werden genutzt für:
 - E-Mails
 - Termine
 - Internet
 - ...
- ◆ Smartphones bieten aber noch mehr:
 - Kamera
 - GPS und Kompass
 - Beschleunigungssensoren
 - Installation neuer Anwendungen

- ◆ Smartphones sind nicht nur für Privatpersonen interessant
- ◆ Für Firmen bietet sie ebenfalls Anreize:
 - Zugriff auf Emails, Termine und Internet von unterwegs
 - Unterstützung der Mitarbeiter bei ihrer täglichen Arbeit durch Business-Anwendungen
- ◆ Native Programmierung der Business Anwendungen erlaubt zudem den vollen Zugriff auf alle Funktionen des Smartphones
- ◆ Dadurch können neue Szenarien abgedeckt werden, die:
 - die aktuelle Position berücksichtigen
 - erstellte Bilder von Objekten benutzen
 - ...

Marktanteil US





- ◆ Anwendungen für Smartphones sind oft über Stores zu beziehen
- ◆ Bequeme und einfache Art, um Anwendungen:
 - zu veröffentlichen
 - zu laden/kaufen
- ◆ iPhone-Entwickler können ihre Apps über den App-Store vertreiben
 - Kostenlos
 - Oder kostenpflichtig



- ◆ Im App Store von Apple sind über 450.000 Apps verfügbar
 - 10% der Apps sind speziell fürs iPad
 - 15% für beide
- ◆ 82% der Anwendungen sind kostenlos
- ◆ Durchschnittlicher Preis für eine App: \$ 1,44
- ◆ Anzahl der im App-Store vertretenen Entwickler: ca. 100.000
- ◆ Mehr als 15 Milliarden Downloads im App-Store
- ◆ Mehr als 200.000.000 iOS Nutzer weltweit
- ◆ Auszahlung an Entwickler (Schätzungen)
 - 2010: \$ 887 Millionen
 - 2011: \$ 2.991 Millionen
 - 2012: \$ 5.387 Millionen

- ◆ iOS ist das gemeinsame Betriebssystem von iPhone, iPod und iPad
 - Native Entwicklung nur mit Macs
 - Cross-Compiler sind erlaubt



- ◆ Aktuell ist das SDK in Version 4.3 verfügbar

© Arbeitsbereich Softwaretechnik

8/26/11

Bilder: Courtesy of Apple

13

- ◆ Native Programmierung mit Objective-C
- ◆ Apple stellt ein vollständiges und kostenloses SDK für die Entwicklung bereit
- ◆ Die Entwicklungsumgebung beinhaltet:
 - Xcode
 - iPhone/iPad-Simulator
 - Instruments
 - Ausführliche Dokumentation
- ◆ Für die Veröffentlichung ist eine kostenpflichtige Entwicklerlizenz nötig
 - 79 €: App-Store-Lizenz (pro Jahr)

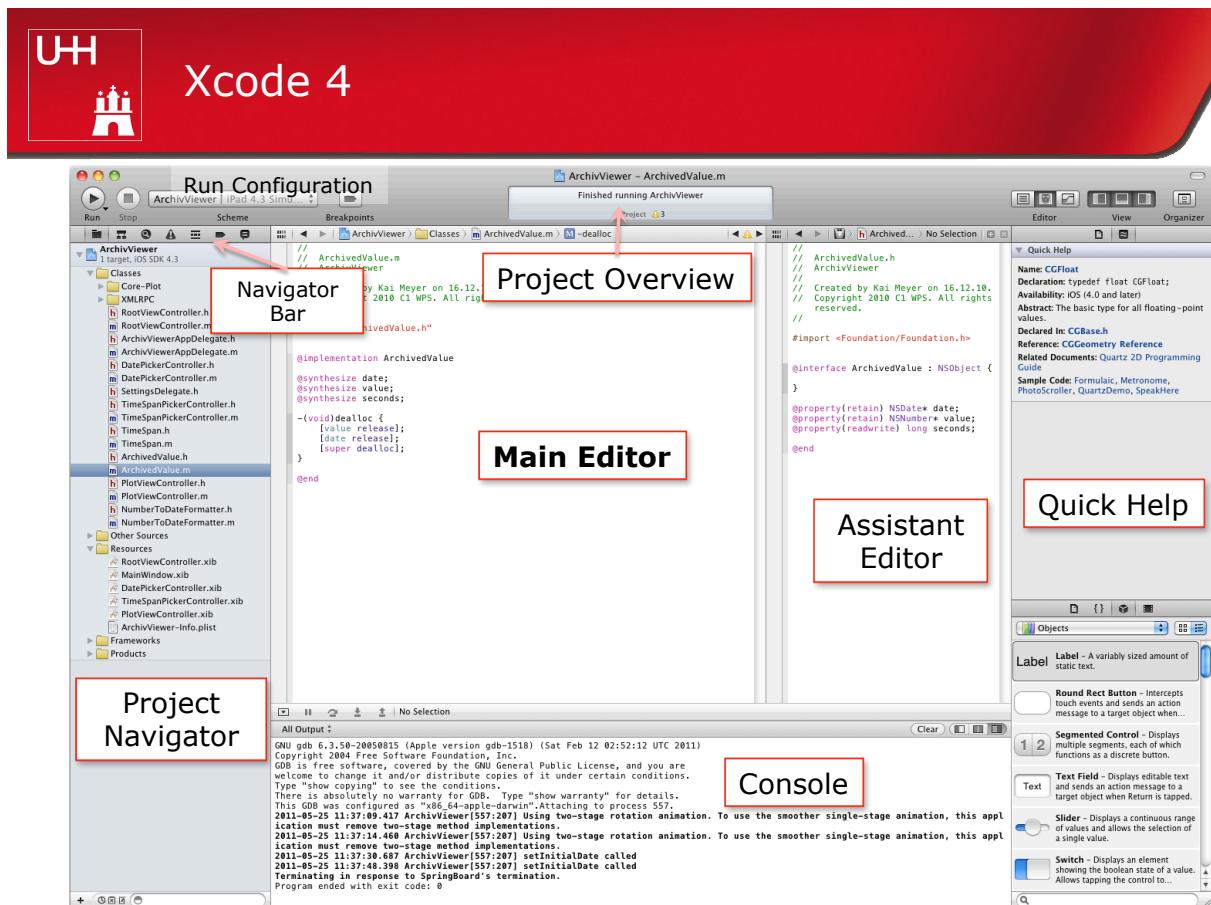
Einführung in Objective-C

Das iOS SDK

- ◆ IDE für Mac- und iOS-Entwicklung
- ◆ Nur auf dem Mac verfügbar

- ◆ Vollständige Entwicklungsumgebung
 - Code-Editor
 - Projektverwaltung
 - Dokumentationsbrowser
 - Diagnose-Tools
 - Interface Builder
 - iPhone- und iPad-Simulator





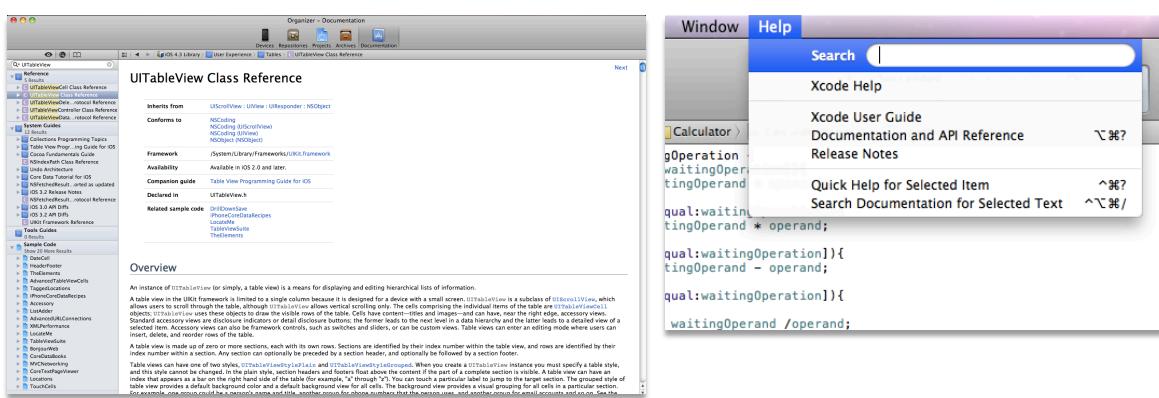
© Arbeitsbereich Softwaretechnik

8/26/11

17



- ◆ Oftmals keine Fehlererkennung „on the fly“
 - erst nach „Build“
- ◆ Auto vervollständigung nicht besonders mächtig
 - mit Esc oder Strg-Leertaste aufrufbar
- ◆ Dokumentations-Browser ist integriert



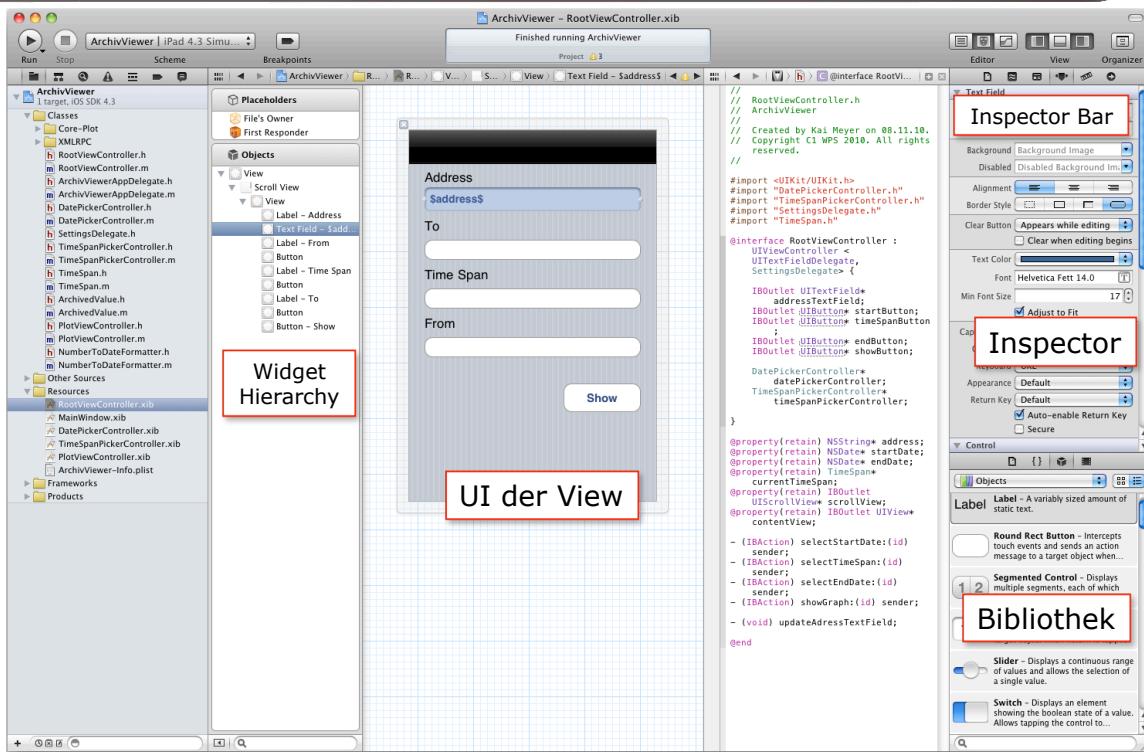
© Arbeitsbereich Softwaretechnik

8/26/11

18



Interface Builder



© Arbeitsbereich Softwaretechnik

8/26/11

19



Interface Builder

- ◆ Grafischer Editor für GUIs
 - Drag&Drop-Erstellung von UIs
 - UIs werden in NIB-Dateien gespeichert (*.xib)
- ◆ Mit Code gekoppelt
 - Spezielle Keywords im Code
 - IBOutlet, um Felder und UI zu verbinden
 - IBAction, um Methoden mit Buttons etc. zu verbinden



© Arbeitsbereich Softwaretechnik

8/26/11

20

- ◆ Schnelles Deployment
- ◆ Ohne Lizenz nutzbar
 - im Gegensatz zu einem echten Gerät
- ◆ Nicht geeignet für Performance-Tests
- ◆ Kein Ersatz für Entwicklung mit Test-Gerät
- ◆ Simulation von 2-Finger-Gesten möglich
- ◆ Rotation + "Schütteln" simuliert



© Arbeitsbereich Softwaretechnik

21

Einführung in Objective-C

Objective-C

◆ Eigenschaften

- Obermenge von C
- objektorientiert
- dynamisch typisiert
- Einfachvererbung

◆ Nachricht vs. Methode

- Jedem Objekt kann jede Nachricht geschickt werden (prinzipiell)
- Neue Nachrichten zur Laufzeit möglich

◆ Aufruf von Methoden:

```
[objekt doSomething];
```



◆ Aufruf mit Argument:

```
[objekt doSomethingWithInt:20];
```

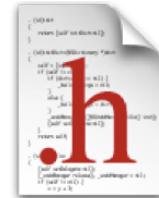
◆ Mehrere Argumente:

```
[objekt doSomethingWithWidth:20 andHeight:30];
```



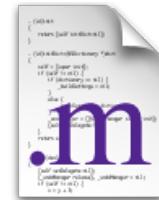
◆ Öffentliches Interface (Header)

- Typischerweise in .h-Datei
- Deklaration von Exemplarvariablen
- Deklaration von Methoden
- Deklaration von Properties



◆ Private Implementation

- Typischerweise in .m-Datei
- Methoden-Implementationen



◆ Deklaration (Konto.h)

