



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



Praktikum iOS-Entwicklung

Sommersemester 2017

Prof. Dr. Linnhoff-Popien

Lenz Belzner, Kyrill Schmid





LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



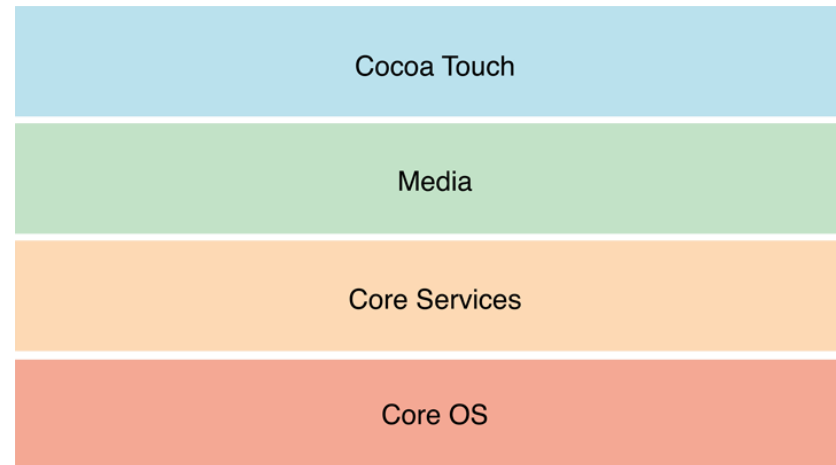
 mobile and
distributed systems group



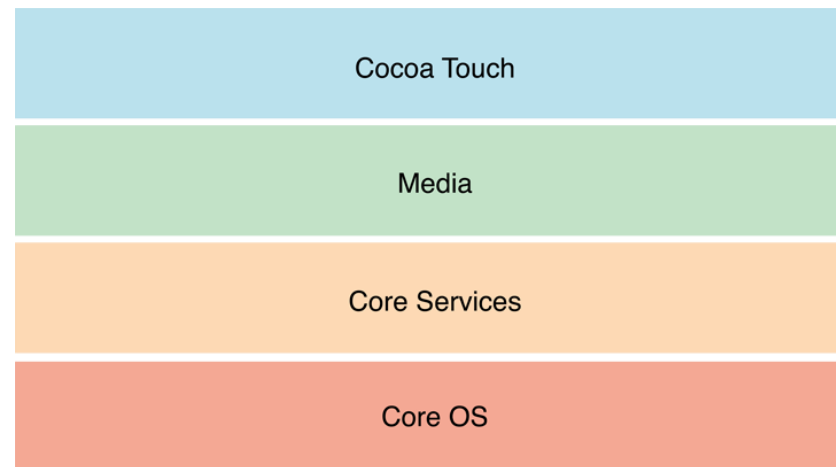
iOS

iOS:

- Betriebssystem das auf iPad, iPhone und iPod Touch Geräten läuft
- iOS verwaltet Hardware und stellt die Schnittstelle zur App-Entwicklung dar
- iOS Software Development Kit (SKD) enthält Tools und Interfaces für (native) App-Entwicklung
- Layer Architektur:



- Untere Layer enthalten low-level Funktionalität (Netzwerk, Security etc.)
- Obere Layer bauen auf low-level Services auf und bieten objektorientierte Abstraktionen
- Empfehlung: Bevorzugung von high-level Frameworks wann immer möglich



iOS Layer setzen sich aus Menge von Frameworks zusammen

Wichtige Frameworks des CocoaTouch-Layers:

- UI Kit
 - Verwaltung der App und der Ressourcen
 - User Interface Management
 - Repräsentation der standard Views und Controller
- MapKit, Message UI, Notification Center

Media Layer :

- Graphics, Audio und Video

Core Services Layer:

- Stellt grundlegende System Dienste für Apps bereit
 - Peer-to-Peer Services (Multipeer Connectivity Framework)
 - iCloud Storage
 - Datenschutz
 - SQLite
 - Foundation Framework
 - Collection data types, String management, Threads und Run-Loops...

Core OS Layer:

- low-level Features bereit
- Meist keine direkte Verwendung der Frameworks zur App-Entwicklung
- Core Bluetooth, Security Services, System...

PROGRAMMIERSPRACHEN UNTER IOS: OBJECTIVE-C UND SWIFT

Zunächst wurde für iOS hauptsächlich in **Objective-C** entwickelt.

