Programmierung mobiler Geräte

Einführung in Objective-C

Dipl.-Inf. Daniel Wilhelm

Kassel, 09.11.2017

Übersicht

ComTec

- Einführung
- Datentypen
- Kontrollstrukturen
- Funktionen
- Klassen
- Wichtige Klassen (Auszüge)
- Blocks
- Anhang
- Quellen

Einführung in Objective-C

EINFÜHRUNG

Schwierigkeiten?

ComTec

- Fragen in der Vorlesung stellen
- Internetrecherche:
  - Learn2Xcode https://www.youtube.com/user/Learn2Xcode/videos
  - Ray Wenderlich https://www.raywenderlich.com/category/ios
  - Stack Overflow https://stackoverflow.com/
- Bücher lesen (ausleihen bei ComTec, Bibliothek)
- Termin vereinbaren, Kaffee holen, Sessel belegen

Objective-C

ComTec

- Basiert auf ANSI-C (ist eine Obermenge davon)
- Erweiterung der Sprache um Mittel der Objekt-orientierung
- Syntax und Konzepte sind an die Sprache *Smalltalk* angelehnt, z.B. dynamisches Binden

ComTec

Einführung in Objective-C

DATENTYPEN

Übersicht Datentypen

ComTec

- Boolesche Werte
- Ganze Zahlen
- Fließkommazahlen
- Zeiger / Pointer
- Referenzen
- Konstanten
- Aufzählungen

Boolesche Werte

ComTec

- Werden für Vergleiche benötigt
- In ANSI-C kein spezieller Typ vorhanden (`0` ist falsch, alles andere wahr)
- In neuerem C++ wird `bool` (teilw. `int`) mit Werten `true` und `false` eingeführt
- In Objective-C wird meist `BOOL` (= `signed char`) mit Werten `YES` und `NO` verwendet

Ganze Zahlen

ComTec

Objective-C	C / C++	Wertebereich	Größe
NSInteger	int / long	-2^{31} bis $2^{31}-1$	32 bit
	long long	-2^{63} bis $2^{63}-1$	64 bit
NSUInteger	unsigned int / unsigned long	0 bis $2^{32}-1$	32 bit
	unsigned long long	0 bis $2^{64}-1$	64 bit

- Größe der C-Typen teilw. plattformabhängig
- Objective-C-Typen decken größtmöglichen Bereich ab
- Sichere Typen seit C99: `uint8_t`, `int16_t`, `int32_t`, `uint64_t`, ...

Fließkommazahlen (1): Übersicht

ComTec

Objective-C	Typ	Wertebereich	Größe
`CGFloat`	`float`	$1{,}5 \cdot 10^{-45}$ bis $3{,}4 \cdot 10^{38}$	**32 bit**
	`double`	$5{,}0 \cdot 10^{-324}$ bis $1{,}7 \cdot 10^{308}$	**64 bit**

- Wertebereiche der Typen sind plattformabhängig
- In Objective-C sind alle Fließkommazahlen automatisch vom Typ `double`
- Führt zu Problemen bei Vergleichen
- Daher bei `float`s „`f`“ anhängen, z.B.:

```
float fVar = 4.2f;
```

Quelle: [1]

Fließkommazahlen (2): Darstellung

ComTec

- Darstellung mit
  - Vorzeichen (*S*),
  - Mantisse (*M*) und
  - Exponent (*E*):
  - $S \cdot M \cdot 10^E$, z.B.: $42 = +0{,}42 \cdot 10^2$
- Binärdarstellung eines `float`s (little Endian):

Byte 3	Byte 2	Byte 1	Byte 0
7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
SEEEEEEE	EMMMMMMM	MMMMMMMM	MMMMMMMM

Quelle: [2]

Fließkommazahlen (3): Mantisse

ComTec

- Es gilt $1 \leq M < 2$ (normalisiert nach IEEE 754), daher sind nur Nachkommastellen zu speichern
- $M = 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + \ldots + 2^{-n}$
  mit $n = 23$ für `float`
  und $n = 52$ für `double`

- *0,3* z.B. lässt sich somit gerade **nicht** abbilden (periodische Binärdarstellung: $1{,}0011\ldots * 2^{-2}$)
- Durch Binärdarstellung kumulieren sich Fehler bei Rechenoperationen

Quelle: [2]

Fließkommazahlen (4): Exponent

ComTec

- *E* kann zunächst nicht negativ werden, daher Konvention: Mitte (*B*) ist 2^0 (bei `float` *B* = *127*, bei `double` *B* = *1023*)
- Darstellung der *0* so zunächst nicht möglich, daher Konvention: Sind alle Bits *0*, ist die dargestellte Zahl *0*