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

@interface Konto : NSObject
{
    Person* inhaber;
    double guthaben;
}

- (Person*) inhaber;
- (void) setInhaber:(Person*) aPerson;
- (double) guthaben;
- (void) setGuthaben:(double) anAmount;
@end
```

Header-Imports: #import <UIKit/UIKit.h>

Konto erbt von NSObject:@interface Konto : NSObject

Referenz auf ein Person-Objekt: Person* inhaber;

Exemplarvariablen: double guthaben;

Methodendeklarationen:

- (Person*) inhaber;
- (void) setInhaber:(Person*) aPerson;
- (double) guthaben;
- (void) setGuthaben:(double) anAmount;

+ Klassenmethoden

- Objektmethoden

◆ Implementation (*Konto.m*)

```
#import "Konto.h"
```

Header der Klasse

```
@implementation Konto
```

// Getter

```
- (Person*) inhaber {  
    return inhaber;  
}
```

Getter:
typischerweise ohne get...

// Setter

```
-(void) setInhaber:(Person*)aPerson {  
    [aPerson retain];  
    [inhaber autorelease];  
    inhaber = aPerson;  
}
```

Setter:
inklusive Speichermanagement

```
@end
```

© Arbeitsbereich Softwaretechnik

8/26/11

27



◆ Rückgabewerte

```
Person* inhaber = [konto inhaber];
```

◆ Verschachtelung

```
-(Person*)inhaber;  
-(NSString*)name;
```

```
[[konto inhaber]name];
```

◆ Klassenmethoden durch + gekennzeichnet

```
+ (id)alloc;
```

◆ Objektmethoden durch - gekennzeichnet

```
- (id)init;
```

© Arbeitsbereich Softwaretechnik

8/26/11

28



◆ Objective-C

```
[objekt doSomething];  
[objekt doSomethingWithInt:20];  
[objekt doSomethingWithWidth:20 andHeight:30];  
[[konto inhaber] name];
```

◆ Java

```
objekt.doSomething();  
objekt.doSomethingWithInt(20);  
objekt.doSomethingWithSize(20, 30);  
konto.getInhaber().getName();
```



◆ @property

- Spezielle Deklaration von Exemplarvariablen
- Vorgefertigte Speicherverwaltung
- Automatisches Erstellen von Gettern + Settern
- Punktnotation zum Zugriff auf Felder

```
@interface Konto: NSObject  
{ }  
@property (retain) Klasse* propertyName;  
@property (readonly) double doubleProperty;  
  
@end
```

◆ Alt:

```
@interface Konto : NSObject
{
    Person* inhaber;
    double guthaben;
}

- (Person*) inhaber;
- (void) setInhaber:(Person*) aPerson;
- (double) guthaben;
- (void) setGuthaben:(double) anAmount;

@end
```



◆ Neu:

```
@interface Konto: NSObject
{ }
@property (retain) Person* inhaber;
@property (readonly) double guthaben;

@end
```

◆ Alt:

```
@implementation Konto
{
    - (Person*) inhaber {
        return inhaber;
    }

    -(void) setInhaber:(Person*)aPerson {
        [aPerson retain];
        [inhaber autorelease];
        inhaber = aPerson;
    }
}
```

@end



◆ Neu:

```
@implementation Konto
@synthesize inhaber;
@synthesize guthaben;
...
@end
```

- ◆ Vereinfachter Zugriff auf Properties
- ◆ Abkürzung für Getter und Setter

```
String *name = [person name];
String *name = person.name;
[person setName:@"Max Muster"];
person.name = @"Max Muster";
int alter = [[konto inhaber] alter];
int alter = konto.inhaber.alter;
```

Achtung!

person.name = @"Max Muster";

Methodenaufruf

name = @"Max Muster";

Zuweisung

- ◆ ... aber nur für Get- und Set-Methoden verwenden!

person.doSomething;



© Arbeitsbereich Softwaretechnik

8/26/11

33

- ◆ Aufruf von 2 Nachrichten zur Objekterstellung:
 1. +alloc, um Speicher für das Objekt bereitzustellen.
 2. -init, um das Objekt anschließend zu initialisieren.
 - wird in eigenen Klassen überschrieben.

Objective-C:

```
-(id) initWithName:(NSString *)aName andText:(NSString *)aText;
Article *article = [[Article alloc] initWithName:@"Objective C"
andText:@"Lorem Ipsum"];
```

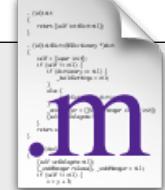
Java:

```
public Article(String name, String text) { ... }
Article article = new Article("Java", "Lorem Ipsum ...");
```

- ◆ Alternativ oft eine Klassenmethode, z.B. [Article newArticle];

◆ Konstruktor-Implementation in Objective-C:

```
- (id) initWithName:(NSString *)aName andText:(NSString *)aText {  
    self = [super init];  
    if (self) {  
        [self setName:aName];  
        [self setText:aText];  
    }  
    return self;  
}
```



- Zuweisung an self und Überprüfung sind Konvention, falls bei der Objekt-Erstellung mal was schief läuft.
- Auch hier ist Speichermanagement nötig, deshalb Zuweisung der Parameter durch Setter.
- Init-Methoden geben das initialisierte Objekt zurück.

- ◆ ähnlich wie in Java
- ◆ if-else-Statement

```
if (condition) {  
    statements  
} else {  
    statements  
}
```

- ◆ for-Schleife

```
for (initial; condition; increment) {  
    statements  
}  
for (NSObject* current in array) {  
    statements  
}
```



- ◆ **id**
 - Allgemeiner Pointer-Typ: "irgendein Objekt"
- ◆ **NSObject**
 - Basisklasse von Cocoa-Klassen
 - Bietet Speichermanagement (retain- und release-Methoden)
 - Nicht alle Klassen erben von NSObject
- ◆ **nil**
 - In Objective-C statt **NULL** verwendet
 - Kann jede Nachricht empfangen, kein Fehler
- ◆ **self**
 - Entspricht **this** in Java
- ◆ **NSString**
 - `NSString *myString = @“hello world!“;`



- ◆ **NSLog** gibt Nachrichten auf der Console aus
 - `NSLog(@“Nachricht für die Console“);` → Nachricht für die Console
- ◆ Formatieren von **NSString** mit % als Platzhalter
 - `NSLog(@“Nachricht %d für die Console“, 42);` → Nachricht **42** für die Console
 - `NSLog(@“Nachricht von %@!!“, someObject);` → Nachricht von **Objekt 4!!**
 - **%@** für beliebige Objekte
 - **%d** für Integer
 - **%f** für Float



DeineAppDelegate.m

Aufgabe 1

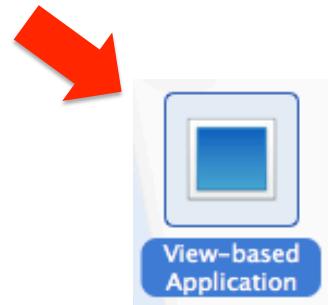
Erstellen einer Klasse

Ziel:

- ◆ Eine Klasse „Article“ ist zu erstellen.
 - Die Attribute der Klasse sollen bei Programmstart auf der Console ausgegeben werden.

Weg:

- ◆ Erstelle ein neues **View-basiertes Projekt** in Xcode.
(iOS → Application → View-based Application).
 - für iPhone
 - ohne Unit Tests
- ◆ Erstelle eine **Klasse „Article“**
(iOS → Cocoa Touch → Objective-C Class).
 - Erzeuge hierzu in dem Ordner (Group), der so heißt wie dein Projekt, eine neue Datei.
 - Die Klasse soll von *NSObject* erben.



- ◆ Article-Objekte sollen folgende **Attribute** haben:
 - *titel* des Artikels als *NSString**,
 - *text* des Artikels als *NSString**,
 - *url* zum Artikel als *NSURL**-Objekt,
 - *datum* der Veröffentlichung als *NSDate**-Objekt.

Hinweis: Klassen groß schreiben: *BeispielKlasse*
Methoden und Variablen klein: *beispielMethode*, *beispielVariable*

- ◆ Die Attribute sollen über **Getter** und **Setter** zugänglich sein.
 - Versuche zuerst, einige Get- und Set-Methoden per Hand zu implementieren.
 - Versuche außerdem, die Getter und Setter per @property und @synthesize zu erstellen.



Aufgabe 1: Erstellen einer Klasse

- ◆ Implementiere eine **Methode**, die das Datum als formatierten String liefert.
 - **NSDateFormatter** bietet hilfreiche Methoden dafür an (siehe Xcode-Doku).

Hinweis: **Dokumentation** zu Elementen im Quellcode: **[alt]+Klick** auf das Element bzw. **[strg]+[alt]+[cmd]+[/]** bei selektiertem Element.

- ◆ Importiere den **Header** deiner Klasse **in das AppDelegate**.
- ◆ Erstelle in der Methode „**didFinishLaunchingWithOptions**“ des **AppDelegate** Objekte deiner Klasse „Article“.
 - Lasse dir einige Werte auf der Console mit **NSLog(...)** ausgeben.

Hinweis: **Kompilieren** mit **[cmd]+[b]**
Starten mit **[cmd]+[r]**

- ◆ Optional: Schreibe eine eigene **Init-Methode** in deine Klasse „Article“, die als zusätzliche Parameter *Titel*, *URL* und *Text* enthält.
 - Erstelle deine Objekte mit dieser Init-Methode.



Wrap up

- ◆ iOS SDK enthält Xcode, iPhone Simulator, Interface Builder, etc.
- ◆ Native iOS-Entwicklung mit Objective-C
 - Aufruf von Methoden

