

Communication Technologies 2

Einführung in Objective-C

Dipl.-Inf. Daniel Wilhelm
Kassel, 17.02.2014



Übersicht

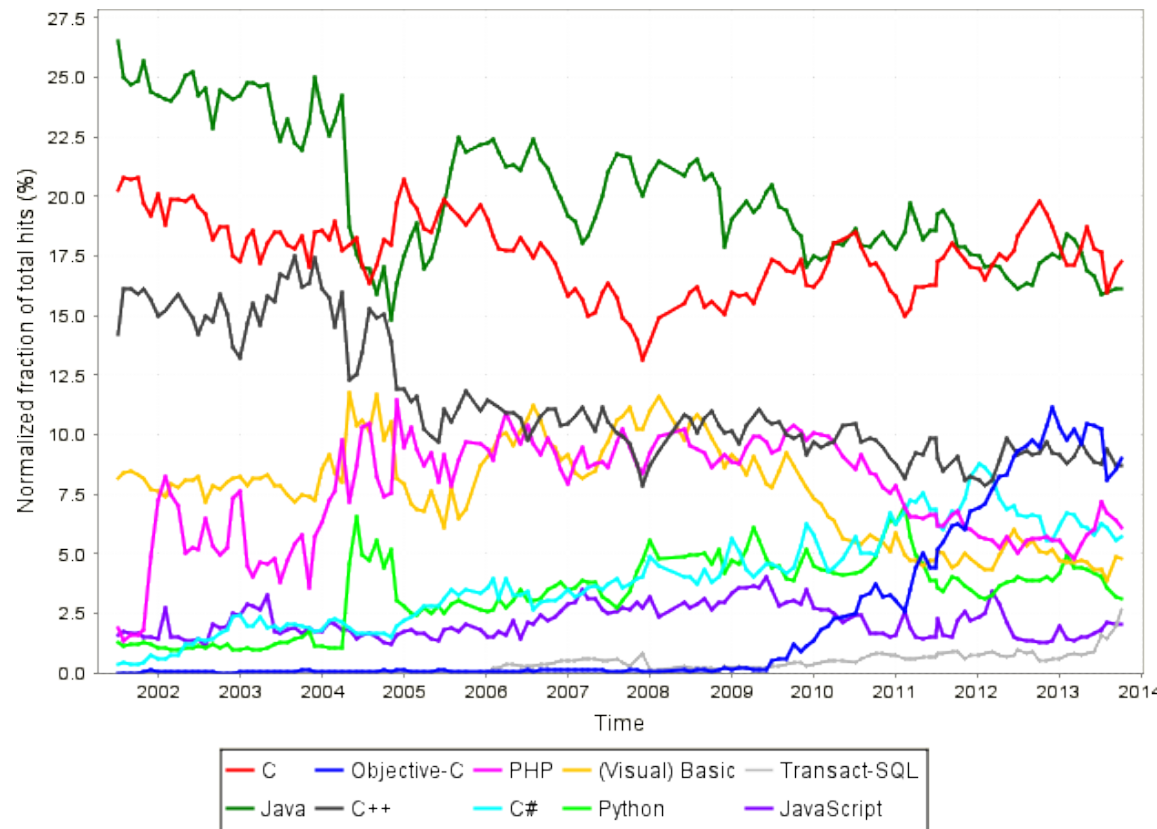
- Motivation & Einführung
- Datentypen
- Kontrollstrukturen
- Funktionen
- Klassen
- Wichtige Klassen (Auszüge)
- Quellen

Einführung in Objective-C

MOTIVATION & EINFÜHRUNG

Popularität von Objective-C (1)