Quelle: [2]

Fließkommazahlen (5): Vergleiche

ComTec

- Vergleiche sind problematisch:

```
double dVar1 = 69.82;              // dVar1 = 69.819999999999993
double dVar2 = 69.2 + 0.62;        // dVar2 = 69.820000000000007
BOOL bEqual = (dVar1 == dVar2);    // bEqual = NO
```

- Besser Vergleich auf annähernde Gleichheit:

```
double dEpsilon = DBL_EPSILON * 100.0;
double dVar1 = 69.82;
double dVar2 = 69.2 + 0.62;
BOOL bEqual = fabs(dVar1 - dVar2) < dEpsilon;
```

Quelle: [2]

Fließkommazahlen (6): Casting

ComTec

- Mischen von `float` und `double` vergrößert den Fehler

```
double dResult1 = 0.0;
double dResult2 = 0.0;

for (int i = 0; i < 1000; ++i) {
    dResult1 += 0.3;   // addiert double
    dResult2 += 0.3f;  // addiert float
}

// Erwartung: dResult1 = dResult2 = 300.0
// dResult1 = 300.00000000000563
// dResult2 = 300.00001192092896
```

Quelle: [2]

Zeiger / Pointer (1)

ComTec

- Speichert die Adresse einer Variablen
- Deklaration mit ∗-Operator, z.B.:

```
int *p_iVar;
```

- Zugriff auf Speicheradresse einer Variablen mit &-Operator, z.B.:

```
int iVar;
iVar = 10;

int *p_iVar;
p_iVar = &iVar;
```

Zeiger / Pointer (2)

ComTec

- Zugriff auf Wert eines Zeigers mit ∗-Operator, z.B.:

```
int iVar1 = 10, iVar2 = 20;
int *p_iVar = &iVar1;
iVar2 = *p_iVar;              // iVar2 = iVar1 = 10
```

- Zeiger erlauben Zugriff auf dieselben Daten an unterschiedlichen Stellen; mit minimalem Speicherverbrauch

Referenzen

ComTec

- Ähnlich eines Zeiger, aber:
  - Deklaration mit &-Operator
  - Muss nicht dereferenziert werden
  - Zeigt immer auf dieselbe Adresse
  - Verfügbar in C++, nicht in ANSI-C
- Beispiel:

```
int iVar = 10;
int &r_iVar = iVar;
r_iVar += 20;       // iVar = 30
```

Konstanten (1)

ComTec

- Durch Typ-Modifizierer `const` kann Wert einer Variablen nicht durch **direkte Zuweisung** geändert werden

```
// konstanter Wert
int const c_iVar = 1701;

// variabler Zeiger auf konstanten Wert
int const *p_iVar;

// konstanter Zeiger auf variablen Wert
int * const p_iVar = &iVar;

// konstanter Zeiger auf konstanten Wert
int const * const p_iVar = &c_iVar;
```

Konstanten (2)

ComTec

- Symbolische Konstanten mittels `#define`
  - Werden vom Präprozessor durch Literale ersetzt
  - Auch Funktionsdefinitionen möglich

```
#define WORLD "Hello World!"

#define screenWidth 325.0

#define ConsoleLog(s, ...) NSLog(@"<%@:%d> %@", [[NSString
stringWithUTF8String:__FILE__] lastPathComponent], LINE,
[NSString stringWithFormat:(s), ##__VA_ARGS__])
```

Aufzählungen (1): Definition

ComTec

- Definition in ANSI-C:

```
enum Name {Liste der Elemente};    // neue Aufzählung
enum Name variable = Element;      // Variable erstellen
```

  - Der Name ist optional
  - Datentyp der Elemente ist (normalerweise) `int`
  - Erstes Element erhält Wert `0`, alle weiteren werden durchgezählt (+`1`)
  - Durch ein = können der Startwert angepasst oder explizite Werte vergeben werden

Aufzählungen (2): Beispiele

ComTec

```
// (1) kein Name vergeben
enum {FALSCH, RICHTIG};    // FALSCH = 0, RICHTIG = 1
int wahrheit = FALSCH;

// (2) Name vergeben
enum direction {NORTH, EAST = 90, SOUTH = 180, WEST = 270};
enum direction my_direction = WEST;

// (3) Kurzschreibweise
enum weekday {MON, TUE, WED, THU, FRI, SAT, SUN} my_weekday;
my_weekday = 0;

// (4) in Verbindung mit Typdefinition (Kurzschreibweise)
typedef enum pet {CAT, DOG, RAT, BIRD} petType;
petType my_pet = RAT;
```

