

C & Objective-C
Kurzreferenz

Mario Gleirscher
überarbeitet von Martin Uhl

4. Mai 2004

Inhaltsverzeichnis

1	Kurzreferenz zu C	2
1.1	Allgemeine Programmstruktur	2
1.2	Gültigkeitsbereiche & Blöcke	2
1.3	Funktionen	2
1.4	Operatoren	3
1.5	Zeiger	3
1.6	Datentypen	4
1.6.1	Elementare Datentypen	4
1.6.2	Ein- und mehrdimensionale Felder	4
1.7	Wichtige Kontroll- und Schleifenanweisungen	5
1.8	Ein einfacher C-Programmrahmen	5
2	Kurzreferenz Objective-C	6
2.1	Allgemeines	6
2.2	Typen & Variablen: Definition, Sicherheitskonzepte	6
2.3	Klassen: Definition, Implementierung und Verwendung	7
2.4	Objekte & Messages	7
2.5	Methoden: Definition, Implementierung	8
2.6	Statische und dynamische Typisierung	9
2.7	Protokolle	9
2.8	Kategorien	9
2.9	Namenskonventionen	9
2.10	Ein einfacher Objective-C-Programmrahmen	10
3	Beispiel Java ↔ Objective-C	10
3.1	Das Java-Programm HelloWorld.java	10
3.2	Das Objective-C-Programm	11
3.2.1	Headerdatei HelloWorld.h	11
3.2.2	Implementation HelloWorld.m	11
3.2.3	Einstiegsdatei Main.m	12
3.2.4	Anmerkungen	13
A	Literaturangaben	13

1 Kurzreferenz zu C

Diese Referenz erhebt keinen Anspruch auf Vollständigkeit, sondern sie soll die Grundlagen zum besseren Verständnis der Sprache *Objective-C* kurz erläutern. Deshalb wird im Folgenden nur das wichtigste und notwendige erklärt. Bei den weiter unten vorkommenden Syntaxbeschreibungen handelt es sich nicht um eine vollständig korrekte BNF (Backus-Naur-Form). Details können den im Literaturverzeichnis angegebenen Referenzen entnommen werden.

C ist eine imperative, nicht-objektorientierte Hochsprache, welche sehr hohe Anforderungen an Effizienz und Leistungsfähigkeit erfüllen kann. Aufgrund dieser Ausrichtung bewegt man sich in C im Vergleich zu Java auf einem viel hardwarenäheren Level.

Die Sprache ist syntaxmäßig abgesehen vom Zeigerkonzept und einigen anderen Feinheiten vielen anderen gleichwertigen Programmiersprachen ähnlich bzw. gleich. Im Gegensatz zu Java benötigt C keine Laufzeitumgebung.

Für genauere Informationen über C siehe [1].

1.1 Allgemeine Programmstruktur

Wie auch in den meisten anderen Programmiersprachen besteht ein C-Programm aus Anweisungen, welche durch ein Semikolon ; getrennt werden.

```
<Anweisung>; <Anweisung>;  
<Anweisung>;
```

Eine Anweisung kann über mehrere Zeilen lang sein und endet immer mit einem Semikolon.

1.2 Gültigkeitsbereiche & Blöcke

Funktionen und Variablen sind nur in dem Bereich gültig, in dem sie definiert wurden. Es gibt einen globalen Gültigkeitsbereich außerhalb der Einstiegsfunktion `int main(...)`, also innerhalb der gesamten Datei. Lokale Gültigkeitsbereiche — auch Blöcke genannt — werden durch ein Paar geschwungener Klammern `{ ... }` gekennzeichnet. Nach einem solchen Block wird kein Semikolon als Anweisungsbegrenzer angefügt.

1.3 Funktionen

Funktionen werden deklariert, bevor sie aufgerufen werden. Die Implementation kann dann an einer beliebigen Stelle erfolgen. Die Deklaration kann wegfallen, wenn alle Aufrufe der Funktion erst später erfolgen.