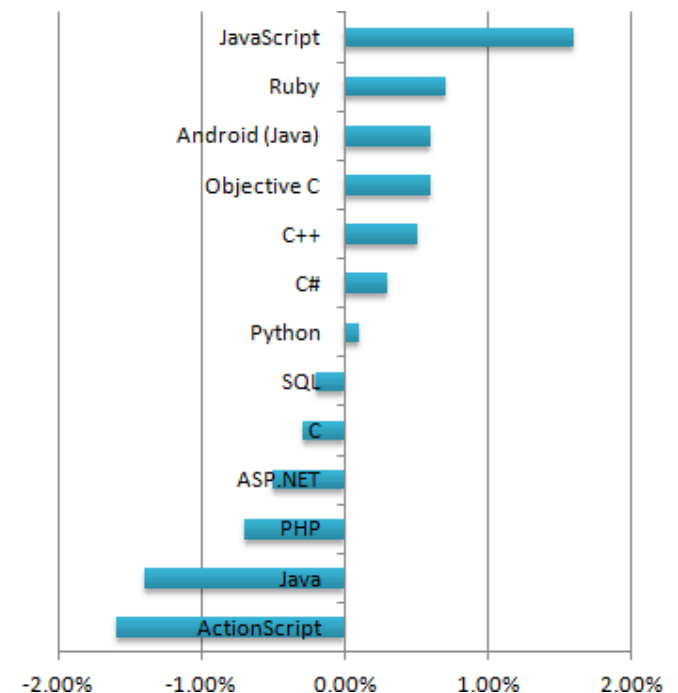
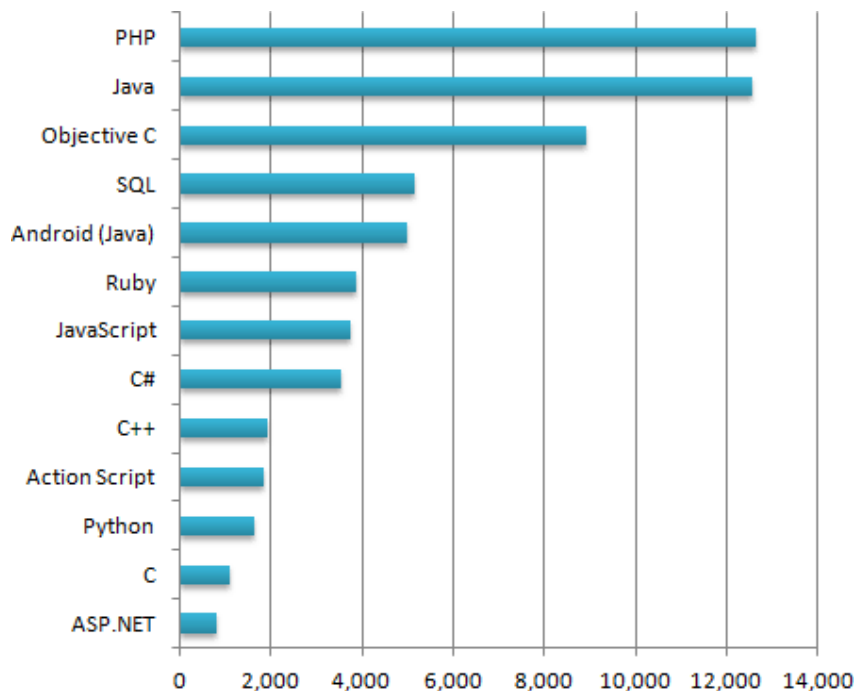
- TIOBE Programming Community Index



Quelle: [1]

Popularität von Objective-C (2)

- Jobs Tractor language trends 2012



Quelle: [2]

- 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

Einführung in Objective-C

DATENTYPEN

Übersicht Datentypen

- Boolesche Datentypen
- Ganze Zahlen
- Fließkommazahlen
- Pointer

Datentypen: Boolesch

- 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 `B00L` (= `signed char`) mit Werten `YES` und `NO` verwendet

Datentypen: Ganze Zahlen

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

- 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`, ...

Datentypen: Fließkommazahlen (1)

Typ	Wertebereich	Größe
<code>float</code>	$1,2 \cdot 10^{-38}$ bis $3,4 \cdot 10^{38}$	4 Byte
<code>double</code>	$2,3 \cdot 10^{-308}$ bis $1,7 \cdot 10^{308}$	8 Byte

- In Objective-C sind alle Fließkommazahlen automatisch vom Typ `double`
- Führt zu Problemen bei Vergleichen
- Daher bei `floats` „f“ anhängen, z.B.:

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

Datentypen: Fließkommazahlen (2)

- 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: [3]

Datentypen: Fließkommazahlen (3)