Aufzählungen (3)

ComTec

- Definition in Objective-C (Makro seit iOS 6):

```
typedef NS_ENUM(Datentyp, Name) {Liste der Elemente};
```

- Werte der Elemente sind analog zu `enum`
- Übersichtlicher, Typprüfungen sind möglich

```
// (5) Makro
typedef NS_ENUM(uint8_t, weekday) {
     MON, TUE, WED, THU, FRI, SAT, SUN
};
weekday my_weekday = FRI;
```

ComTec

Einführung in Objective-C

KONTROLLSTRUKTUREN

Übersicht Kontrollstrukturen

ComTec

- Bedingte Ausführung
- `while`-Schleife
- `do-while`-Schleife
- `for`-Schleife

Bedingte Ausführung (1)

ComTec

```
if (Bedingung) {
   // Bedingung wahr
}

else {
   // Bedingung nicht wahr
}
```

- Auswahloperator / Bedingter Ausdruck:

  `(Bedingung) ? Anweisung : Alternative`

  Bsp.: `double dVar = (iVar < 5) ? 10.0 : 20.0;`

Bedingte Ausführung (2)

ComTec

```
switch (Ausdruck) {
   case 1:
      // Anweisungen
      break;

   case 2:
   case 3:
      // Anweisungen
      break;

   default:
      // Anweisungen
      break;
}
```

`while`-Schleife

ComTec

- Zunächst wird die Bedingung geprüft, dann die Schleife ggf. ausgeführt
- Vorgang wird wiederholt so lange *Bedingung* wahr ist

```
while (Bedingung) {
   // Anweisungen
}
```

do-while-Schleife

ComTec

- Schleife wird mind. einmal ausgeführt
- Dann wird erst die Bedingung geprüft und die Schleife ggf. noch einmal ausgeführt
- Vorgang wird wiederholt so lange *Bedingung* wahr ist

```
do {
   // Anweisungen
} while (Bedingung);
```

`for`-Schleife (1): Syntax

ComTec

- Standard:

```
for (Initialisierung; Bedingung; Fortsetzung) {
   // Anweisungen
}
```

- Fast Enumeration:

```
for (Typ *Name in Kollektion) {
   // Anweisungen
}
```

for-Schleife (2): Beispiele

ComTec

- Standard:

```
for (NSUInteger i = 0; i < myArray.count; i++) {
   NSString *myString = [myArray objectAtIndex:i];
   // Anweisungen
}
```

- Fast Enumeration:

```
for (NSString *myString in myArray) {
   // Anweisungen
}
```

ComTec

Einführung in Objective-C

FUNKTIONEN

Übersicht Funktionen

ComTec

- Spezielle Datentypen
- Definition von Funktionen
- Funktionsaufrufe
- Variadische Funktionen

Spezielle Datentypen

ComTec

- `void` - Funktion hat keinen Rückgabewert
- `id` ist ein Zeiger auf einen beliebigen Objekttyp (dieser wird erst zur Laufzeit festgelegt)
- `nil` ist ein Objekt vom Typ `id` und repräsentiert den Null-Wert; Standardinitialisierung von Objektzeigern; Achtung: `nil` != `NULL`

Definition von Funktionen (1)

ComTec

- **Instanzmethoden** müssen auf der Instanz einer Klasse aufgerufen werden, z.B.:

```
- (void)doSomething;
```

- **Klassenmethoden** werden direkt auf einer Klasse aufgerufen, z.B.:

```
+ (void)doAnything;
```

Definition von Funktionen (2)

ComTec

	Methode ohne Argumente
Beispiel	`- (int)length;`
Funktionsname	`length`
Erläuterung	`- (`*`Typ`*`)`*`Name`*`;`

	Methode mit einem Argument
Beispiel	`- (void)setLength:(int)newLength;`
Funktionsname	`setLength:`
Erläuterung	`- (`*`Typ`*`)`*`Name`*`:(`*`Typ`*`)`*`Argument`*`;`

	Methode mit zwei Argumenten
Beispiel	`- (void)setWidth:(int)newWidth` `  andHeight:(int)newHeight;`
Funktionsname	`setWidth:andHeight:`
Erläuterung	`- (`*`Typ`*`)`*`Name1`*`:(`*`Typ`*`)`*`Arg1 Name2`*`:(`*`Typ`*`)`*`Arg2`*`;`

Funktionsaufrufe

ComTec

- Aufruf einer Methode entspricht dem Versand einer Nachricht an ein Objekt

```
[Empfänger Nachricht];
z.B.: int length = [line length];
z.B.: [rectangle setWidth:10 andHeight:10];
```