Auf der WWDC 2014 wurde die bis dahin unbenutzte Sprache **Swift** vorgestellt.

Aktuelle Version ist Swift 3.1

Code in **C** und **C++** wird von iOS genauso unterstützt.

Und was machen wir?

Hi Michael,

[...] Swift is a new option for developing on the platform. We have no plans to drop C, C++ or Objective-C. If you're happy with them, please feel free to keep using them.

-Chris

<http://lists.apple.com/archives/xcode-users/2014/Jun/msg00024.html>

Verfügbare Apps im iOS-AppStore (Stand März 2017):

ca. 2.300.000

Gesamtanzahl der App-Downloads:

> 140.000.000.000

Großteil dieser Apps sind in Objective-C, C oder C++ entwickelt worden.

In dieser Veranstaltung werden wir uns deshalb mit beiden Sprachen beschäftigen.



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



GRUNDLAGEN IN OBJECTIVE-C

Wikipedia:

- „Objective-C is a general-purpose, high-level, object-oriented programming language that adds Smalltalk-style messaging to the C programming language“.

Objective-C basiert auf C.

- Jedes C Programm kann mit einem Objective-C-Compiler kompiliert werden. Die Objective-C LLVM erlaubt sogar überall die Verwendung von C und C++!
- Objective-C erweitert C um Objektorientierung.
- Methodenaufrufe erfolgen mittels **message passing**

Swift und Objective-C können zusammen im selben Projekt verwendet werden!
(s. Hausaufgabe)



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Objective-C: Was ist das?

 mobile and
distributed systems group



DEMO

Struktur der Objekte wird in Klassen definiert

- Üblicherweise werden zwei Files angelegt (.h und .m)
- .h-File: Deklaration des (öffentlichen) Klassen-Interface
- .m-File: (private) Implementierung

Methodensignaturen müssen innerhalb einer Klasse eindeutig sein!

- Signaturen müssen sich in den Argumenten unterscheiden und nicht nur in den Rückgabewerten

Objekte haben Properties

- `@property (nonatomic) typ name;`
- Implizite Deklaration von getter und setter im Interface
- Und: implizite Deklaration von Instanzvariablen (Prefixed: `_name`)!
- Alle Objekttypen müssen als Pointer (`*name`) angegeben werden

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

@interface Person : NSObject // Person erbt von NSObject
{
    // alle Instanzvariablen gehören in diesen Block
    @property int alter;
    //@property implizit. Möglich: @protected, @private (analog zu Java)

    float groesse;
}

// alle Methoden außerhalb der geschweiften Klammern
- (void)setAlter:(int)neuesAlter;
- (int)alter;
- (void)setGroesse:(float)neueGroesse;
- (float)groesse;
- (void)setAlter:(int)neuesAlter undGroesse:(float)neueGroesse;

@end
```

```
#import "Person.h"
@implementation Person

- (void)setAlter:(int)neuesAlter {
    alter = neuesAlter;
}

- (int)alter {
    return alter;
}

- (void)setGroesse:(float)neueGroesse {
    groesse = neueGroesse;
}

- (float)groesse {
    return groesse;
}

- (void)setAlter:(int)neuesAlter undGroesse:(float)neueGroesse{
    alter = neuesAlter;
    groesse = neueGroesse;
}
@end
```



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

@interface Person : NSObject

@property (nonatomic, readwrite) int alter;
@property (nonatomic, readwrite) float groesse;

/**
 * nonatomic: Sollte immer angegeben werden.
 * readwrite (standard): erzeugt getter UND setter
 * readonly: erzeugt nur getter
 */

- (void)setAlter:(int)neuesAlter undGroesse:(float)neueGroesse;

@end
```

```
#import "Person.h"
```

```
@implementation Person
```

```
@synthesize alter, groesse; //kann i.d. Regel weggelassen werden
```

```
- (void)setAlter:(int)neuesAlter undGroesse:(float)neueGroesse{  
    alter = neuesAlter;  
    groesse = neueGroesse;  
}  
@end
```

Statische Variablen werden mit `static` deklariert.

Statische Variablen 'leben' nicht auf dem Stack und werden beim Verlassen der Methode nicht gelöscht.

Auf `theAnswer` kann nur aus der Methode `doSomethingWeird` heraus zugegriffen werden.