```
[object doSomething];
```

```
[object doSomething:20];
```
 - Properties

```
@property(retain) NSString* name;
```

```
object.name = @"Name";
```



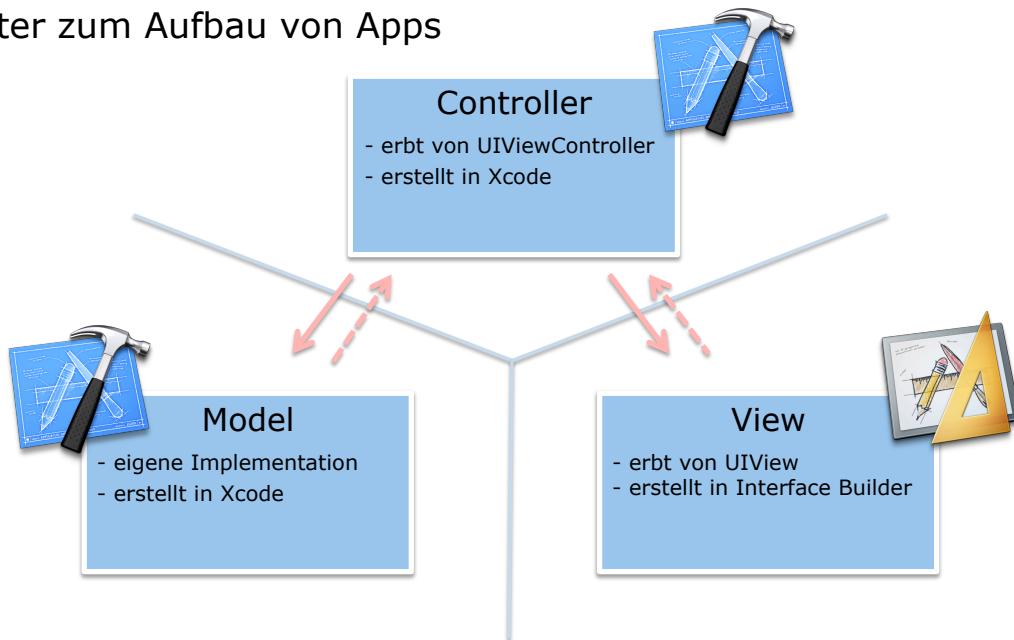
Agenda Tag 1

- | | |
|-------|------------------------------------------------------------------|
| 09:00 | Einführung in Objective-C
Aufgabe: Erstellen einer Klasse |
| 12:00 | Pause |
| 13:00 | → Einführung in iOS-Entwicklung
Aufgabe: Erstellen einer View |
| 17:00 | Ende |

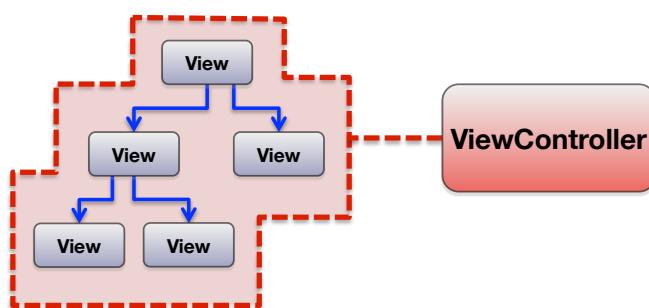


Einführung in iOS-Entwicklung

- ◆ Muster zum Aufbau von Apps



- ◆ Views werden in Hierarchien angeordnet
- ◆ Controller sind zuständig für eine Teilhierarchie
 - Meistens für "einen Bildschirm" in der App





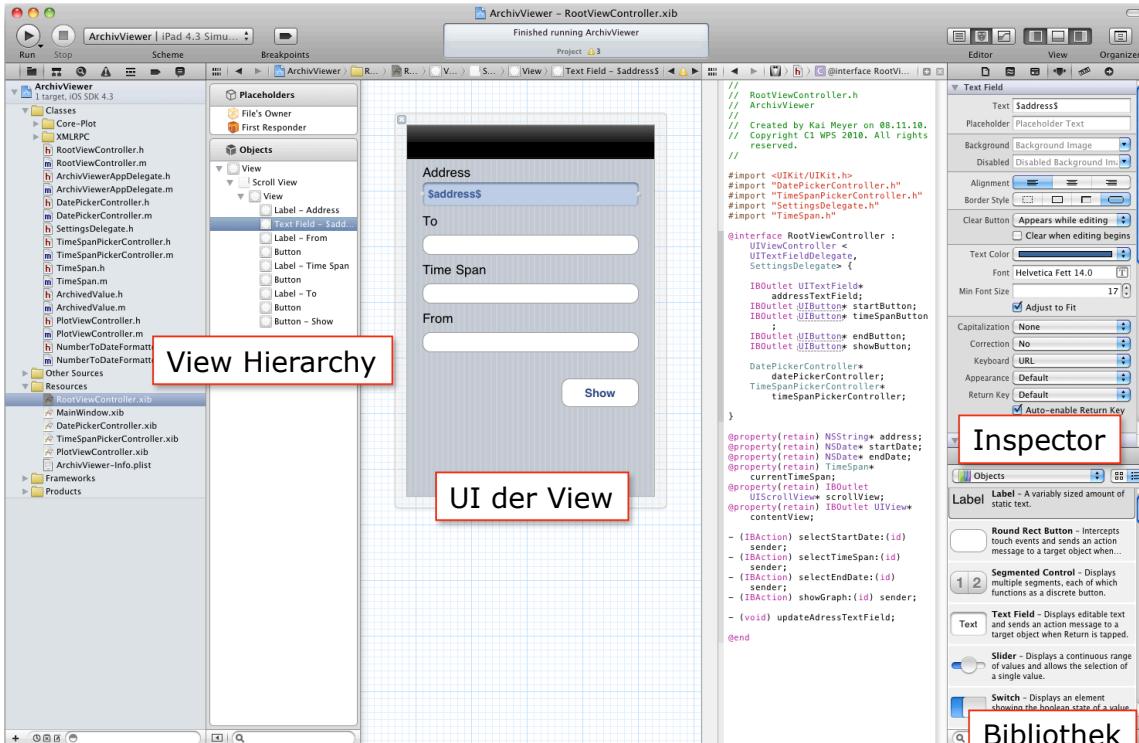
Interface Builder

◆ Grafischer Editor für GUIs

- Drag & Drop-Erstellung von UIs
- In Xcode integriert
- UI-Objekte werden zur Design-Zeit erzeugt und serialisiert
- In XIB-Datei gespeichert
- Durch spezielle Schlüsselwörter im Code Verbindung mit Controller
 - ❖ IBOutlet (für Exemplarvariablen von UI-Elementen)
 - ❖ IBAction (um Methoden und Kontroll-Elemente zu verbinden)

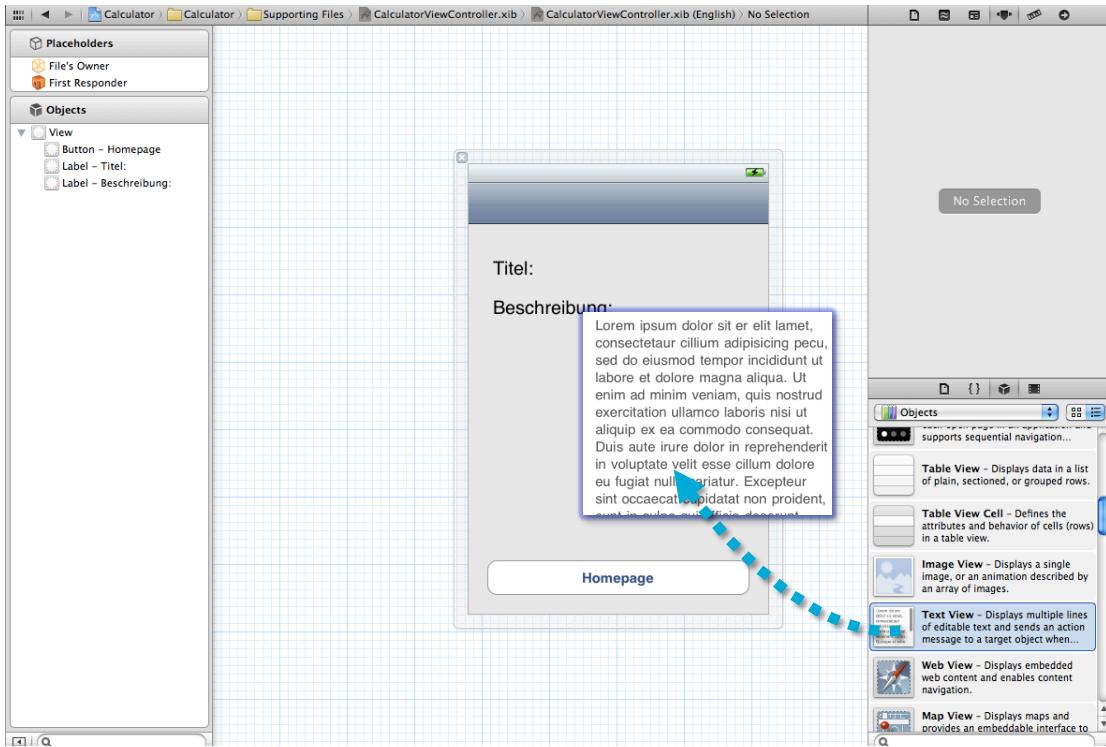


Interface Builder





Drag & Drop zum Aufbau der UI



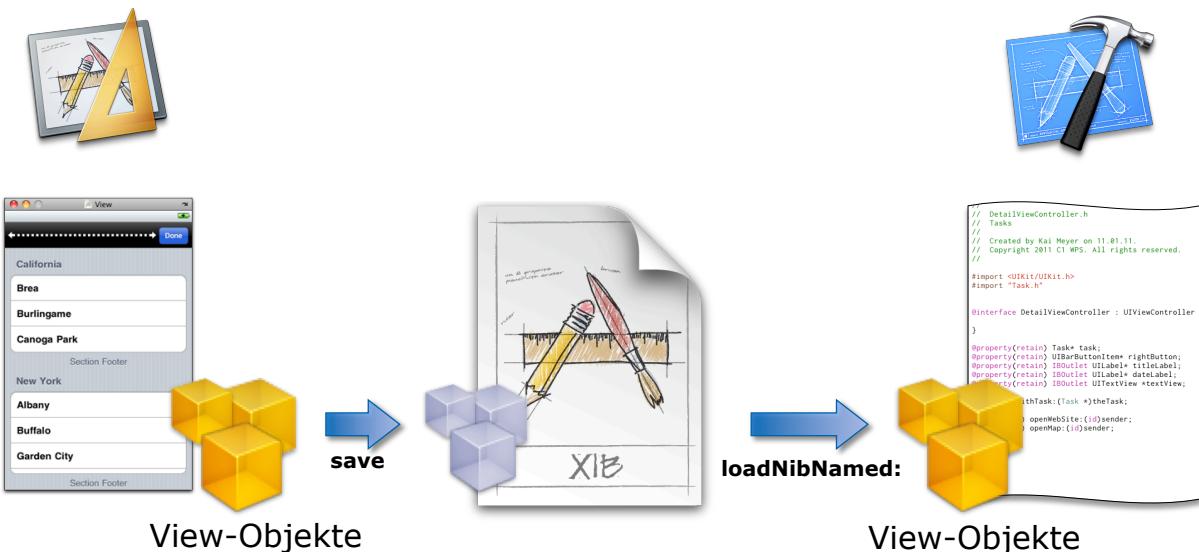
© Arbeitsbereich Softwaretechnik

8/26/11

51



XIB-Dateien



© Arbeitsbereich Softwaretechnik

8/26/11

52



XIB: File's Owner

- ◆ Verantwortliches Objekt für XIB-Datei
- ◆ Typischerweise Controller-Objekt
- ◆ Lädt die XIB-Datei im Code
- ◆ Beispiel:

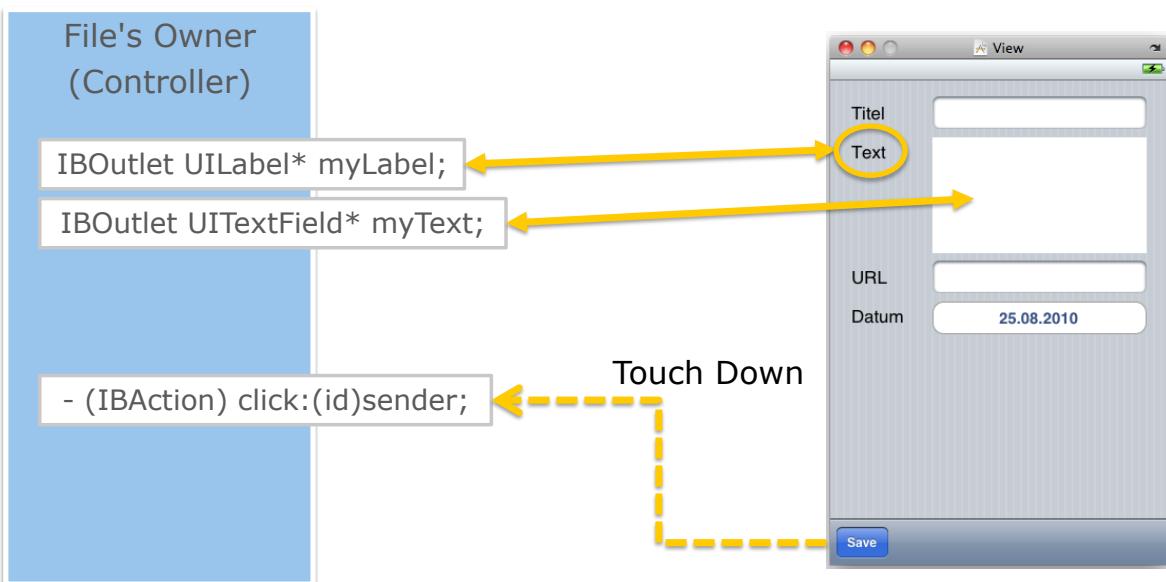
MyViewController.M:

```
@implementation  
-(void)someMethod {  
    ...  
    NSBundle* mainBundle = [NSBundle mainBundle];  
    [mainBundle loadNibNamed:@"SignGridCell" owner:self options:nil];  
}  
@end
```

Name der XIB **self == File's Owner**



IBAction & IBOutlet





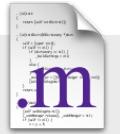
IBOutlet (1)

- ◆ Mit Outlets werden UI-Elemente mit Exemplarvariablen verknüpft
- ◆ Deklaration in Xcode, Verknüpfung in Interface Builder
- ◆ Beispiel Deklaration:

```
@interface YourClass {  
    IBOutlet UILabel *label;  
    ...  
}  
@property (nonatomic, retain) IBOutlet UISlider *slider;  
...  
@end
```

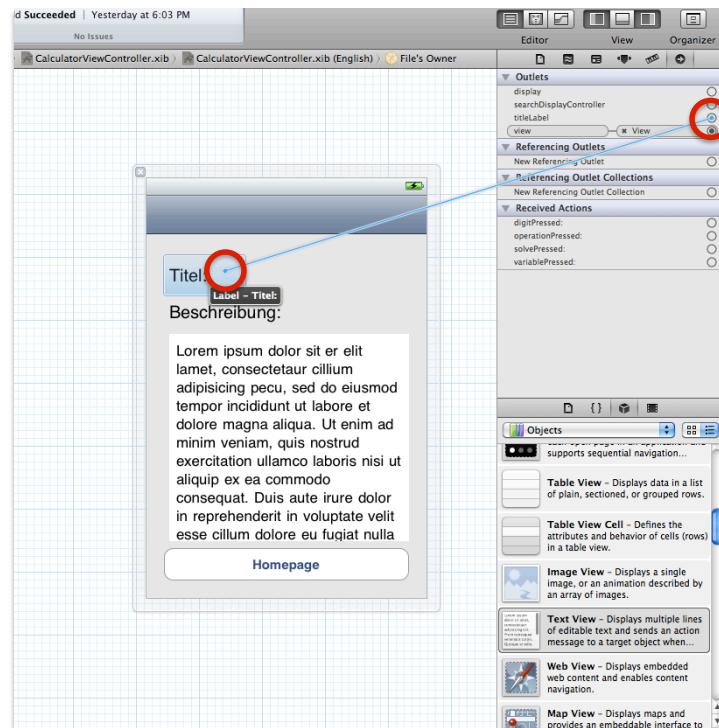


```
@implementation YourClass  
@synthesize slider;  
...
```



IBOutlet (2)

- ◆ Beispiel Verknüpfung:



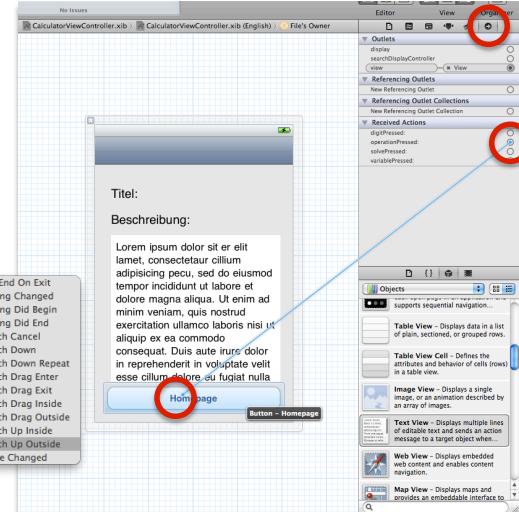
- ◆ Rückgabewert für Action-Methoden

- ◆ Deklaration