- Sonderfall: Kurzschreibweise bei Getter und Setter (unter Einhaltung gewisser Konventionen)

```
z.B.: int length = line.length;
z.B.: line.length = 10;
```

Variadische Funktionen (1)

ComTec

- Funktionen mit unbestimmter Arität (Parameter-zahl wird nicht bei Deklaration festgelegt)

	Methode mit beliebig vielen Argumenten – Format Strings
Beispiel	`- (id)initWithFormat:(NSString *)format, ...;`
Funktionsname	`initWithFormat:`
Erläuterung	`- (Typ)Name:(Typ)Arg1, ...;`

- Erster Parameter fest, der Rest variabel (aber nicht zwingend optional)
- Anzahl und Typ weiterer Parameter werden durch bzw. im Format String festgelegt

Variadische Funktionen (2)

ComTec

	Methode mit beliebig vielen Argumenten – Nil Terminated Lists
Beispiel	`- (id)initWithObjects:(id)firstObj, ... NS_REQUIRES_NIL_TERMINATION;`
Funktionsname	`initWithObjects:`
Erläuterung	`- (Typ)Name:(Typ)Arg1, ... NS_REQUIRES_NIL_TERMINATION;`

- Letztes Argument **muss** `nil` sein
- Parameter sind **immer** Pointer
- Anzahl der Parameter = Anzahl vor erstem `nil`

ComTec

Einführung in Objective-C

KLASSEN

Übersicht Klassen

ComTec

- OOP-Denkweise
- Interface-Definitionen
- Implementierung & Kategorien
- Propertys
- Objektinstanzen
- Referenzzähler

OOP-Denkweise

ComTec

- Trennung der Implementierung von der Definition; entspricht der objektorientierten Denkweise („Abstraktion“)
- Nur eine Klasse pro Datei üblich

Dateiendung	Beschreibung
h	Header-Datei Enthält öffentliche Schnittstellen (Definition von Klassen, Funktionen)
m	Implementierungsdatei Enthalten sowohl Objective-C- als auch reinen C-Code
mm	Implementierungsdatei Enthalten sowohl Objective-C-, C++- als auch reinen C-Code

Interface-Definition (1)

ComTec

```
@interface Klassenname : Oberklasse <Protokolle> {
   // Definition von Instanzvariablen
   @protected
}
// Definition von Methoden und Propertys
@end
```

Direktive	Auswirkung
@private	Zugriff nur innerhalb der Klasse, in der sie definiert wurde, möglich
@protected	Standard; Zugriff innerhalb der Klasse, in der sie definiert wurde und allen Unterklassen möglich
@public	Zugriff von überall aus möglich
@package	Zugriff nur innerhalb des Implementierungspaketes möglich

Interface-Definition (2): Protokolle

ComTec

- Mechanismus zur Definition eines Interfaces
- Enthält eine Liste von Methodendefinitionen (Variablen können nicht definiert werden)
- Zur Gruppierung von Methoden für bestimmte Aufgaben
- Syntax:

```
@protocol Protokollname
// Definition der Methoden
@end
```

Interface-Definition (3): Protokolle

ComTec

Direktive	Beschreibung
`@required`	Standard; Methode muss implementiert werden
`@optional`	Methode kann implementiert werden

- Eine Klasse erfüllt ein Protokoll, wenn sie alle dort definierten Methoden implementiert, die nicht optional sind

Implementierung & Kategorien

ComTec

```
#import "Klassenname+Kategoriename.h"

@implementation Klassenname (Kategoriename)
// Implementierung der Methoden
@end
```

- Kategorien erlauben das Hinzufügen von Funktionen zu einer Klasse ohne Ableitung
- Kategorien können auch auf private Variablen zugreifen und stehen Unterklassen zur Verfügung

Propertys (1): Definition

ComTec

```
@interface Line : NSObject {
   int length;
}
@property int length;
@end
```

```
@implementation Line
@synthesize length;
@end
```

```
- (int)length {}
- (void)setLength:(int)length {}
```

- Definition:

  `@property (Attribute) Typ Name;`

Propertys (2): Attribute

ComTec

Attribut	Beschreibung
`getter`=*Name*	Festlegung des Namens der Getter-Methode Standard: Name der Property; z.B. `length`
`setter`=*Name*	Festlegung des Namens der Setter-Methode Standard: `set` + Name der Property mit großem Anfangsbuchstaben; z.B. `setLength`
`readwrite`	Property kann gelesen und geschrieben werden Dieser Wert ist das Standardverhalten
`readonly`	Property kann nur gelesen werden
`assign` / `weak`	Reine Zuweisung Dieser Wert ist das Standardverhalten
`retain` / `strong`	Alter Wert erhält `release`-, der neue `retain`-Nachricht; funktioniert nur bei Objective-C-Datentypen
`copy`	Durch Kopie wird eine neue Instanz erzeugt, der alten eine `release`-Nachricht gesendet; `NSCopying`-Protokoll beachten