Deklaration:

```
<Rückgabety> <Funktionsname>  
( <Parametertyp> <Parametername>, ... );
```

Implementation:

```

<Rückgabety> <Funktionsname> ( <Parametertyp> <Parametername>, ... ) {
    // Funktionsrumpf
}

```

Beispiel:

```

int fakultaet( int n ) {
    if ( n <= 0 )
        return 1;
    else
        return fakultaet( n - 1 ) * n;
}

```

1.4 Operatoren

Einige wichtige Operatoren:

Operator	Arität	Erklärung
+ , - , * , /	2	Arithmetische Operatoren
++ , --	1	Inkrement- u. Dekrementoperator, Anwendung vor oder nach einem Ausdruck ++Ausdruck, Ausdruck--
&&	2	Logisches Und
	2	Logisches Oder
&	1	Adressoperator, liefert die Speicheradresse einer Variable
*	1	1. Definiert eine Zeigervariable in einer Variablen- deklaration
*	1	2. dereferenziert eine Zeigervariable

1.5 Zeiger

In C gibt es das Konzept der Zeiger oder zu englisch "Pointer". Ein Zeiger ist eine Variable, welche die Speicheradresse einer anderen Variablen enthält. Zeiger können jederzeit "verbogen" werden, in dem man ihnen die Adresse einer anderen Variablen zuweist. Um auf den Wert der Variablen zuzugreifen, muss ein Zeiger dereferenziert werden.

Ein Beispiel:

```

:
:
int Zahl = 333;          // Eine Variable vom Typ int
int *Zeiger = &Zahl;   // Zeiger auf eine Variable vom Typ int

```

```
int Zahl2 = *Zeiger; // Zuweisung der Variable Zahl an Zahl2

// Zahl2 hat nun den Wert 333

:
```

1.6 Datentypen

1.6.1 Elementare Datentypen

Folgende elementare Datentypen werden unterstützt:

- void ...Variable ohne Typ
- char ...ASCII-Zeichen (8 bit)
- short int ...Integer (16 bit)
- long int ...für große Zahlen (32 bit)
- float ...Gleitkommazahlen (32 bit)
- (long) double ...große Gleitkommazahlen (64 bit bzw. 80 bit)

Die Schlüsselworte `signed` oder `unsigned` werden zur Spezifikation der Vorzeichenbehaftung verwendet.

Die genauen Wertebereiche hängen von der jeweilig verwendeten Architektur ab. Sie können mit der `sizeof()` Funktion festgestellt werden.

1.6.2 Ein- und mehrdimensionale Felder

Eindimensionale Felder werden wie folgt definiert:

```
<Typ> <Variablenname>[ <Elementeanzahl> ];
```

Definition mehrdimensionaler Felder:

```
<Typ> <Variablenname>[ <El.anz. 1. Dim.> ]
                    [ <El.anz. 2. Dim.> ]
                    ...
                    [ <El.anz. n. Dim.> ];
```

Auf ein Element greift man wie folgt zu:

```
int array[ 10 ]; array[ 3 ] = 28;
```

Wobei zu beachten ist, dass die Zählung der Elemente immer von 0 an beginnt. Ein Array mit 5 Elementen hat also Elemente von 0 bis 4. Hier muss mit besonderer Vorsicht ans Werk gegangen werden, denn Arraygrenzen werden im Gegensatz zu Java nicht überprüft!

1.7 Wichtige Kontroll- und Schleifenanweisungen

Definition der if-Anweisung:

```
if ( <Ausdruck> )
    <Block> | <Anweisung>;
```

Definition einer for-Schleife:

```
for ( <Ausdruck>; <Ausdruck>; <Ausdruck>; )
    <Block> | <Anweisung>;
```

Definition einer while-Schleife:

```
while ( <Ausdruck> )
    <Block> | <Anweisung>;
```

Definition einer do while-Schleife:

```
do
    <Block> | <Anweisung>;
while ( <Ausdruck> )
```

1.8 Ein einfacher C-Programmrahmen

```
/* Ein einfacher C-Programmrahmen
 *
 * Hier werden evtl. benötigte Bibliotheksheader
 * eingebunden,
 * wie z. B.:
 */
#include <stdio.h>

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

    /*
     * Hier steht der Programmcode
     */

    return 0;
}
```

2 Kurzreferenz Objective-C

Die folgenden Kapitel sollen einige Grundkenntnisse über das Wesen und die Syntax der Programmiersprache vermitteln. *Objective-C* ist eine objektorientierte Erweiterung von C. C ist eine Untermenge von *Objective-C* und damit vollständig verwendbar.