- 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 = 300.000000000000563
// dResult2 = 300.00001192092896
```

Quelle: [3]

Datentypen: Pointer (1)

- 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;
```

Datentypen: Pointer (2)

- Zugriff auf Wert eines Pointers mit *-Operator, z.B.:

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

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

Einführung in Objective-C

KONTROLLSTRUKTUREN

Übersicht Kontrollstrukturen

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

Bedingte Ausführung (1)

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

- Auswahloperator / Bedingter Ausdruck:

(Bedingung) ? Anweisung : Alternative

Bsp.: `double dVar = (bVar == NO) ? 10.0 : 20.0;`

Bedingte Ausführung (2)

```
switch (Ausdruck) {  
    case 1:  
        // Anweisungen  
        break;  
  
    case 2:  
    case 3:  
        // Anweisungen  
        break;  
  
    default:  
        // Anweisungen  
        break;  
}
```

while-Schleife

- Zunächst wird die Bedingung geprüft, dann die Schleife ggf. ausgeführt

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

do-while-Schleife

- Schleife wird mind. einmal ausgeführt
- Dann wird erst die Bedingung geprüft und die Schleife ggf. noch einmal ausgeführt

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

for-Schleife (1): Syntax

- Standard:

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

- Fast Enumeration:

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

for-Schleife (2): Beispiele

- Standard:

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

- Fast Enumeration:

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

Einführung in Objective-C

FUNKTIONEN

Übersicht Funktionen

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

Spezielle Datentypen

- `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` \neq `NULL`

Definition von Funktionen (1)

- **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)

Methode ohne Argumente

Beispiel	– (int)length;
Funktionsname	length
Erläuterung	– (<i>Typ</i>)Name;

Methode mit einem Argument

Beispiel	– (void)setLength:(int)newLength;
Funktionsname	setLength:
Erläuterung	– (<i>Typ</i>)Name:(<i>Typ</i>)Argument;

Methode mit zwei Argumenten

Beispiel	– (void)setWidth:(int)newWidth andHeight:(int)newHeight;
Funktionsname	setWidth:andHeight:
Erläuterung	– (<i>Typ</i>)Name1:(<i>Typ</i>)Arg1 Name2:(<i>Typ</i>)Arg2;

Funktionsaufrufe

- 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)

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

Methode mit beliebig vielen Argumenten – Format Strings	
Beispiel	– (<i>id</i>)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)

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

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

Einführung in Objective-C

KLASSEN

Übersicht Klassen

- OOP-Denkweise
- Interface-Definition
 - Protokolle
- Implementierung & Kategorien
- Propertys
- Objektinstanzen erzeugen
- Referenzzähler

- 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)

```
@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

- 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

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

```
#import "Klassenname+Kategorienname.h"  
  
@implementation Klassenname (Kategorienname)  
// 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)

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

```
@implementation Line  
@synthesize length; {  
    - (int)length {}  
    - (void)setLength:(int)length {}  
@end
```

- Definition:

```
@property (Attribute) Typ Name;
```

Property (2)

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

Objektinstanzen erzeugen

```
// 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

- 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

Einführung in Objective-C

WICHTIGE KLASSEN

Übersicht wichtige Klassen

- NSNumber
- NSString
- NSArray
- NSDictionary

NSNumber

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

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

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

```
NSNumber *myNumber = @10;  
myNumber = @YES;
```

NSString (1)

- 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)

- 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 release];
```

NSArray (1)

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

```
NSArray *myArray = [NSArray arrayWithObjects:@"Text 1",  
                                                         @"Text 2", 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 1", @"Text 2"];
```


NSArray (2)

- 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 release];
```

NSDictionary (1)

- (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"];
```

NSDictionary (2)

- 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:@"value" forKey:@"key"];  
[myDict removeObjectForKey:@"key"];  
[myDict release];
```

Einführung in Objective-C

ANHANG

Übersicht Anhang

- Beispiel dynamisches Binden
- Darstellung von Fließkommazahlen
- Rechenbeispiel Fließkommazahl (**float**)
- Umsetzung einer variadischen Funktion
- Weiterführende Links

Beispiel dynamisches Binden (1)