Objektinstanzen erzeugen

ComTec

```
// Speicher reservieren
Line *myLine = [Line alloc];
// Objekt initialisieren
myLine = [myLine init];


// Alternative Kurzschreibweise:
// Speicher reservieren und Objekt initialisieren
Line *myLine = [[Line alloc] init];



// Speicher freigeben
[myLine release];
```

Referenzzähler (1): Prinzip

ComTec

- Jedes Objekt hat einen Zähler, der die Anzahl auf ihn zeigender Zeiger zählt
  - Objekte, deren Referenzzähler *>0* ist, bleiben im Speicher
  - Fällt der Zähler auf *0*, wird das Objekt gelöscht
- Die Methode(n) ...
  - ... `alloc`, `copy` und `retain` (und einige weitere) erhöhen den Referenzzähler um *1*
  - ... `release` dekrementiert den Zähler um *1*

Referenzzähler (2): ARC

ComTec

- Automatic Reference Counting (ARC) eingeführt mit iOS 5
- Compiler fügt Zählfunktionen selbstständig hinzu
- Neue Direktiven: `strong` und `weak`

- Programmierer hat weniger Arbeit
- Weniger fehleranfällig
- Funktionsaufrufe wie `retain`, `release` etc. sind verboten

Einführung in Objective-C

WICHTIGE KLASSEN

Übersicht wichtige Klassen

ComTec

- `NSNumber`
- `NSString`
- Kollektionen:
  - `NSArray`
  - `NSSet`
  - `NSDictionary`
- `NSPredicate`

NSNumber

ComTec

- Dient zum Speichern von (unveränderlichen) Zahlen (`BOOL`, `int`, `float`, ...)
- Umwandlungsfunktionen für alle unterstützten Typen vorhanden, z.B.:

```
NSNumber *myNumber = [NSNumber numberWithDouble:0.666];
float fVar = [myNumber floatValue];
```

- Abkürzung zur Number-Erstellung: `@`, z.B.:

```
NSNumber *myNumber = @42;
myNumber = @YES;
```

NSString (1)

ComTec

- Klasse zum Umgang mit (unveränderlichen) Zeichenketten
- Viele hilfreiche Funktionen wie z.B.: Suchen und Ersetzen, Unterstützung von Kodierungen etc.
- Abkürzung zur String-Erstellung: @"", z.B.:

```
NSString *myString = @"Hello World!";
```

NSString (2)