```
- (IBAction) doneButtonPressed;
- (IBAction) respondToButtonClick:(id)sender;
```

- ◆ Wird von IB erkannt

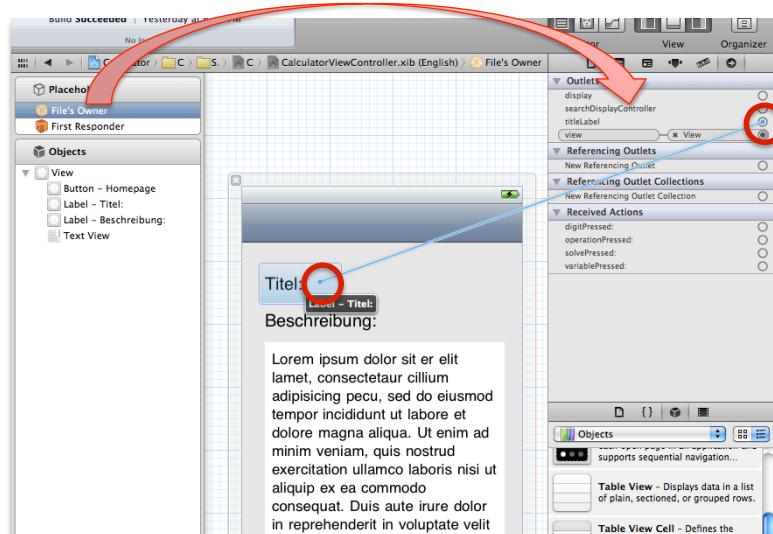
- Verknüpfung der Methode mit Buttons, Slidern etc.



- ◆ Verwalten die Darstellung eines Bildschirms der App
- ◆ Basisklasse für eigene Controller
- ◆ Bieten bereits Funktionalität an
 - Rotation der Views
- ◆ Look & Feel von iPhone-Apps bei
 - Animation
 - Darstellung von Listen
(UITableViewController)
 - Darstellung von Detail-Informationen
(UINavigationController)
 - Aufteilung der UI in Tabs
(UITabBarController)

◆ Lädt bei Initialisierung die XIB-Datei

- Controller ist der **File's Owner** in der XIB-Datei
- Buttons können mit **IBAction**-Methoden verknüpft werden
- Views können über **IBOutlets** verknüpft werden



Aufgabe 2

Erstellen einer View

Ziel

- ◆ Eine **Oberfläche** (View) für das Anlegen von Artikeln ist zu erstellen.
- ◆ Ein Artikel soll folgende **Attribute** haben:
 - *Titel* des Artikels,
 - *Text* des Artikels,
 - *URL* zum Artikel,
 - *Datum* (optional).
- ◆ Der **ViewController ist der MVC-Controller** und soll das Modell (den Artikel) verwalten.
- ◆ Beim drücken auf den **Speichern-Button** sollen die Daten aus der UI in das Modell (Artikel) übertragen und auf der Console ausgegeben werden.



mögliche Oberfläche

Weg

- ◆ Öffne die bereits zu deinem View-basierten Projekt angelegte **Nib-Datei** (.xib) und erstelle die Oberfläche.
 - **Achtung:** Nicht die MainWindow.xib verändern!
- ◆ Verknüpfe **grafische Elemente** über **IBOutlets** mit dem Quellcode.
 - Deklariere und implementiere bzw. synthetisiere die IBOutlets im Quellcode.
 - Verknüpfe die grafischen Elemente im InterfaceBuilder mit den IBOutlets.
 - Die IBOutlets gehören in den MVC-Controller (ViewController).
- ◆ Verknüpfe den **Save-Button** über eine **IBAction** mit dem Quellcode.
 - Deklariere und implementiere die IBAction im Quellcode.
 - Verknüpfe das grafische Element im InterfaceBuilder mit der IBAction
 - In der IBAction sollen die Daten aus den Outlets in das Modell übertragen werden und anschließend auf der Console ausgegeben werden. → Dazu braucht der MVC-Controller (ViewController) eine Referenz auf das Modell (Artikel)
 - Die IBActions gehören in den MVC-Controller (ViewController).



Aufgabe 2: Erstellen einer View

- ◆ Die **Tastatur** soll beim drücken auf den return-Button wieder ausgeblendet werden.
 - Erstelle dazu eine IBAction für die Textfelder.
 - Verknüpfe das Event „**Did End On Exit**“ der Textfelder mit der IBAction.
 - Verschicke aus der IBAction die Nachricht „**resignFirstResponder**“ an das Textfeld.
 - Hinweis: Das Textfeld über gibt in der IBAction eine Referenz auf sich.
- ◆ Optional: Der **Date-Picker** kann über **eine weitere View** eingebunden werden.
 - Die neue View kann über „**presentModalViewController**“ angezeigt werden.
`[self presentModalViewController:controller animated:YES];`



Die Summerschool im Überblick

- Tag 1: Einführung in Objective-C
 Einführung in iOS-Entwicklung
- Tag 2:** **iOS Schichten**
 Cocoa in a Nutshell
- Tag 3: Objective-C Teil 2
 Testen
 Deployment
 Wrap-Up und Ausblick



Wrap up

- ◆ Native iOS-Entwicklung mit Objective-C
- ◆ UI wird mit dem Interface Builder erstellt
- ◆ Verknüpfung wird über IBOutlets und IBActions erzeugt

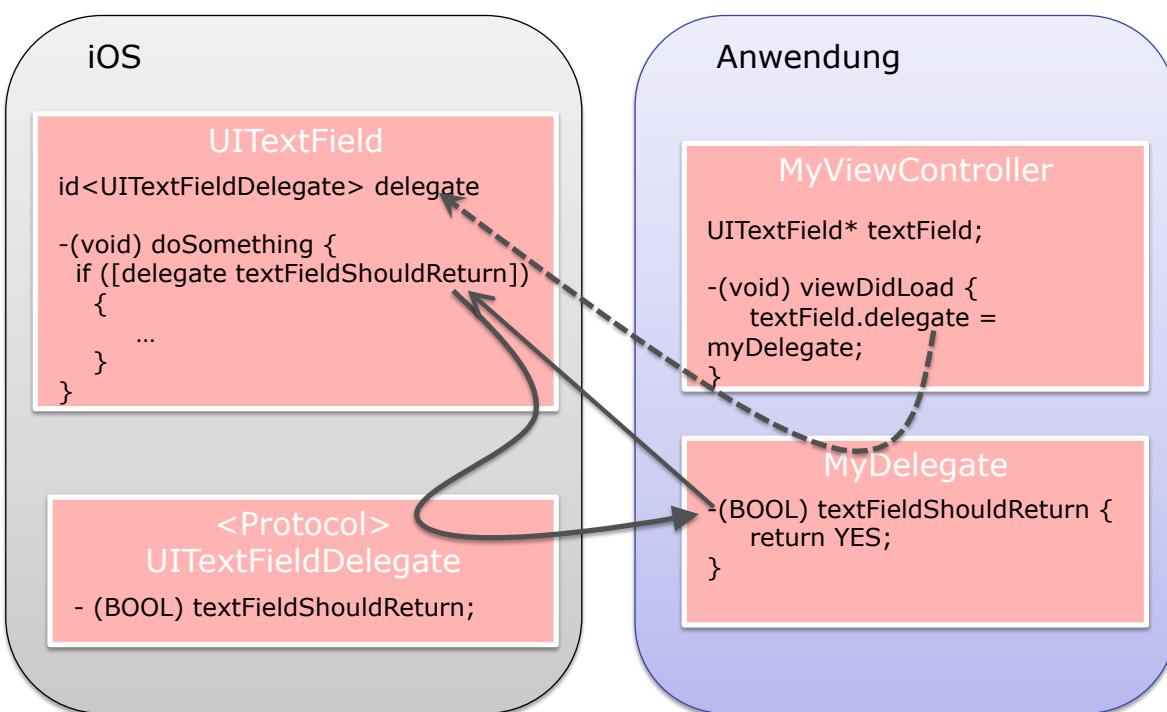
```
@property(nonatomic,retain) IBOutlet UIWebView* webView;  
- (IBAction) openWebSite:(id)sender;
```



Agenda Tag 2

09:00	→ Einführung in iOS-Entwicklung: Delegation iOS Schichten Aufgabe: Erstellen einer TableView
12:00	Pause
13:00	Cocoa in a Nutshell Aufgabe: Verwenden des NavigationControllers
17:00	Ende

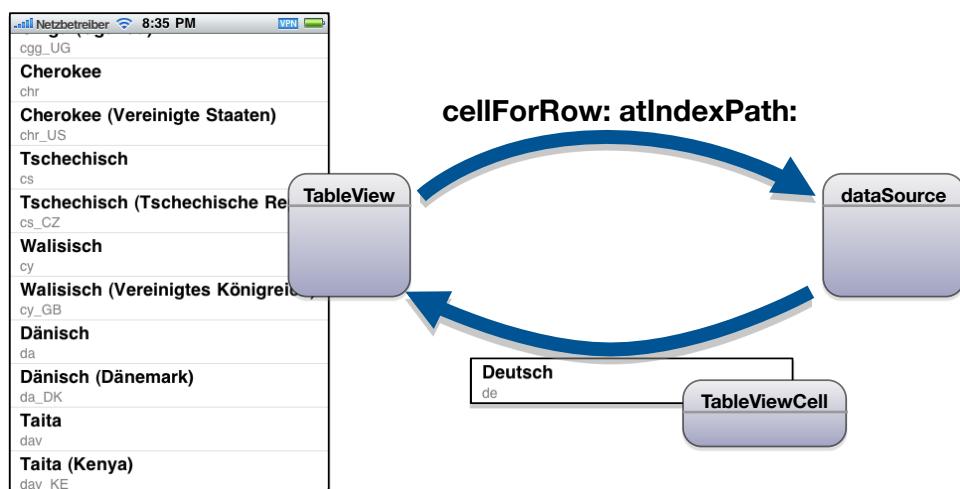
- ◆ Wird an vielen Stellen im iOS eingesetzt
- ◆ Erlaubt die Integration von individuellem Code in Framework-Klassen, ohne Vererbung einzusetzen
- ◆ Wird über Protokolle realisiert
 - Entspricht Interfaces in Java
 - Methoden können optional sein
 - `@interface RSSParser : NSObject<NSXMLParserDelegate>`
- ◆ Beispiele:
 - UITextFieldDelegate
 - UITableViewDelegate
 - UITableViewDataSource
 - NSXMLParserDelegate



- ◆ Vorgefertigte Framework-Funktionen können mit eigenen Inhalten angereichert werden
- ◆ Erben von komplexen Framework-Klassen ist nicht nötig
- ◆ Lediglich das Protokoll muss erfüllt werden
 - Viele Methoden in Protokollen sind optional
→ Fokussierung auf die benötigten Funktionen
- ◆ Meistens implementieren die Controller selbst die benötigten Protokolle
 - Zum Beispiel

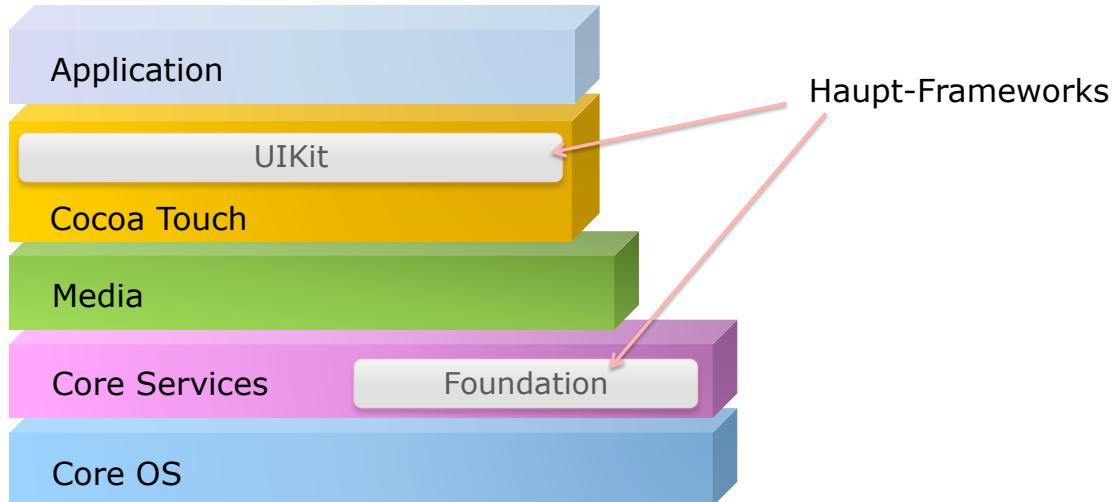
```
@interface UITableViewcontroller : UIViewController
<UITableViewDelegate, UITableViewDataSource>
```

- ◆ UITableViews haben zwei Delegates:
 - ◆ delegate, um auf Aktionen des Benutzers zu reagieren
 - ◆ dataSource, um die darzustellenden TableViewCells zu erhalten



iOS Schichten

- ◆ Vieles schon dabei
 - Look and Feel
 - Bedienkonzepte
 - Animationen
 - Rotation
- ◆ Aufbau der UI mit MVC:
 - UIViews zur Darstellung
 - UIViewController zur Verknüpfung von Modell und View
 - Modell-Klassen vom Entwickler
 - Pro Bildschirmseite ein ViewController



- Core OS
 - ◆ Kernel
 - ◆ Dateisystem
 - ◆ Netzwerk
 - ◆ Security
 - ◆ Treiber

- Core Services
 - ◆ Strings
 - ◆ Collections
 - ◆ Kontakte
 - ◆ Zugriff auf: GPS, Kompass, Beschleunigungssensor, Gyroskop



Media

- ◆ Core Graphics
- ◆ Core Animation
- ◆ OpenGL ES
- ◆ Audio
- ◆ Video



UIKit

Cocoa Touch

- ◆ Map Kit
- ◆ iAd

UIKit

- ◆ Views
- ◆ ViewController
- ◆ Gestenbehandlung
- ◆ Interface Builder zum Erstellen von UIKit-Objekten



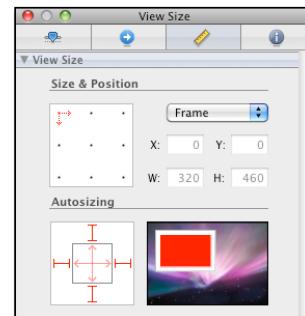
Rotation

- ◆ Eine Methode im ViewController bestimmt, wie rotiert werden kann