Die folgende Einführung setzt Kenntnisse über objektorientierte Konzepte und Programmierung voraus und verwendet teilweise Java, um dem Lernenden einige Analogien zu bieten.

Als Hilfe zum Verständnis der Syntax soll das weiter hinten angefügte Beispiel `HelloWorld` dienen.

2.1 Allgemeines

Objective-C hat im Aufbau der Objekthierarchie Ähnlichkeit mit Java. `NSObject` ist die Standardwurzelklasse. Gewöhnliche Objekte sind von ihr abgeleitet. Dieses Mutterobjekt und viele andere wichtige Objekte, Typen und Variablen werden im Cocoa-Framework bereitgestellt. Dieses Framework kann mit dem Java-API verglichen werden. Diese Elemente arbeiten sehr nahe mit dem sogenannten *Objective-C*-Runtimesystem zusammen, welches die elementaren Routinen und Schnittstellen für ein *Objective-C*-Programm mitbringt.

In jedem *Objective-C*-Framework muss es eine solche Wurzelklasse geben, die elementare Funktionalität zur Verfügung stellt. Nach Bedarf kann auch selbst eine neue Wurzelklasse definiert werden, was jedoch selten üblich ist.

2.2 Typen & Variablen: Definition, Sicherheitskonzepte

Es gibt in *Objective-C* abgesehen von den elementaren C-Datentypen noch einige spezielle, vordefinierte Datentypen:

- `id` ... ein typunabhängiger Zeiger auf ein Objekt
- `Class` ... ein Zeiger auf eine Klasse
- `BOOL` ... ein boole'scher Wert: YES oder NO
- `SEL` ... ein Selektor: Identifikator für einen Methodennamen

Sinnvolle Bezeichner (in `objc.h` definiert):

- `nil` ... ein Null-Objekt-Zeiger (vgl. `null` in Java)
- `Nil` ... ein Null-Klassen-Zeiger

Als sogenannter "typenloser Typ" dient der Typ `void`.

Innerhalb von Klassendefinitionen kann man Variablen analog zu Java schützen, indem man ihnen die Schlüsselwörter `@private`, `@protected` oder `@public` voranstellt. Standardmäßig ist eine Variable `@protected`. Weiteres zum Kapselungskonzept siehe [3].

2.3 Klassen: Definition, Implementierung und Verwendung

Objective-C-Klassen werden in einer nicht notwendig gleichnamigen Header-Datei (Suffix *.h) deklariert und in einer Quelldatei (Suffix *.m) werden ihre Methoden implementiert.

Definition einer Klasse:

```
@interface <Klassenname> : <Mutterklasse>
{
    \\ Deklaration von Instanzvariablen
}
\\Deklaration von Methoden
@end
```

Implementation ihrer Methoden:

```
@implementation <Klassenname> : <Mutterklasse>
    \\Implementation von Methoden
@end
```

2.4 Objekte & Messages

Objekte werden von den Klassen instantiiert:

```
// Deklaration einer Variable
<Klassenname> <Instanzvariable>;

// Ein neues Objekt im Speicher anlegen
<Instanzvariable> = [[<Klassenname> alloc] init];

// Das Objekt wieder aus dem Speicher loeschen
[<Instanzvariable> release];
```

Mit den bereits vordefinierten Methoden `alloc` und `init` werden ein Speicherbereich für eine Objektinstanz reserviert und die Objektdaten initialisiert. Nun kann mit der Objektinstanz gearbeitet werden.

Zugriffe auf die Methoden einer Klasse werden in *Objective-C* als sogenannte "Messages" bezeichnet.