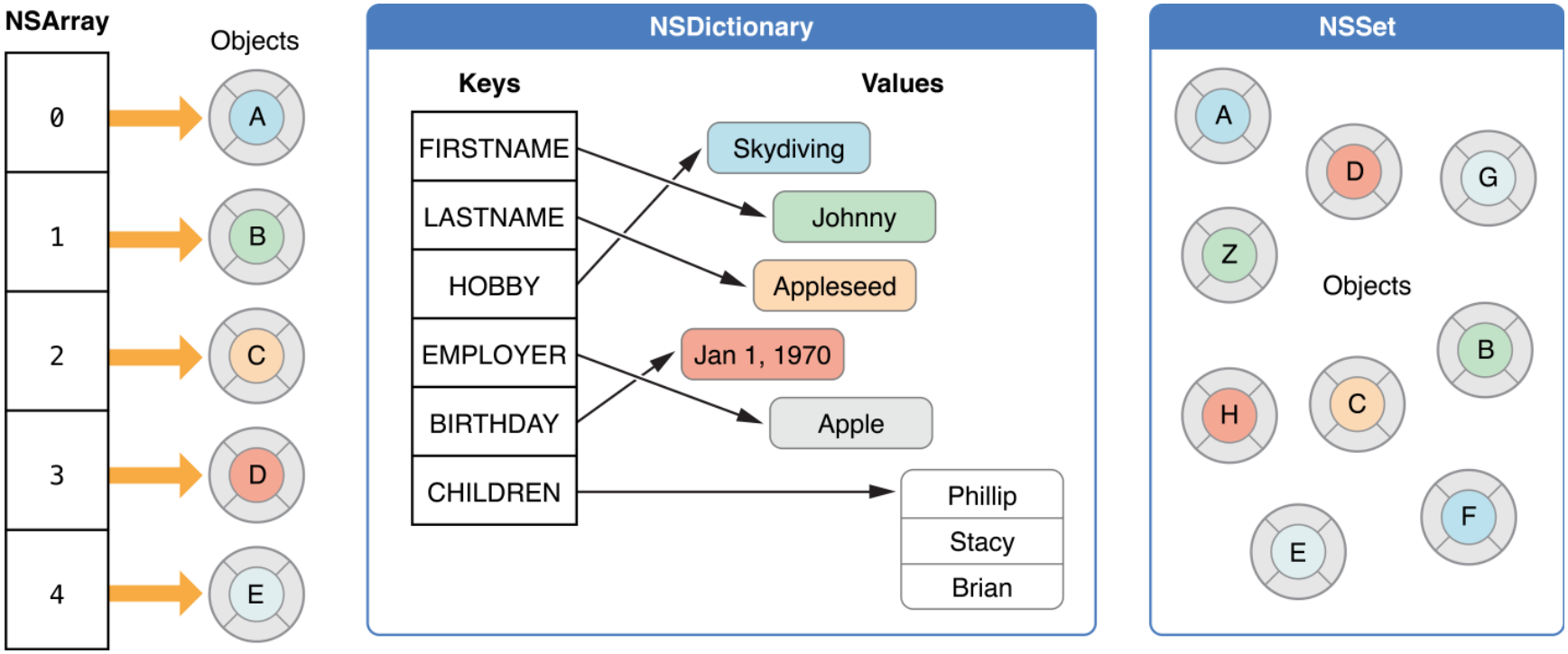
ComTec

- Zum Ändern einer Zeichenkette muss NSMutableString verwendet werden, z.B.:

```
NSMutableString *myString = [[NSMutableString alloc]
                             initWithString:@"HeXlo "];

[myString appendString:@"World!"];

[myString replaceOccurrencesOfString:@"X"
          withString:@"l"
          options:NSCaseInsensitiveSearch
          range:NSMakeRange(0, myString.length)];



// myString = @"Hello World!"
```

Kollektionen

ComTec



Quelle: [3]

Kollektionen: NSArray (1)

ComTec

- (Unveränderliche) Klasse zum Speichern von Objekten beliebiger Art, z.B.:

```
NSArray *myArray = [NSArray arrayWithObjects:@"Text", @42,
                    nil];
```

- Gespeicherte Objekte werden über einen Index (beginnend bei 0) abgerufen, z.B.:

```
id myObject = [myArray objectAtIndex:0];
```

- Abkürzung zur Array-Erstellung: `@[]`, z.B.:

```
NSArray *myArray = @[@"Text", @42];
```

Kollektionen: NSArray (2)

ComTec

- Zum Hinzufügen oder Entfernen von Objekten muss NSMutableArray verwendet werden

```
NSMutableArray *myArray = [[NSMutableArray alloc]
                           initWithArray:@[@"World!",
                           @"Folks!"]];

[myArray removeObjectAtIndex:1];

[myArray addObject:@"Hello"];

[myArray sortedArrayUsingSelector:
         @selector(caseInsensitiveCompare:)];


// myArray = @[@"Hello", @"World!"]
```

Kollektionen: NSSet

ComTec

- Im Unterschied zu Arrays:
  - Unsortiert
  - Erlaubt keine Duplikate
  - Sehr schnell bei Suchen
  - Prüfung auf Gleichheit, Unter- und Schnittmengen
  - Zugriff normalerweise über Vergleiche („Ist folgendes Objekt in der Kollektion?“)

Kollektionen: NSDictionary (1)

ComTec

- (Unveränderliche) Klasse zum Speichern von Objekten beliebiger Art, z.B.:

```
NSDictionary *myDict = [NSDictionary
                        dictionaryWithObjectsAndKeys:@"value1",
                        @"key1", @"value2", @"key2", nil];
```

- Gespeicherte Objekte werden über einen Schlüssel abgerufen, z.B.:

```
id myObject = [myDict objectForKey:@"key1"];
```

Kollektionen: NSDictionary (2)

ComTec

- Abkürzung zur Dictionary-Erstellung: `@{}`, z.B.:

```
NSDictionary *myDict = @{@"key" : @"value"};
```

- Zum Hinzufügen oder Entfernen von Objekten muss `NSMutableDictionary` verwendet werden, z.B.:

```
NSMutableDictionary *myDict = [[NSMutableDictionary alloc]
                               init];
[myDict setObject:@42 forKey:@"key"];
[myDict removeObjectForKey:@"key"];
```