```
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)orientation
{
    return YES;
}
```

- ◆ Views müssen sich an neue Größe anpassen:

- Konfiguration im Interface Builder



UIKit

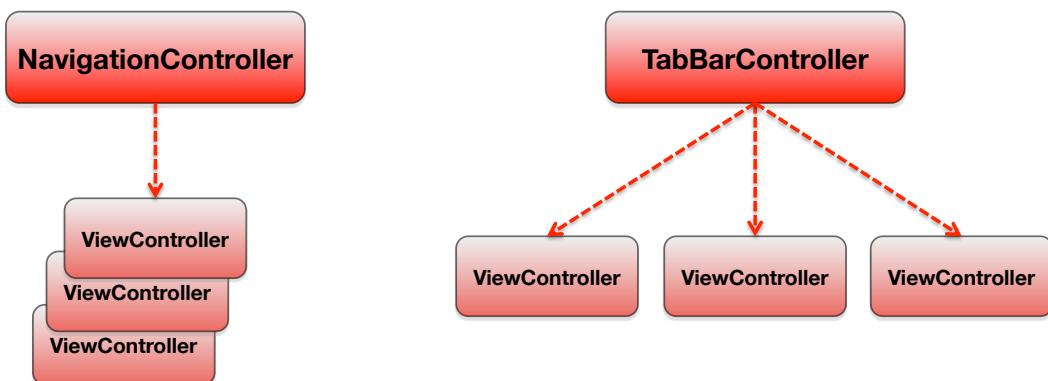
- ◆ Vorgefertigte Controller für
 - Bedienführung in der App (Übersicht zu Details)
 - Tabs
 - Listen
 - Webseiten





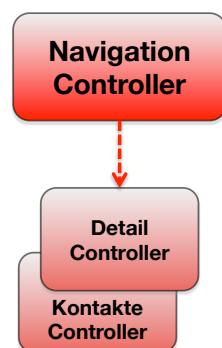
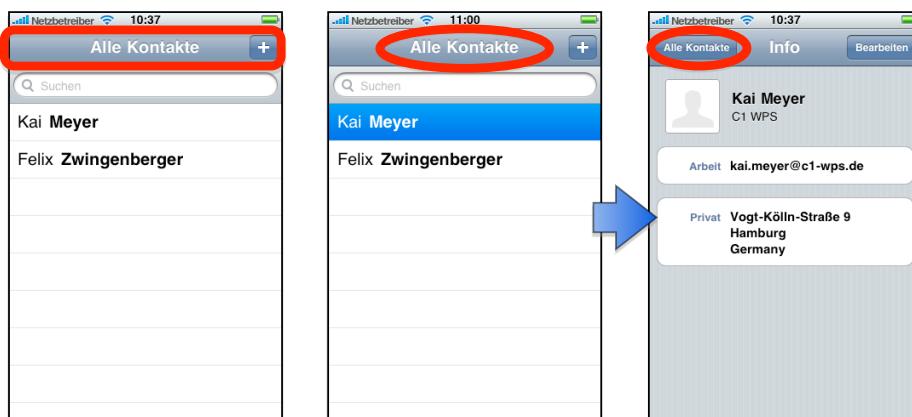
UIKit: ViewController-Hierarchien

- ◆ ViewController können hierarchisch angeordnet werden
 - in speziellen Container-ViewControllern



UIKit: UINavigationController

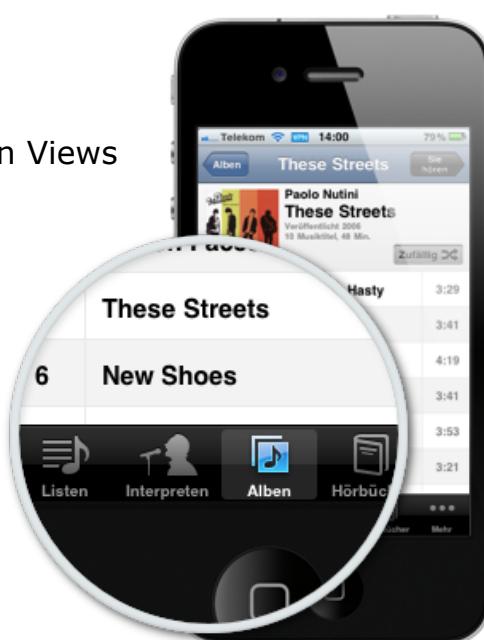
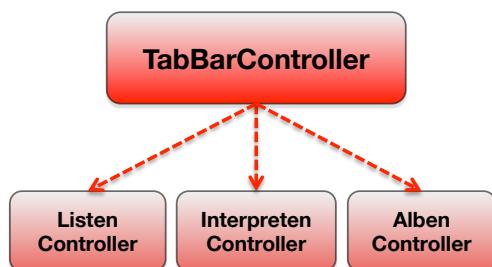
- ◆ Navigation
 - Konsistentes Bedienkonzept
 - Allgemein -> Detailsicht
 - Als Stack angeordnet



- ◆ Navigations-Bildschirme landen auf einem Stack

```
// zum Laden eines neuen ViewControllers  
- (void)pushViewController:(UIViewController *)viewController animated:(BOOL)animated  
  
// Zum Zurücknavigieren (wird z.B. durch Button links oben aufgerufen)  
- (UIViewController *)popViewControllerAnimated:(BOOL)animated
```

- ◆ Tabs
 - Unabhängige Views
 - Controller verwaltet Liste von Views





UIKit: TableView

◆ Listen / Tabellen



© Arbeitsbereich Softwaretechnik

8/26/11

83



UIKit: TableView

Styles:

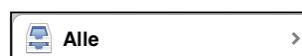
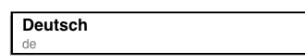
plain



grouped



Zellen:

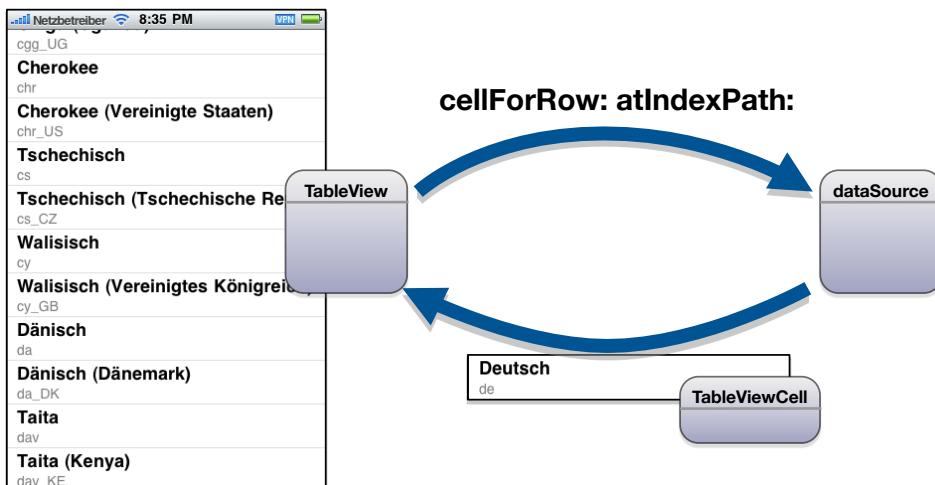


© Arbeitsbereich Softwaretechnik

8/26/11

84

- **UITableViews haben zwei Delegates:**
 - *delegate*, um auf Aktionen des Benutzers zu reagieren
 - *dataSource*, um die darzustellenden TableViewCells zu erhalten



- **UITableViewController implementiert beide Delegate-Protokolle**

- **UITableViewDataSource**

```
// Liefert die Zellen für die einzelnen Tabellen-Zeilen
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)
    *)indexPath

// Anzahl der Zeilen in einem Tabellenabschnitt
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
```

- **UITableViewDelegate**

```
// Wird aufgerufen wenn eine Zeile angetippt wurde
- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath
```



UIKit: UITableView

- **UITableViewCell**
- Aus Performancegründen sollten ihr Zellen wiederverwendet werden

```
// Gibt eine schon vorhandene Zelle für den identifier?  
- (UITableViewCell *)dequeueReusableCellWithIdentifier:(NSString *)identifier
```

- Beispiel:

```
// in tableView:cellForRow:atIndexPath:  
  
UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"standard"];  
  
if(cell == nil) {  
    cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle  
        reuseIdentifier:@"standard"];  
    [returnCell autorelease];  
}  
  
...
```



Aufgabe 3

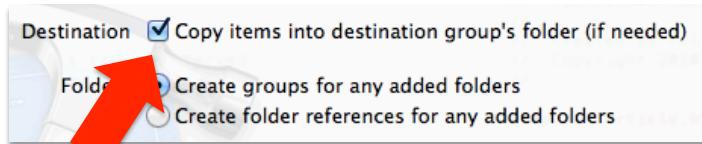
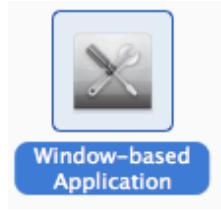
Erstellen einer UITableView

Ziel

- ◆ Eine **TableView** ist zu erstellen, die in ihren Zellen „**Article**“-Objekte aus einem RSS-Feed darstellen kann.

Weg

- ◆ Erstelle ein neues **Window-basiertes Projekt** in Xcode.
- ◆ Zum Befüllen der Zellen deiner Tabelle kannst du ein Objekt der Klasse **RSSParser** verwenden.
 - Bei Übergabe einer **NSURL** an **RSSParser** wird ein Array von „Article“-Objekten zurückliefert.
 - Kopiere die Dateien „RSSParser.h“, „RSSParser.m“, „Article.h“ und „Article.m“ in dein Projekt.
 - Beispiel-Feed-Adressen sind in „**feeds.txt**“ zu finden.



- ◆ Erzeuge deinen **UITableViewController**.
 - Add → New File... → Cocoa Touch Class → UIViewController subclass.
 - Selektiere die Option „UITableViewController subclass“.
 - Lasse dir keine XIB erzeugen.

Hinweis: UIViewController beerbende Klassen sollten im Namen auf „ViewController“ enden.

- ◆ Dein UITableViewController ist **Delegate** und **DataSource** deiner TableView.
- ◆ Setze in deinem **AppDelegate** den RootViewController an dem Window-Objekt deiner App.
 - Erstelle in der Methode „**didFinishLaunchingWithOptions**“ des AppDelegate ein Objekt deines UITableViewControllers.
 - Setze an dem „window“-Objekt in deinem AppDelegate deinen UITableViewController als „rootViewController“.



Aufgabe 3: Erstellen einer TableView

- ◆ Implementiere die **Methoden des Delegate** deiner TableView.
 - Die **DataSource** (ebenfalls ein Delegate) befüllt die darzustellenden Zellen mit Daten.
 - Das **Delegate** reagiert auf die Selektionen in der Tabelle
 - Jede Zelle soll Informationen zu einem „Article“-Objekt des RSS-Feed anzeigen.
 - Der *RSSParser* liefert dir die „Article“-Objekte.



Agenda Tag 2

09:00	iOS Schichten Aufgabe: Erstellen einer TableView
12:00	Pause
13:00	→ Cocoa in a Nutshell Aufgabe: Verwenden des NavigationControllers
17:00	Ende



Wrap up

- ◆ iOS-Apps werden nach dem MVC-Muster aufgebaut
 - Model \Leftrightarrow Eigene Implementation
 - View \Leftrightarrow UIView, UITextField, ...
 - Controller \Leftrightarrow UIViewController
- ◆ Framework bietet bereits
 - Ready-To-Use Klassen: UITableViewController, WebView, UINavigationController
 - Protokolle um eigene Funktionen zu integrieren: UITextFieldDelegate, UITableViewDataSource
- ◆ Gesten-Erkennung bereits im Framework enthalten



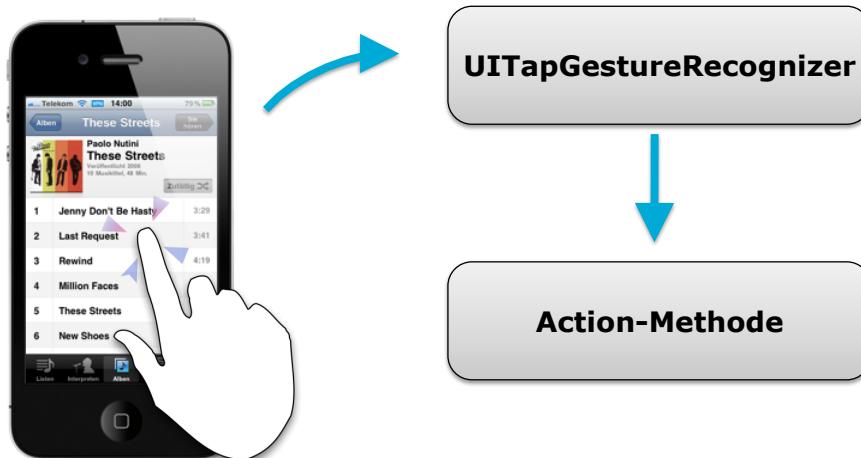
Bedienung

- ◆ Touch-Erkennung
 - Einzelne Finger
 - Bewegung
 - Loslassen
- ◆ Gesten
 - Benachrichtigung für vordefinierte Gesten
 - Tipp
 - Wischen
 - Verschieben
 - Pinch (Zoomen)
 - etc.

- ◆ Gesten können von View-Objekten erkannt werden
 - Viele Gesten sind schon vordefiniert z.B. tippen, streichen, verschieben, rotieren
 - Einheitliche Gestenerkennung in Apps

- ◆ UIGestureRecognizer-Objekte zur Erkennung
 - Subklassen für konkrete Gesten
 - Methode zum Handling implementieren

- ◆ Ablauf einer Tippgesten-Erkennung:



- ◆ Recognizer beim View anmelden