Definition einer "Message":

```
[<Instanzvariable> <Methodenaufruf>];
```

In *Objective-C* gibt es eine Wurzelklasse, von der alle anderen Klassen abgeleitet werden. Diese Klasse heißt `NSObject` und stellt grundlegende Funktionen des Runtime-Systems zur Verfügung, wie z. B. die Methoden `alloc`, `init` und `release`.

2.5 Methoden: Definition, Implementierung

Allgemeine Deklaration:

Ohne Parameter:

```
+(<Rückgabety>) <Methodenname>; //Klassenmethode
-(<Rückgabety>) <Methodenname>; //Instanzmethode
```

Mit einem oder mehreren Parametern:

```
-(<Rückgabety>) <Methodenname>:(<Parametertyp1><Parametername1>
    <Methodenname_Teil2>:(<Parametertyp2>
        <Parametername2>
    ...;

```

Die selbe Definition kann auch mit einem + am Anfang zur Definition einer Klassenmethode geschrieben werden.

Im Gegensatz zu Java können den Parametern einer Objective-C Methode Namen zugeordnet werden, welche auch mit in die Signatur einfließen. So wird `initWithString:length:` von `initWithString:upTo:` unterschieden. Es gibt jedoch kein Argument Overloading.

Falls eine Methode keine Werte zurückgibt, muss sie mit dem Rückgabety `void` deklariert werden. Die Typdeklaration ist analog zur C-Cast Syntax.

Implementierung von Methoden:

```
<Methodendeklaration>
{
    // Methodenrumpf
}
```

Es gibt, wie oben definiert, zwei Arten von Methoden in *Objective-C*:

- Klassenmethoden: Diese Methoden können aufgerufen werden, ohne vorher eine Instanz derselben Klasse erzeugen zu müssen. Es wird ein + vor die Methodendeklaration angefügt.
- Instanzmethoden: Es wird ein - vor die Methodendeklaration angefügt. Instanzmethoden sind zu einer jeweiligen Instanz zugehörig, und können auch erst verwendet werden, nachdem diese angelegt wurde.

Ähnlich wie in Java gibt es in Objective-C auch Klassenobjekte, jedoch keine Klassenvariablen. Klassenmethoden können, ohne eine Objektinstanz erzeugen zu müssen, in Messages an Klassenobjekte verwendet werden. So geschieht es z. B. beim instantiieren von Objekten mit der vordefinierten Methode `alloc`.

Innerhalb von Methoden kann zusätzlich über den Zeiger `self` auf Felder und Methoden der eigenen Klasse zugegriffen werden. In Java geschieht dies mit dem Schlüsselwort `this`. Ganz analog zu Java kann Objective-C mit dem Zeiger `super` auf vererbte Methoden zugreifen.

2.6 Statische und dynamische Typisierung

Es gibt folgende zwei Methoden, um einer Variablen einen Typ zuzuordnen:

- **Dynamische Typisierung**
Mit einer Anweisung `id <Instanzvariable>;` wird zur Übersetzungszeit nur festgelegt, dass diese Variable ein Objekt darstellt. Der Typ des Objektes wird dann zur Laufzeit zugeordnet, was sich als sehr flexibel erweist.
- **Statische Typisierung**
Mit einer Anweisung `<Klassenname> *<Instanzvariable>;` wird ein herkömmlicher C-Zeiger auf ein Objekt des Typs `<Klassenname>` erzeugt. Somit ist der Typ dieser Instanzvariable schon zur Übersetzungszeit festgelegt.

2.7 Protokolle

Ein Protokoll in Objective-C ist eine Liste von Methoden, welche zu keiner bestimmten Klasse gehören. Dieses Konzept ist dem Interface-Konzept von Java ähnlich. Klassen, welche ein solches Protokoll verwenden, müssen die vorgegebenen Methoden für sich implementieren. Protokolle werden folgendermaßen definiert:

```
@protocol
    // Deklaration von Methoden
@end
```

Protokolle werden in eine Klassendeklaration folgendermaßen aufgenommen:

```
@interface KlassenName : IhreSuperKlasse < ProtokollListe >
```

Weiteres dazu siehe [2, Kapitel Protocols, Seite 32].