NSPredicate

ComTec

- Enthält logische Bedingungen zum Durchführen von Suchen und Filtern, z.B.:

```
NSArray *data = @[@{@"name" : @"Adam", @"age" : @22},
                  @{@"name" : @"Eve",  @"age" : @21}];
NSPredicate *predicate = [NSPredicate predicateWithFormat:
                          @"name like 'Eve'"];
[data filteredArrayUsingPredicate:predicate];
// Ergebnis: @[@{@"name" : @"Eve", @"age" : @21}]

[NSPredicate predicateWithFormat:@"name beginswith 'A'"];
[NSPredicate predicateWithFormat:@"age > 21"];
// Ergebnis: @[@{@"name" : @"Adam", @"age" : @22}]
```

ComTec

Einführung in Objective-C

BLOCKS

Übersicht Blocks

ComTec

- Einführung
- Blocks mit Parameter
- Blocks ohne Parameter
- Gültigkeitsbereich
- Blocks als Parameter
- Vereinfachung

Einführung Blocks

ComTec

- Ähnlich („anonymer“) Funktion ohne Name
- Verfügbar in C, C++ und Objective-C
- Werden als Objekte betrachtet und können somit
  - Funktionen als Parameter übergeben,
  - in Kollektionen gespeichert und
  - als Propertys verwendet werden
- Können auf Variablen im Gültigkeitsbereich (Scope) zugreifen

Beispiel ohne Parameter

ComTec

- Beispiel eines Blocks, gespeichert in Variable; ohne Parameter und ohne Rückgabewert:

```
void (^blockName)(void) = ^{
   NSLog(@"Ich stehe in einem Block.");
};
```

- Aufruf des Blocks:

```
blockName();
```

Beispiel mit Parametern

ComTec

- Beispiel eines Blocks, gespeichert in Variable; mit Parameter und Rückgabewert:

```
int (^multiply)(int, int) = ^ int (int first, int second) {
   return first * second;
};
```

- Aufruf des Blocks:

```
int result = multiply(3, 14);      // result = 42
```

Gültigkeitsbereich (1)

ComTec

- Werte im Scope werden bei der Deklaration kopiert (können auch nicht verändert werden):

```
int iVar = 42;

void (^output)(void) = ^{
   NSLog(@"iVar hat den Wert %d", iVar);
};

iVar = 3;

output();   // iVar hat den Wert 42
```

Gültigkeitsbereich (2)

ComTec

- Zum Ändern einer Variable muss deren Speicher mit dem Block geteilt werden:

```
__block int iVar = 42;

void (^output)(void) = ^{
   NSLog(@"iVar hat den Wert %d", iVar);
};

iVar = 3;

output();   // iVar hat den Wert 3
```

Blocks als Parameter (1)

ComTec

- Code wird an anderer Stelle ausgeführt, z.B.
  - als Callback-Code oder
  - für Wiederholungen bei Stapelverarbeitungen
- Definition (Bsp. ohne Parameter):

```
- (void)methodWithBlock:(void (^)(void))block;
```

- Implementierung (Bsp. mit Parametern):

```
- (void)methodWithBlock:(void (^)(double, int))block {
    block(0.666, 42);
}
```

Blocks als Parameter (2)

ComTec

- Zur Erhaltung der Lesbarkeit:
  - Nur ein Block pro Methode
  - Definition des Blocks immer am Ende

```
[self beginTaskWithName:@"Mein Task" completion:^{
   NSLog(@"Ich bin fertig.");
}];
```

Vereinfachung

ComTec

- Zur Vereinfachung können Blocks per `typedef` vordefiniert werden, z.B.

```
typedef void (^voidBlock)(void);

voidBlock block = ^{
   // ...
};

- (void)doSomething:(voidBlock)block {
   // ...
   block();
}
```

ComTec

Einführung in Objective-C

ANHANG

Übersicht Anhang

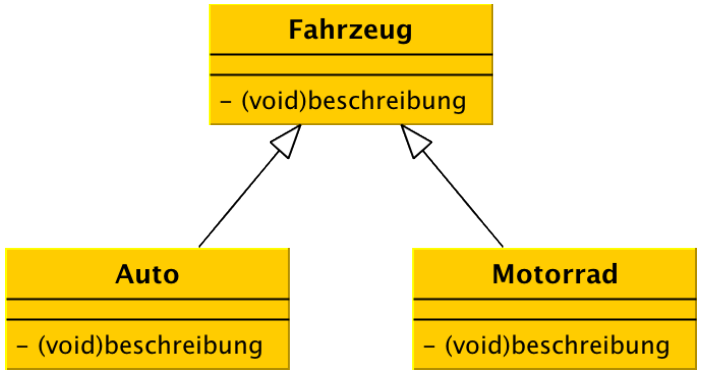
ComTec

- Beispiel dynamisches Binden
- Rechenbeispiel Gleitkommazahl (`float`)
- Beispielumsetzung einer variadischen Funktion
- Weiterführende Links