Mit statischen Variablen kann das Singleton-Pattern realisiert werden.

```
+ (int)doSomethingWeird
{
    static int theAnswer;
    if (!theAnswer)
        theAnswer = 42;
    return theAnswer;
}
```

```
// MyCoolObject.m

import "MyCoolObject.h"

@interface MyCoolObject()
// someValue ist nur innerhalb dieser Klasse zugänglich.
@property (nonatomic) int someValue = -1;
@end

@implementation MyCoolObject : NSObject

// valueFromPublicInterface ist in der MyCoolObject.h definiert, und daher eine
// öffentlich zugängliche Instanzvariable(sozusagen Teil der Public-API).
@synthesize valueFromPublicInterface = _valueFromPublicInterface;
@synthesize someValue = _someValue;

[...]

@end
```

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

int main(int argc, const char *argv[])
{
    @autoreleasepool { //Bestandteil des ARC. Details folgen.
        Person *erwin = [[Person alloc] init]; //new Person() in Java

        [erwin setGroesse:1.85];
        NSLog(@"Erwin ist %.2f Meter groß", [erwin groesse]);

        [erwin setAlter:23 undGroesse:1.87];
        NSLog(@"Erwin ist %.2f Meter groß", [erwin groesse]);

        erwin = nil; //Zerstört das Objekt
    }
    return 0;
}
```

Methodenaufruf in Java:

- `myObject.hasAWeirdMethod(thatTakesAnArgument,orTwo);`

Methoden sind immer an Code-Block gebunden (Der Teil zwischen den geschweiften Klammern)

Ziel des Methodenaufrufs steht zur compile-Zeit fest.

Objective-C: **Message Passing** anstelle von Calling!

Objective-C: Methodenaufruf ist Nachricht an ein Objekt.

Ziel erst zur Laufzeit bekannt. Das heißt:

- es ist kein type-checking möglich und
- es ist nicht sicher, ob Objekte auf Nachricht reagieren.

Aber: Das macht Objective-C sehr dynamisch!

- Stichwort: Categories (Details folgen)

Receiver Selector Argument

↓ ↓ ↙

```
[person setAlter:neuesAlter  
          undGroesse:neueGroesse ];
```

Instanz-Methoden werden an die Instanz,

- (z.B. `init`)

Klassen-Methoden dagegen an die Klasse gesendet.

- (z.B. `alloc`)

Klassen-Methoden haben keinen Zugriff auf Instanzen oder deren Variablen
(Analog zu statischen Methoden in Java)

Klassen-Methoden werden mit '+' deklariert, Instanzmethoden mit '-'.

- `+ (returnSomething)AndDoSomethingWeird;`

id:

- "Ein Zeiger auf ein beliebiges Objekt"
Return-Typ aller init-Methoden - muss unspezifisch sein wegen Vererbung!
Analog zu (`void *`) in C

isa:

- Jedes Objekt hält seinen eigenen Klassennamen
Entscheiden darüber, welche Klasse für einen Methodenaufruf ausgewählt wird.

self:

- Impliziter Pointer auf das 'eigene' Objekt
- in anderen Sprachen häufig `this`

Alle Objekte erben von `NSObject`

- (Analog zur Oberklasse `Object` in Java)

Keine Mehrfachvererbung möglich!

- (Objekte haben genau eine Oberklasse, die mit `super` angesprochen werden kann) Bsp.:
`self = [super init];`

Methoden überschreiben möglich

- (Methoden können denselben selector haben, aber keine unterschiedlichen return-types.)
- Bsp.:
`-(id)init //überschreiben des Standard-Initialisierers`

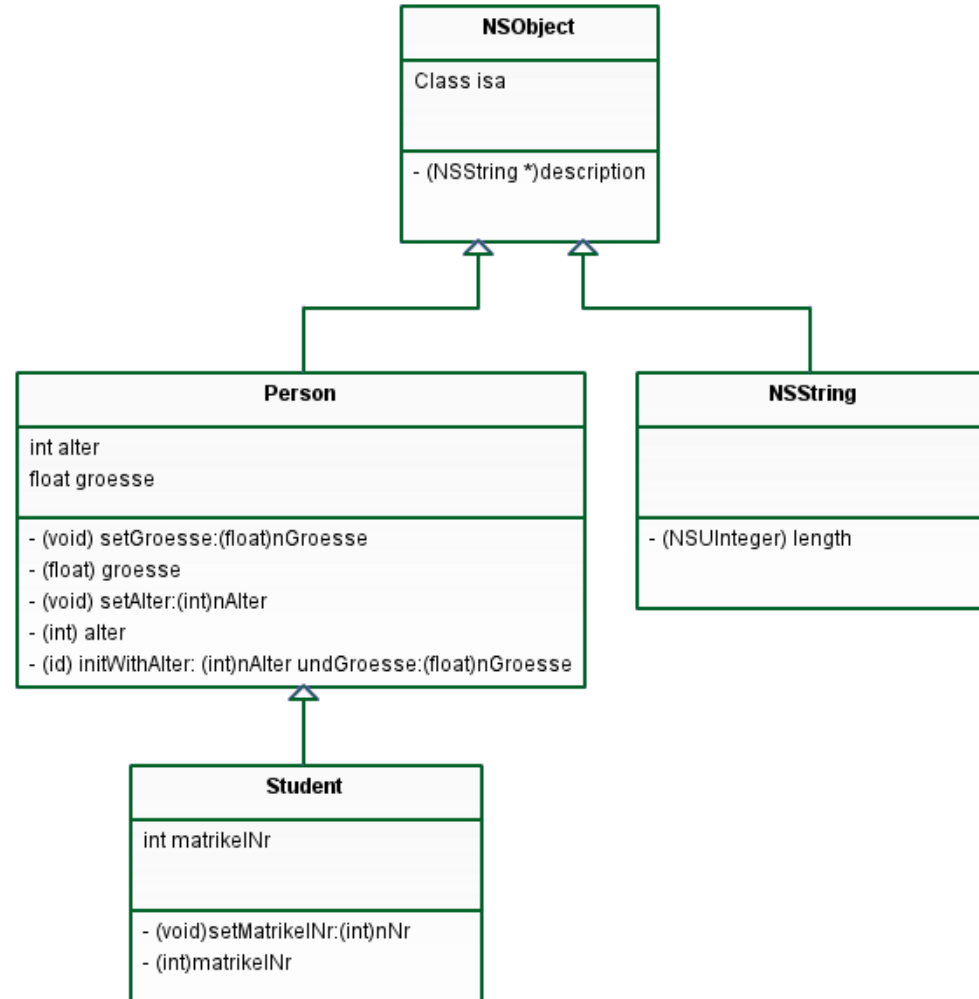
```
#import "Person.h"