```
// Nachdem der View geladen wurde  
-(void) viewDidLoad {  
    UIPanGestureRecognizer *panRecognizer =  
    [[UIPanGestureRecognizer alloc] initWithTarget:self  
    action:@selector(schiebeGeste:)];  
    [self.view addGestureRecognizer:panRecognizer];  
    [panRecognizer release];  
}
```

- ◆ Action und Target

- ◆ Bei Erkennung der Geste wird der **Action-Selector** am Objekt **target** aufgerufen.

- ◆ Methode zum Gesten-Handling implementieren
- ◆ Recognizer bieten jeweils Methoden zur Behandlung der Geste
- ◆ UIPanGestureRecognizer:

```
- (CGPoint)translationInView:(UIView *)view  
- (CGPoint)velocityInView:(UIView *)view  
- (void)setTranslation:(CGPoint)translation inView:(UIView *)view
```

- ◆ UIRotationGestureRecognizer:

```
@property CGFloat rotation  
@property (readonly) CGFloat velocity
```

- ◆ Diese Methoden und Properties können in der Action-Methode verwendet werden



◆ Beispiel einer Action-Methode:

```
// Verschieben-Geste wurde erkannt
-(void)schiebeGeste:(UIPanGestureRecognizer*)recognizer {
    CGPoint trl = [recognizer translationInView:self.view];
    CGPoint oldCenter = rectView.center;
    rectView.center = CGPointMake(oldCenter.x + trl.x, oldCenter.y + trl.y);
    [recognizer setTranslation:CGPointZero inView:self.view];
}
```



◆ Recognizer haben einen Zustand

```
@property(nonatomic,readonly) UIGestureRecognizerState state;
```

Zustand	Verhalten
possible	Geste noch nicht erkannt, möglich
recognized	Geste erkannt (diskret)
failed	Geste nicht erkannt
began	Geste erkannt (kontinuierlich)
changed	Geste modifiziert (kontinuierlich)
ended	Geste abgeschlossen (kontinuierlich)
cancelled	Geste abgebrochen (kontinuierlich)



- ◆ Vorgefertigte Recognizer für Gesten:

Geste	Recognizer
Tippen	UITapGestureRecognizer
Pinch (Zoomen)	UIPinchGestureRecognizer
Verschieben	UIPanGestureRecognizer
Wischen	UISwipeGestureRecognizer
Rotieren	UIRotationGestureRecognizer
Pressen und halten	UILongPressGestureRecognizer



- ◆ Delegate für Gesten-Recognizer ermöglicht z.B. gleichzeitige Erkennung mehrerer Gesten

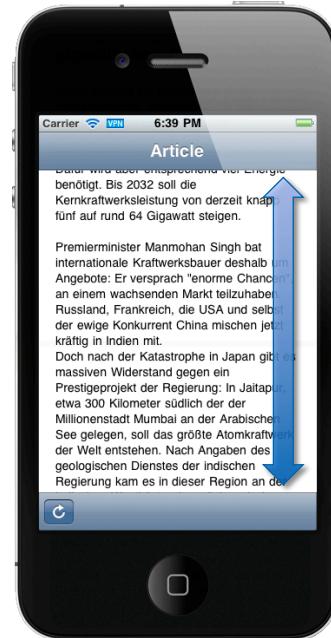
```
// Anmeldung als Delegate, z.B. in viewDidLoad...
panRecognizer.delegate = self;
```

```
// Implementation der Delegate-Methode
-(BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
    shouldRecognizeSimultaneouslyWithGestureRecognizer:
        (UIGestureRecognizer *)otherGestureRecognizer
{
    return YES;
}
```



UIKit: ScrollView

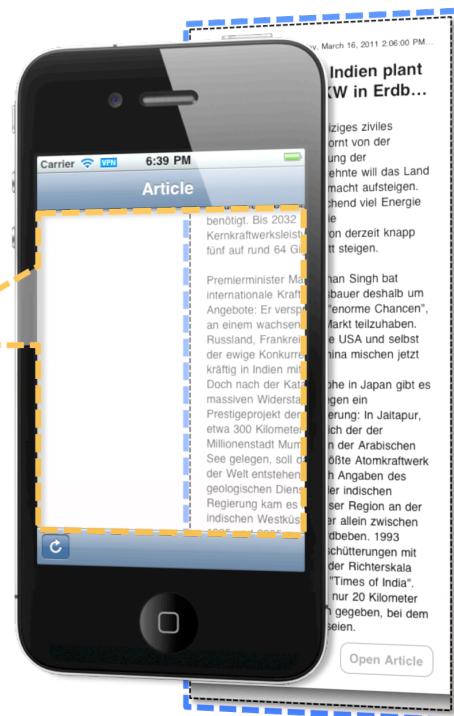
- ◆ Ermöglicht das Scrollen von Views
 - wenn Views zu groß sind
 - wenn die Tastatur eingeblendet wird
- ◆ ScrollViews ermöglichen auch zoomen
 - mit Pinch-Geste



UIKit: ScrollView

scrollView.size

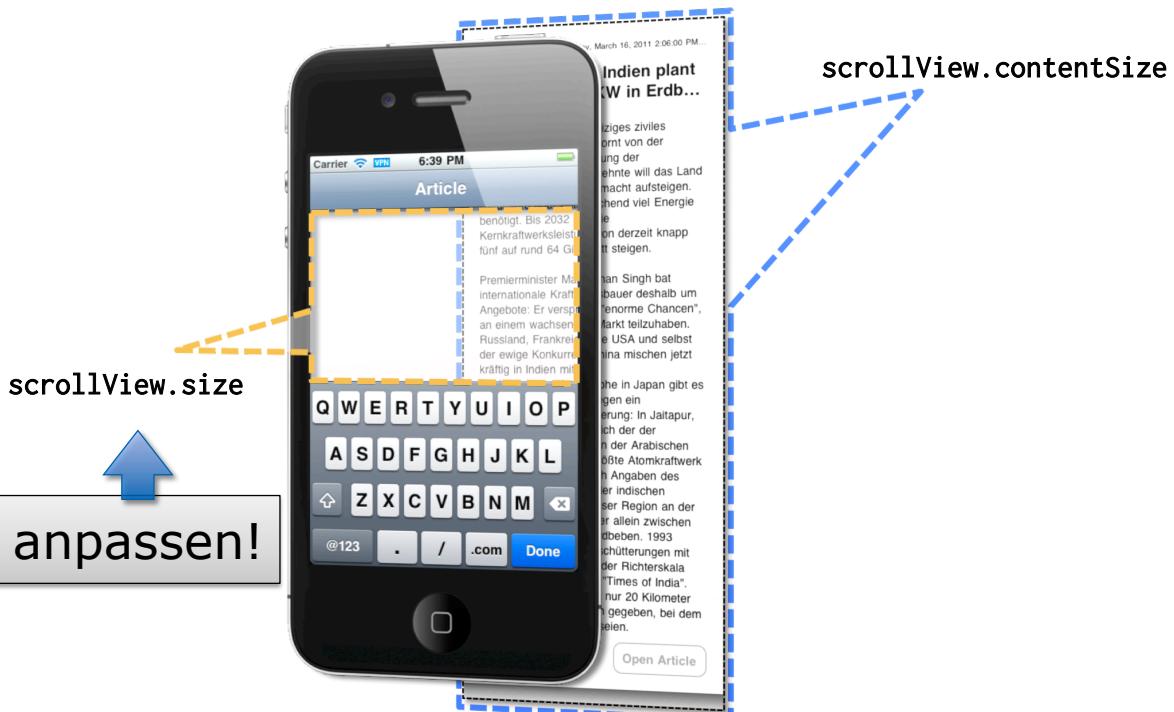
scrollView.contentSize



- ◆ Anzeigen / Verbergen der Tastatur
 - wird automatisch eingeblendet, wenn ein Texteingabefeld Fokus erhält
 - wird **nicht** automatisch ausgeblendet

```
// Tastatur ausblenden
-(IBAction)doneButtonPressed:(id)sender {
    [textField resignFirstResponder];
}
```

- ◆ Tastatur kann Teile der UI verdecken
 - Ggf. muss die UI gescrollt werden
 - Views können in UIScrollView eingebettet werden
 - ❖ Anpassen der Größe des Scrollviews bei Einblendung der Tastatur



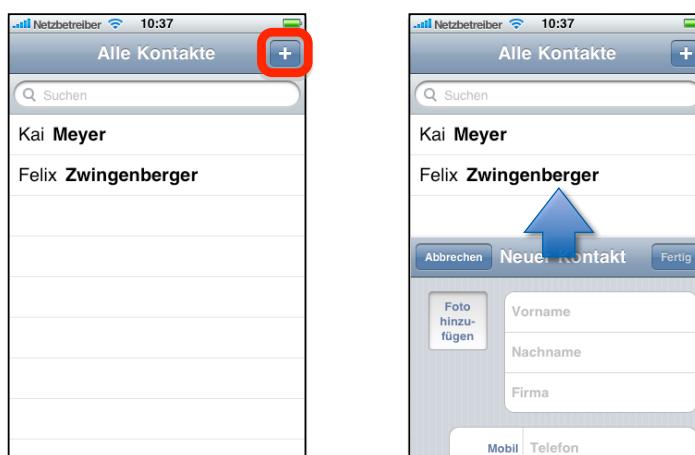
◆ Anpassen des Views

1. Anmelden für Keyboard-Notifications
2. Anpassen der Scrollview-Größe an Tastatur-Einblendung
3. Scrollen des Views (z.B. mit scrollRectToVisible:animated:)

◆ Anleitung als Foliensatz



```
- (void) presentModalViewController:(UIViewController *)modalViewController  
animated:(BOOL)animated  
  