Beispiel dynamisches Binden (1)

ComTec

```
@implementation Fahrzeug : NSObject

- (void)beschreibung {
    NSLog(@"Ich bin ein Fahrzeug.");
}

@end

@implementation Auto : Fahrzeug

- (void)beschreibung {
    NSLog(@"Ich bin ein Auto.");
}

@end

@implementation Motorrad : Fahrzeug

- (void)beschreibung {
    NSLog(@"Ich bin ein Motorrad.");
}

@end
```



Beispiel dynamisches Binden (2)

ComTec

```
@implementation ViewController

- (void)viewDidLoad {
    NSArray *fuhrpark = [NSArray arrayWithObjects:[[Fahrzeug alloc] init],
                                                  [[Auto alloc] init],
                                                  [[Motorrad alloc] init],
                                                  nil];
    for (Fahrzeug *fahrzeug in fuhrpark) [fahrzeug beschreibung];
}

@end
```

```
// Ausgabe:
// Ich bin ein Fahrzeug.
// Ich bin ein Auto.
// Ich bin ein Motorrad.
```

Rechenbeispiel Gleitkommazahl

ComTec

Aufgabe: *18,4* umrechnen in Binärdarstellung (32 bit) nach IEEE 754

1. Vorzeichen-Bit bestimmen:
   $(-1)^S \rightarrow S = 0$

2. Umrechnung der Mantisse:
   $M = 18{,}4_{10} = 10010{,}011001100110011..._2$

3. Normierung der Mantisse:
   $M = 10010{,}01100110011... \cdot 2^0 = 1{,}001001100110011... \cdot 2^4$

4. Berechnung des Biaswertes (n: Größe des Exponenten):
   $B = 2^{(n-1)} - 1 = 2^{(8-1)} - 1 = 2^7 - 1 = 128 - 1 = 127$

5. Berechnung des Exponenten:
   $\text{Exponent + Bias} = 4_{10} + 127_{10} = 131_{10} = 10000011_2$

6. Gleitkommazahl bilden:
   0 10000011 00100110011001100110011

Quelle: [4]

Umsetzung variadische Funktion

ComTec

```
- (void)appendStrings:(NSString *)firstArg, ... {
   va_list args = NULL;       // Pointer auf variable Liste
   va_start(args, firstArg); // Initialisierung, zeigt auf 2.
                                Element
   for (NSString *arg = firstArg;          // Start bei 1. Element
        arg != nil;                        // Abbruch bei nil
        arg = va_arg(args, NSString *)) { // Holt nächstes Element
           [contentString appendString:arg];
        }

   va_end(args);  // Speicher freigeben
}
```

Quelle: [5]

Weiterführende Links

ComTec

- Matt Thompson: „BOOL / bool / Boolean / NSCFBoolean“ (http://nshipster.com/bool/, 2013)
- Stack Overflow: „iphone - Objective-C ARC: strong vs retain and weak vs assign“ (http://stackoverflow.com/a/15541801, 2013)
- Matt Thompson: „NSPredicate“ (http://nshipster.com/nspredicate/, 2013)

Stand: 7. November 2014

Einführung in Objective-C

QUELLEN

Quellen (1)

ComTec

- Stäuble, Markus: „Programmieren fürs iPhone“ (dpunkt.verlag GmbH, 1. Auflage 2009)
- Sadun, Erica: „Das große iPhone Entwicklerbuch“ (Addison-Wesley Verlag, 2010)
- Dr. Koller, Dirk: „iPhone-Apps entwickeln“ (Franzis Verlag GmbH, 2010)
- Apple Inc.: „iOS Developer Library“ (https://developer.apple.com/library/ios/navigation/)

Quellen (2)

ComTec

- [1] Dr. Böhme, P.: „C/C++ - Datentyp float “ (http://www2.informatik.uni-halle.de/lehre/c/c623.html, 1995)
- [2] Smode, Dieter: „Fließkommadarstellung und Problembehandlung“ (http://www.mpdvc.de/artikel/FloatingPoint.htm, 2004)
- [3] Apple Inc.: „About Collections“ (https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/Collections/Collections.html, 2010)
- [4] Wikipedia: „Gleitkommazahl“ (http://de.wikipedia.org/wiki/Gleitkommazahl, 2013)
- [5] Gallagher, Matt: „Variable argument lists in Cocoa“ (http://www.cocoawithlove.com/2009/05/variable-argument-lists-in-cocoa.html, 2009)

Stand: 23. November 2016