@interface Student : Person

@property (nonatomic) int matrikelNr;

- (void)setMatrikelNr:(int)nNr;
//oder @synthesize in Student.m

@end
```



Kinder erben Instanzvariablen und Methoden ihrer Eltern.

Jede Klasse sollte genau einen designierten Initialisierer haben! (implizit oder selbst implementiert)

```
#import "Student.h"

@implementation Student

- (id)init //überschreiben des Standard-Initialisierers
{
    return [self initWithMatrikelNr:1337];
}

- (id)initWithMatrikelNr:(int)nNr //neuer designierter Initialisierer
{
    self = [super init]; //Initialisieren der Oberklasse Person
    if(self){
        //Nur wenn Initialisierung von Person erfolgreich
        [self setMatrikelNr:nNr]; //setzen der Matrikelnummer
    }
    return self;
}

@end
```

Instanzvariablen vom Typ eines beliebigen Objekts

- Bisher wurden nur atomare Datentypen betrachtet...
- Nicht atomar Bsp.:
 - `@property (strong, nonatomic) NSString *name;`
 - `// erzeugt implizit die Instanzvariable NSString* _name`

Objektinstanzvariablen sind immer Zeiger.

- Objekte "befinden" sich nicht in anderen Objekten.

Objekte "besitzen" andere Objekte!



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN



SPEICHERVERWALTUNG

```
#import <stdlib.h>
```

```
int main(int argc, const char *argv[]){
```

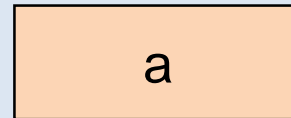
```
    int *a, *b;
```

```
    a = malloc(sizeof(int));
```

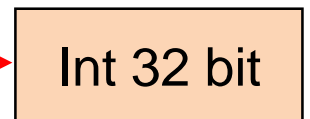
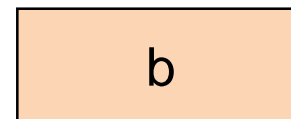
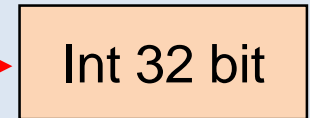
```
    b = malloc(sizeof(int));
```