- (void) dismissModalViewControllerAnimated:(BOOL)animated
```



Aufgabe 4

Verwenden des NavigationControllers

Ziel

- ◆ Die **TableView App.** ist zu erweitern.
 - Durch tippen auf einen Artikel in der TableView soll eine weitere View **Details** zu dem Artikel anzeigen.
 - ❖ Titel, Text und das Datum des Artikels sollen angezeigt werden.
 - ❖ Ein Button soll die URL zu dem Artikel in einer **WebView** öffnen.
 - ❖ Über Back-Buttons soll zur jeweils vorigen View **navigiert** werden.
 - Ein Button in der TableView soll eine **ModalView** öffnen können, in der die RSS-Feed Adresse eingegeben werden kann.



Aufgabe 4: Verwenden des NavigationControllers

Weg

- ◆ Erzeuge in der Methode „**didFinishLaunchingWithOptions**“ deines AppDelegate einen UINavigationController.
 - Setze den UINavigationController anstelle deines TableViewController als „rootViewController“ an dem „window“ deiner App.
 - Push deinen TableViewController auf den UINavigationController.



Aufgabe 4: Verwenden des NavigationControllers

- ◆ Erzeuge einen **neuen UIViewController** mit XIB zum Anzeigen der Details eines Artikels - im folgenden genannt **DetailViewController**.
 - Der DetailViewController soll durch Tippen auf eine Zelle in deiner TableView zu öffnen sein.
 - **Nutze die Vorlage** in der Methode „*didSelectRowAtIndexPath*“ deines TableViewController.
 - ❖ In der Vorlage heißt die DetailViewController-Klasse „ViewController“. Denke daran ggf. auch den String für die Xib-Datei zu ändern.
- ◆ Erzeuge einen **WebViewController** (mit Xib) zum öffnen der URL eines Artikels.
 - Der WebViewController ist ein ViewController mit **UIWebView**.
 - Der WebViewController sollte das **<UIWebViewDelegate>** Protokoll implementieren.

- ◆ Füge deiner Detail-View einen **Button zum Anzeigen der WebView** hinzu.
 - Push zum Anzeigen der WebView den ViewController auf den NavigationController.
 - Teile der **UIWebView** mit, welche URL geöffnet werden soll.
 - Hinweis: Jeder ViewController besitzt automatisch eine Referenz auf den NavigationController.
- ◆ Erzeuge einen neuen ViewController zum eingeben der RSS-Feed-Adresse. Im folgenden genannt „**FeedSelectionViewController**“.
 - Der RSS-Feed soll über ein Textfeld einzugeben sein.
 - Lasse diesmal die Tastatur verschwinden, indem du das „**UITextFieldDelegate**“-Protokoll verwendest.

- ◆ Füge der NavigationBar deines NavigationControllers ein „**UIBarButtonItem**“ hinzu (programmatisch). Dieser Button soll den **FeedSelectionViewController öffnen**.
 - Erzeuge ein neues UIBarButtonItem-Objekt
 - Setze es als „leftBarButtonItem“ dem „navigationItem“ deines TableViewController.
 - Schreibe eine Methode als „action“ für den Button.
 - Setze „target“ und „action“ am UIBarButtonItem.
 - ❖ Erzeuge den **FeedSelectionViewController** in der „action“.
 - ❖ Öffne in der „action“ den FeedSelectionViewController als **ModalViewController**. („presentModalViewController“ und „dismissModalViewController“).
- **Definiere ein Protokoll**, damit du z.B. deinen TableViewController als Delegate an den FeedSelectionViewController übergeben kannst.
 - ❖ Erzeuge dazu eine neue Datei vom Typ **Objective-C Protocol**
 - ❖ Nutze eine im Protokoll definierte Methode, um dem Delegate eine **neue RSS-Feed-Adresse zu übergeben**.
 - ❖ Das **Delegate soll den neuen RSS-Feed laden** und die TableView mit „reloadData“ aktualisieren.



Tag 1: Einführung in Objective-C
 Einführung in iOS-Entwicklung

Tag 2: iOS Schichten
 Cocoa in a Nutshell

Tag 3: **Objective-C Teil 2**
Testen
Deployment
Wrap-Up und Ausblick



- ◆ Bedienkonzepte von iOS
 - UINavigationController: Für Detailierung
 - UITabBarController: Unabhängige Views
 - ModalViewController: „Modale“ Views
- ◆ Ausblenden von Tastaturen durch „resignFirstResponder“
 - Über IBAction und Event „Did End on Exit“
 - Über UITextFieldDelegate und
 - `(BOOL)textFieldShouldReturn:(UITextField *)textField`



Agenda Tag 3

- 09:00 → Objective-C Teil 2
Testen
Deployment
Aufgabe: Speichermanagement
- 12:00 Pause
- 13:00 → iPad
Wrap-Up, Ausblick und Aufgabenstellung
Aufgabe: Verwenden des SplitViewControllers
- 17:00 Ende



Objective-C Teil 2

- ◆ Keine Garbage Collection auf iPhone und iPad
- ◆ Jedes Objekt hat einen "retain count"
 - Zähler für Referenzen
- ◆ Interesse an Objekt: **retain** aufrufen

```
[objekt retain]; // retain count + 1
```

- ◆ Kein Interesse mehr an Objekt: **release** aufrufen

```
[objekt release]; // retain count - 1
```

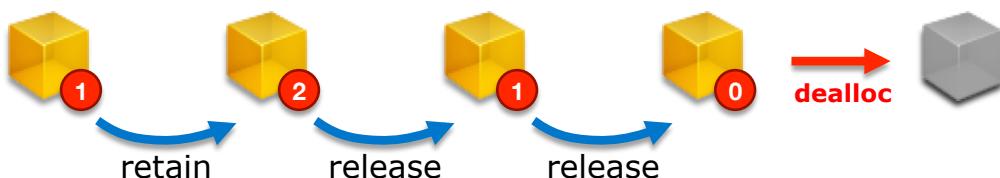
- ◆ Erreicht ein Objekt retain count == 0, wird es automatisch dealloziert
- ◆ Objekte starten mit retain count 1
 - wenn mit **alloc**, **new** oder **copy** erstellt
 - ansonsten nicht

◆ Beispiel

```
konto = [[Konto alloc] init];
[person setKonto:konto];
[konto release];
```

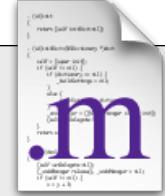
```
Person.m
-(void) setKonto:(Konto*) konto
{
  ...
  [konto retain];
}

-(void) dealloc
{
  ...
  [konto release];
}
```



◆ Destruktor-Implementation in Objective-C:

```
- (void) dealloc {  
    [name release];  
    [text release];  
    [super dealloc];  
}
```



1. Überschreiben von -(void)dealloc;
 2. Freigabe aller bis dahin gehaltenen Objekte mit release
 3. Anschließend Aufruf von [super dealloc];
- ◆ dealloc niemals selbst aufrufen
- (außer [super dealloc];)

◆ Verzögern von **release**

```
-(Kurs *) seminar {  
    Kurs *derKurs = [[Kurs alloc] init];  
    return derKurs;           ← Retain count ist 1!  
}
```

◆ **Problem: Wir sollten release aufrufen**

```
-(Kurs *) seminar {  
    Kurs *derKurs = [[Kurs alloc] init];  
    return derKurs;  
    [derKurs release];       ← zu spät!  
}
```



Autorelease (2)

```
- (Kurs *) seminar {
    Kurs *derKurs = [[Kurs alloc] init];
    [derKurs release];           ← zu früh!
    return derKurs;
}
```

Das Objekt wird schon dealloziert

◆ Lösung: autorelease

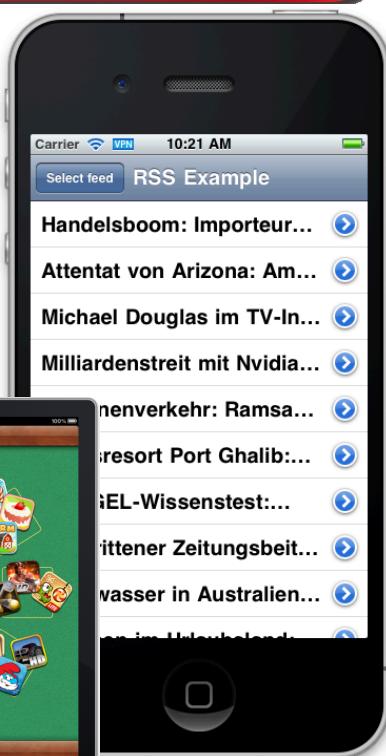
```
- (Kurs *) seminar {
    Kurs *derKurs = [[Kurs alloc] init];
    [derKurs autorelease];      ← wird „demnächst“ dealloziert
    return derKurs;
}
```



Testen

- ◆ Verschiedene Ebenen für Tests:
 - Simulator
 - Unit Tests
 - Application Tests
 - Analyse von Speicherlecks
 - Test auf realem Gerät
- ◆ Xcode beinhalten bereits ein Test-Framework für
 - Unit Tests
 - Application Tests
- ◆ Instruments ist für die Analyse von Speicherlecks

- ◆ Schnelles Deployment
- ◆ Ohne Lizenz nutzbar
 - im Gegensatz zu einem echten Gerät
- ◆ Nicht geeignet für Performance-Tests
- ◆ Kein Ersatz für Entwicklung mit Test-Gerät
- ◆ Simulation von 2-Finger-Gesten möglich
- ◆ Rotation + "Schütteln" simuliert



- ◆ Tests werden ähnlich wie in JUnit aufgebaut
- ◆ Testen die Funktion einer Klasse
- ◆ Keine UI-Tests oder Tests der IBOutlets und IBActions
- ◆ Können automatisch bei jedem Start der Anwendung ausgeführt werden
- ◆ Benötigen ein eigenes Target
 - „Unit Test Bundle“ dient als Template
- ◆ Für die Tests sollte das Template „Objective-C test case“ verwendet werden

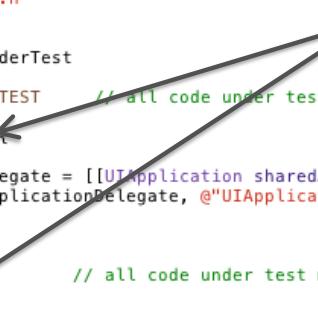
```
//  
// TaskProviderTest.h  
// TaskViewer  
//  
// Created by Kai Meyer on 11.03.11.  
// Copyright 2011 C1 WPS. All rights reserved.  
//  
// See Also: http://developer.apple.com/iphone/library/documentation/Xcode/Conceptual/iphone  
  
// Application unit tests contain unit test code that must be injected into an application t  
// Define USE_APPLICATION_UNIT_TEST to 0 if the unit test code is designed to be linked into  
  
#define USE_APPLICATION_UNIT_TEST 1  
  
#import <SenTestingKit/SenTestingKit.h> ← Bezug zum Test-Frameworks  
#import <UIKit/UIKit.h>  
//#import "application_headers" as required  
  
@interface TaskProviderTest : SenTestCase {  
}  
  
#if USE_APPLICATION_UNIT_TEST  
- (void) testAppDelegate; // simple test on application  
#else  
- (void) testMath; // simple standalone test  
#endif  
  
@end
```



Test Case Implementation

```
//  
// TaskProviderTest.m  
// TaskViewer  
//  
// Created by Kai Meyer on 11.03.11.  
// Copyright 2011 C1 WPS. All rights reserved.  
  
#import "TaskProviderTest.h"  
  
@implementation TaskProviderTest  
  
#if USE_APPLICATION_UNIT_TEST // all code under test is in the iPhone Application  
- (void) testAppDelegate {  
    id yourApplicationDelegate = [[UIApplication sharedApplication] delegate];  
    STAssertNotNil(yourApplicationDelegate, @"UIApplication failed to find the AppDelegate");  
}  
  
#else // all code under test must be linked into the Unit Test bundle  
- (void) testMath {  
    STAssertTrue((1+1)==2, @"Compiler isn't feeling well today :-(" );  
}  
  