```
@implementation Fahrzeug : NSObject

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

@end

@implementation Auto : Fahrzeug

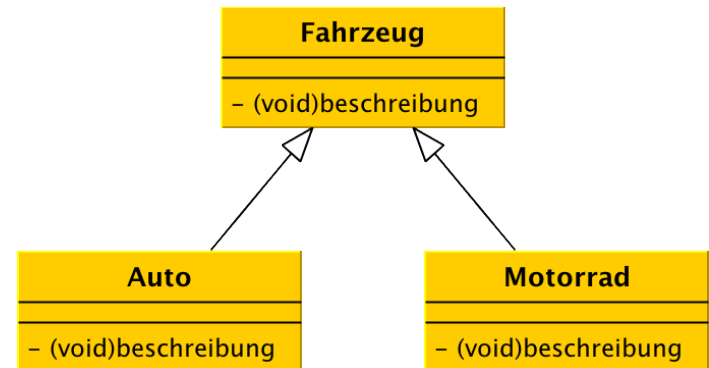
- (void)beschreibung {
    NSLog(@"Ich bin ein Auto.");
}

@end

@implementation Motorrad : Fahrzeug

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

@end
```



Beispiel dynamisches Binden (2)

```
@implementation ViewController

- (void)viewDidLoad {
    NSArray *fuhrpark = [NSArray arrayWithObjects:[[[Fahrzeug alloc] init] autorelease],
                                                    [[[Auto alloc] init] autorelease],
                                                    [[[Motorrad alloc] init] autorelease],
                                                    nil];

    for (Fahrzeug *fahrzeug in fuhrpark) [fahrzeug beschreibung];
}

@end
```

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

Darstellung Fließkommazahlen (1)

- 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 **floats** (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

- Mantisse:
 - Es gilt $1 \leq M < 2$ (normalisiert nach IEEE 754), daher sind nur Nachkommastellen zu speichern

Quelle: [3]

Darstellung Fließkommazahlen (2)

- Mantisse:
 - $M = 2^0 + 2^{-1} + 2^{-2} + 2^{-3} + \dots + 2^{-n}$
mit $n = 23$ für **float** und $n = 52$ für **double**
- ▶ 0,3 z.B. lässt sich somit gerade **nicht** abbilden
(periodische Binärdarstellung)
- ▶ Durch Binärdarstellung kumulieren sich Fehler
bei Rechenoperationen

Quelle: [3]

Darstellung Fließkommazahlen (3)

- Exponent:
 - 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: [3]

Rechenbeispiel Fließkommazahl

Aufgabe: 18,4 umrechnen in Binärdarstellung (32bit) 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:

$$B = 2^{(n-1)} - 1 = 2^{(8-1)} - 1 = 2^7 - 1 = 128 - 1 = 127$$

5. Berechnung des Exponenten:

$$\text{Exponent} + \text{Bias} = 4_{10} + 127_{10} = 131_{10} = 10000011_2$$

6. Gleitkommazahl bilden:

$$0 \ 10000011 \ 00100110011001100110011$$

Quelle: [5]

Umsetzung variadische Funktion

```
- (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: [4]

Weiterführende Links

- Matt Thompson: „BOOL / bool / Boolean / NSCFBoolean“ (<http://nshipster.com/bool/>, 2013)

Stand: 14. November 2013

Einführung in Objective-C

QUELLEN

Quellen (1)

- 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)

- [1] TIOBE Software BV: „TIOBE Programming Community Index for October 2013“ (<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.htm>, 2013)
- [2] Craig Buckler: „What is the Best Programming Language to Learn in 2013?“ (<http://www.sitepoint.com/best-programming-language-of-2013/>, 2013)
- [3] Smode, Dieter: „Fließkommadarstellung und Problembehandlung“ (<http://www.mpdvc.de/artikel/FloatingPoint.htm>, 2004)
- [4] Gallagher, Matt: „Variable argument lists in Cocoa“ (<http://www.cocoawithlove.com/2009/05/variable-argument-lists-in-cocoa.html>, 2009)
- [5] Wikipedia: „Gleitkommazahl“ (<http://de.wikipedia.org/wiki/Gleitkommazahl>, 2013)

Stand: 14. November 2013