```
}
```

Pointer (int)



Speicher



```
#import <stdlib.h>
```

```
int main(int argc, const char *argv[]){
```

```
    int *a, *b;
```

```
    a = malloc(sizeof(int));
```

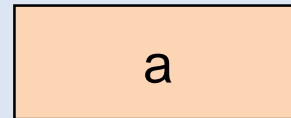
```
    b = malloc(sizeof(int));
```

```
    a = b;
```

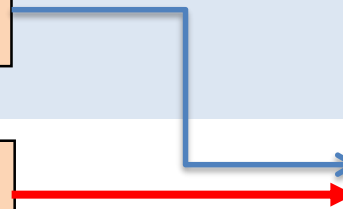
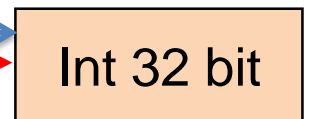
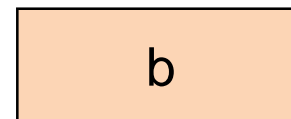
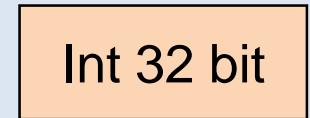
```
    return 0;
```

```
}
```

Pointer (int)



Speicher



Objekte ohne Besitzer sollen zerstört werden.

Objekte mit mindestens einem Besitzer dürfen nicht zerstört werden!

In C:

- Verwaltung von Hand mit `malloc` und `free`

In Objective-C ursprünglich:

- `retain` und `release`

ARC zählt Referenzen auf Objekte.

ARC verwaltet "Besitz" von Objekten.

- (Objekte gehen automatisch in den Besitz dessen über, der die Instanziierung veranlasst.)

ARC löscht Objekte, die keinen Besitzer haben automatisch aus dem Speicher.

- (Ähnlich der Garbage Collection in Java)

Nur Objective-C (in Swift automatisch):

Alle Objekte, die innerhalb eines `@autoreleasepool{}` erzeugt werden, verlieren ihre Besitzer beim Verlassen des Blocks.

Ziel: Einem Objekt seinen Besitzer entziehen.

Die Variable, die auf ein Objekt zeigt...

- auf ein anderes Objekt zeigen lassen
- auf `nil` setzen
- wird selbst gelöscht

Swift bietet für Klassen Deinitialisierer `deinit`, der aufgerufen wird, direkt bevor das Objekt zerstört wird.

```
class Person{
    var name: String
    ...
    deinit{
        println("\(name) wird jetzt gelöscht")
    }
}
```

Problem: Zwei Objekte zeigen gegenseitig aufeinander!

Lösung:

- Beziehung zwischen Objekten als Eltern-Kind-Beziehung begreifen.
- Eltern 'besitzen' ihre Kinder, aber nicht umgekehrt.

Lösung:

- schwache Referenzierung, damit kein Besitz vom referenzierten Objekt ergriffen wird!
Objective-C: `__weak Student *myStudent;`
Swift: `weak var myStudent: Student?`

`myStudent` wird automatisch `nil` wenn das Objekt keinen Besitzer mehr hat!

Über properties können Memory Management Attribute definiert werden

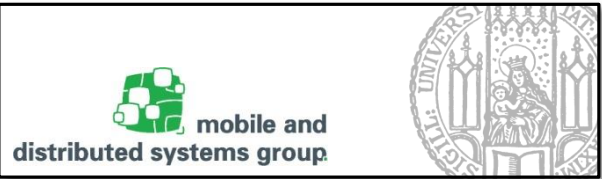
Bsp.:

- `@property (nonatomic, strong) NSString *name;`
- Möglich sind:
 - **Strong** (Default)
 - Starke Referenzierung (Besitz von Objekt)
 - **Weak**
 - Schwache Referenzierung (Keinen Besitz)
 - **Copy**
 - Referenziert auf die Kopie des Objekts
 - Bsp.: Sicherheitsaspekt, falls nicht bekannt, was mit Originalobjekt gemacht wird
 - **Unsafe_unretained**
 - für direct assignments Bsp.: int



LUDWIG-
MAXIMILIANS-
UNIVERSITÄT
MÜNCHEN

Literatur



<https://developer.apple.com/library/ios/navigation/>

<https://www.raywenderlich.com/tutorial-archive>