#endif  
  
@end
```

Test-Methoden

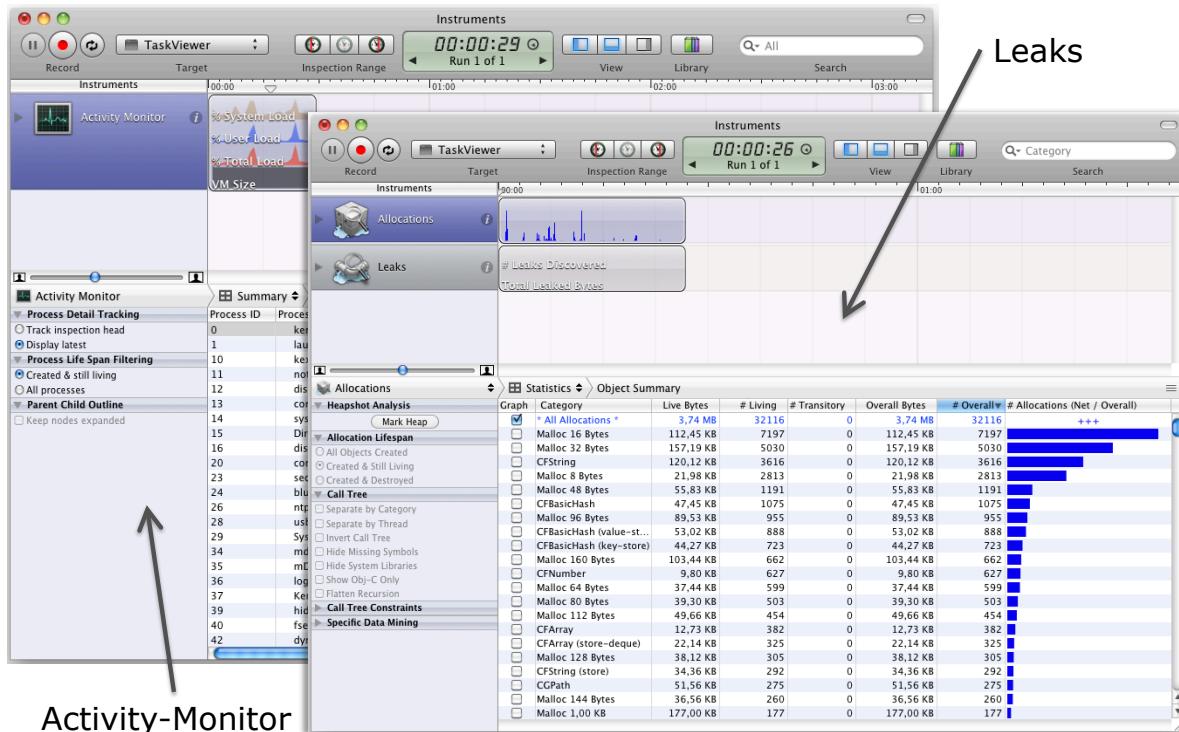


Application Tests

- ◆ Testen das gesamte Verhalten der Anwendung
 - Inklusive UI, IBOutlets und IBActions
 - ◆ Tests werden auf die gleiche Weise implementiert
-
- ◆ Konfiguration der Tests ist aufwendiger
 - ◆ Application Tests sind nur auf einem echten Gerät ablauffähig



Instruments



© Arbeitsbereich Softwaretechnik

8/26/11

131



Instruments

- ◆ Zombies
- ◆ Threads
- ◆ Multicore
- ◆ GC Monitor
- ◆ File Activity
- ◆ Core Data
- ◆ Core Animation
- ◆ Leaks
- ◆ CPU Sampler
- ◆ Allocations
- ◆ Activity Monitor
- Analyse von „over-released“ Objekten
- Anzahl und Status von Threads
- MultiCore Performance
- Garbage Collector
- Arbeit mit dem Dateisystem
- Core Data Aufrufe
- Animationen (fps)
- Speicherlecks
- CPU-Auslastung der Anwendung
- Alloc-Aufrufe
- Virtueller Speicher

© Arbeitsbereich Softwaretechnik

8/26/11

132

Deployment

Deployment

- ◆ Es wird eine (kostenpflichtige) Lizenz benötigt
 - Standard: 99 \$ / Jahr
 - Enterprise: 299 \$ / Jahr
 - University: für Universitäten kostenfrei
- Testbetrieb auf bis zu 100 Geräten möglich
- Vertrieb über den „App Store“ (iTunes)
 - Außer bei der Enterprise-Lizenz
 - Verteilung der App über eigenen Server möglich
- Anwendungen können kostenfrei oder kostenpflichtig angeboten werden
 - Apple bekommt 30% des Verkaufspreises





Test auf Geräten

- ◆ Voraussetzungen für Tests auf Geräten sind:
 - Development Certificate ist für den Entwickler vorhanden
 - Testgeräte sind registriert
 - App-ID ist erstellt
 - Provisioning Profil ist für die App-ID und die Testgeräte erstellt
 - Ein Testgerät ist an dem Entwicklungsrechner angeschlossen
- ◆ Über Xcode kann mit dem Provisioning Profil der Code kompiliert und auf das Testgerät übertragen werden



Distribution

- ◆ Für die Veröffentlichung muss ein Distribution Profil erstellt werden
- ◆ Die kompilierte App wird von Apple ausführlich getestet bevor sie in den App-Store hochgeladen wird
- ◆ Verwaltung der Zertifikate, Profile und die Registrierung der Testgeräte wird im iOS Provisioning Portal vorgenommen
- ◆ Das Provisioning Portal ist über das iOS Dev Center erreichbar
<http://developer.apple.com/iphone/>

Aufgabe 5

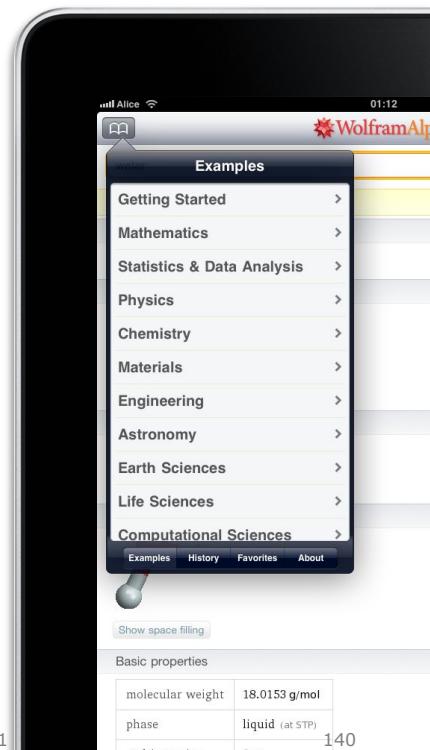
Speichermanagement

Aufgabe 5: Speichermanagement

- ◆ Die RSS-Reader App. soll auf **Speicherlecks** überprüft werden und korrektes Speichermanagement bekommen.
- ◆ Setze **Instruments** ein, um Speicherlecks ausfindig zu machen.
- ◆ Analysiere deine App in Xcode mit **[cmd]+[shift]+[b]**
- ◆ Vermeide Speicherlecks durch den Einsatz von
 - *dealloc*,
 - *release*,
 - *autorelease*,
 - und *properties*.

- | | |
|-------|----------------------------------------------------|
| 09:00 | Objective-C Teil 2 |
| | Testen |
| | Deployment |
| | Aufgabe: Speichermanagement |
| 12:00 | Pause |
| | |
| 13:00 | → iPad |
| | Wrap-Up, Ausblick und Aufgabenstellung |
| | Aufgabe: Verwenden des SplitViewControllers |
| 17:00 | Ende |

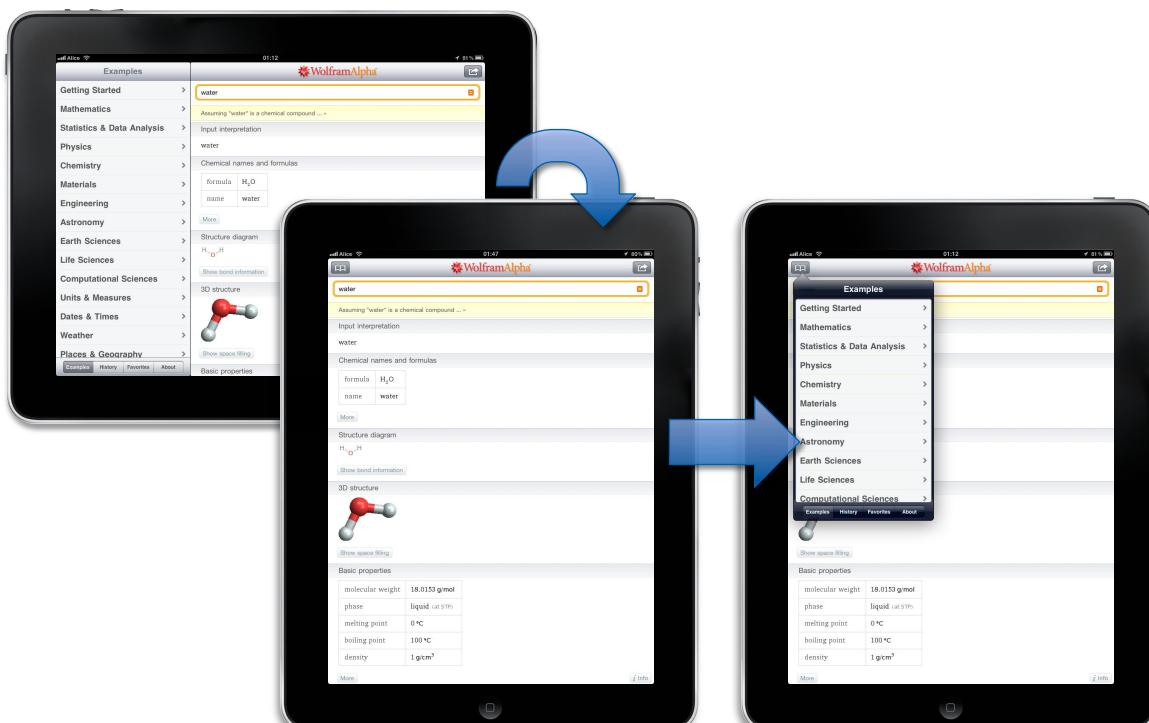
- ◆ Zeigt einen ViewController als Popup auf dem Bildschirm an
- ◆ Ist selbst kein ViewController
- ◆ Wird verwendet:
 - ◆ für Zusatzinformationen,
 - ◆ für Werkzeuge,
 - ◆ zur Anzeige von linkem ViewController bei SplitViews in Portrait-Ansicht



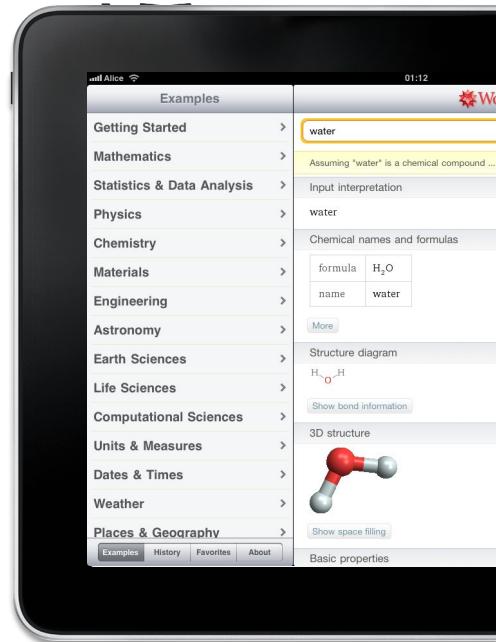
◆ Verwendung von UIPopoverController:

```
- (IBAction)popupButtonPressed:(id)sender
{
    MyViewController* content = [[MyViewController alloc] init];
    UIPopoverController* pController = [[UIPopoverController alloc]
        initWithContentViewController:content];
    pController.delegate = self;
    [content release];

    [pController presentPopoverFromBarButtonItem:sender
        permittedArrowDirections:UIPopoverArrowDirectionAny
        animated:YES];
}
```



- ◆ Verwaltet 2 ViewController:
 - ◆ Master (linke Seite)
 - ◆ Detail (rechte Seite)
- ◆ Muss ganz oben in der View-Hierarchie stehen
 - ◆ nicht in UINavigationController / TabBarController

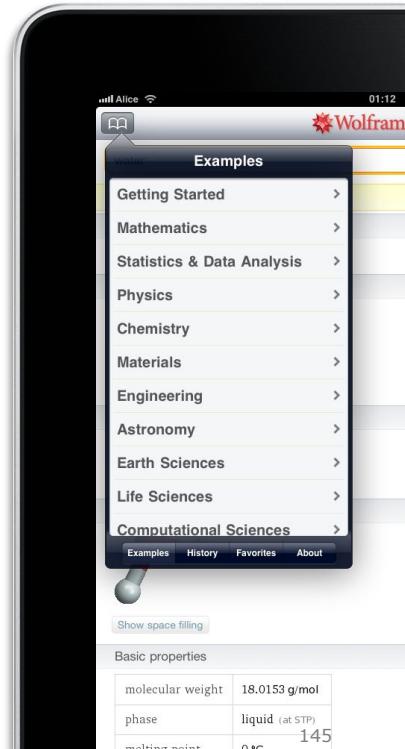


- ◆ Verwendung von UISplitViewController:

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {  
    ...  
    MasterViewController* masterVC;  
    DetailViewController* detailVC;  
    UISplitViewController* svc = [[UISplitViewController alloc] init];  
    svc.controllers = [NSArray arrayWithObjects:masterVC, detailVC, nil];  
    [window addSubview:[svc view]];  
}
```

- ◆ Master-View wird im Portrait-Modus ausgeblendet
- ◆ SVC informiert seinen Delegate über Rotation
- ◆ SVC übergibt dem Delegate einen Button + PopoverController zur Anzeige des Master-Views
 - ◆ Popover-Button für Master kann in die Navigationsleiste vom Detail-View
 - ◆ Deshalb DetailController am besten in NavigationController einbetten

```
// UISplitViewControllerDelegate
-(void)splitViewController:(UISplitViewController *)svc
    willHideViewController:(UIViewController *)aViewController
    withBarButtonItem:(UIBarButtonItem*)barButtonItem
    forPopoverController:(UIPopoverController*)pc;
```



- ◆ Die Sprache Objective-C
 - ◆ Klassen-Definition, Property-Definition, Speichermanagement
- ◆ Das iOS und die Tools
 - ◆ Xcode, Interface Builder, Simulator, Instruments
- ◆ MVC Pattern
 - ◆ UIView ⇔ UIViewController ⇔ Modell
- ◆ Verbindungen zwischen UI und Code
 - ◆ IBOutlets und IBAction
- ◆ Bedienkonzepte
 - ◆ UINavigationController, UITabBarController, ModalViewController
- ◆ Gesten-Erkennung
 - ◆ Pinch, Tab, Swipe, Pan, ...
- ◆ Testen und Deployment



Was haben wir nicht behandelt

- ◆ iOS

- Animation
- Zugriff auf Beschleunigungssensoren, Gyroskop, Kamera und Kontakte
- URL-Schemas
- Umgang mit Open GL
- Push-Benachrichtigungen

- ◆ Objective-C

- Definition von Protokollen
- Categories
- Blocks



Wo geht es weiter?

- ◆ iOS Dev Center:

<http://developer.apple.com/devcenter/ios/index.action>

- ◆ Beispiel-Programme (erreichbar über Xcode Hilfe)

- ◆ Developing Apps for iOS (Stanford)

<http://itunes.stanford.edu/>

Aufgabe 6

Verwenden des SplitViewControllers



Aufgabe 6: Verwenden des SplitViewControllers

Ziel:

- ◆ Eine SplitView ist zu erstellen, um die RSS-Reader App. für den Einsatz auf dem iPad anzupassen.

Weg:

- ◆ Erstelle ein neues iPad spezifisches Projekt und kopiere deine Dateien dort hinein.
 - Die XIB-Dateien haben noch das iPhone-Format und können leider nicht weiter verwendet werden.
 - Erstelle zu jeder XIB-Datei eine neue XIB-Datei im iPad-Format.
 - Kopiere die UI-Elemente aus der alten XIB-Datei in die neue, passe ihre Größe und Position an.
 - Setze im InterfaceBuilder die Klasse des Files Owner.
 - Verbinde die UI-Elemente erneut mit den IBOutlets und IBActions des Files Owner, insbesondere auch die View.

- ◆ Verwende als obersten ViewController den SplitViewController.
 - Die Feed-Tabelle und Details sollen gleichzeitig dargestellt werden.
 - Überlege, wie der UINavigationController weiterhin sinnvoll eingesetzt werden kann.
 - Überlege, wie sinnvoll mit der Rotation umgegangen werden kann.

Hinweis: Es ist sinnvoll, in dem SplitViewController sowohl auf der Linken, als auch auf der rechten Seite einen UINavigationController zu haben.

- ◆ Der SplitViewController kann eine Delegation-Objekt haben, dem er bei Rotation in den Portrait-Mode einen fertigen Button für die Einbindung in die Toolbar der rechten Seite schickt.
 - Implementiere das Protokoll für dieses Delegate in deinem DetailViewController.
 - Setze deinen DetailViewController als Delegate an dem SplitViewController.
 - Binde den Button ein.

**Vielen Dank für
Ihre Aufmerksamkeit und Ihr Interesse**



**Feedback, Fragen und Anregungen
sind willkommen**