2.8 Kategorien

Kategorien können verwendet werden, um zu einer bereits deklarierten und implementierten Klasse direkt neue Methoden hinzuzufügen, ohne dabei den originalen Quellcode modifizieren und eine neue Unterklasse definieren zu müssen. Weiteres dazu siehe [2, Kapitel Categories, Seite 31].

2.9 Namenskonventionen

In *Objective-C* gibt es für die Benennung von Klassen, Objektinstanzen, Variablen, usw. Namenskonventionen.

- Variablen und Methoden beginnen immer mit einem Kleinbuchstaben.
- Klassen-, Protokoll- und Kategorienamen beginnen immer mit einem Großbuchstaben.

2.10 Ein einfacher Objective-C-Programmrahmen

```

// Runtimesystem einbinden
// #import <objc/objc.h>
// Unter Mac OS X
#import <Foundation/Foundation.h>
// Für Fensteranwendungen
// #import <Cocoa/Cocoa.h>

/*
 * Hier werden evtl. benoetigte Bibliotheksheader
 * eingebunden,
 * wie z. B.:
 *
 * stdio.h:
 * Standard-C-Bibliotheksfunktionen
 * fuer Ein- und Ausgabe
 *
 * Cocoa/Cocoa.h:
 * Das Apple Cocoa Framework fuer
 * die Entwicklung von
 * Aqua-Anwendungen (s.o.)
 */

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

    /*
     * Hier steht der C- und ObjC-Programmcode
     * in beliebiger, jedoch korrekter Mischung
     */

    return 0;
}

```

Mit der Anweisung `#import <...>` werden analog zu `#include <...>` Header-Dateien eingebunden. Jedoch wird automatisch sichergestellt, dass jede Header-Datei nur einmal eingebunden wird.

3 Beispiel Java ↔ Objective-C

Das folgende Beispiel zeigt die Analogien zwischen Java und Objective-C an zwei äquivalenten Programmen.

3.1 Das Java-Programm HelloWorld.java

```
import java.lang.System;
```

```

public class HelloWorld {
    private String text;

    public void set( String newText ) {
        System.out.println( "Text setzen ...\n" );
        text = newText;
    }

    public void print() {
        System.out.println( "Text ausgeben:\n" + text );
    }

    public static void main( String[] argv ) {
        HelloWorld meinObjekt = new HelloWorld();
        meinObjekt.set( "Das ist mein erstes Java-Programm!\n" );
        meinObjekt.print();
    }
}

```

Die "Garbage Collection" in Java sorgt für das freigeben von besetztem Speicher nach Beendigung des Programmes.

3.2 Das Objective-C-Programm

3.2.1 Headerdatei HelloWorld.h

```

/*
 * HelloWorld.h
 */

// Runtime System einbinden
#import <Foundation/Foundation.h>

@interface HelloWorld : NSObject
{
    @private
    id text;
}
-setText:(NSString *)newText;
-print;
@end

```

3.2.2 Implementation HelloWorld.m

```

/*
 * HelloWorld.m

```

```

*/

#import "HelloWorld.h"

@implementation HelloWorld

-setText:(NSString *)newText {
    // retain: Referenzzähler von übergebenem Objekt
    // inkrementieren
    [newText retain];
    // release: altes text-Objekt freigeben u.
    // evtl. löschen
    [text release];
    NSLog( @"Text setzen ...\n\n" );

    text = newText;
}

-print {
    NSLog( @"Text ausgeben:\n" );
    NSLog( text );
}

@end

```

Erläuterung zu HelloWorld.m:

Die Methode `release` dekrementiert den Referenzzähler und löscht das bezeichnete Objekt, wenn der Zähler auf 0 geht. Die Methode `retain` inkrementiert den Referenzzähler auf das bezeichnete Objekt. Mit diesen Methoden wird eine saubere Speicherverwaltung mit den Objektinstanzen möglich. Es kann somit vermieden werden, dass Objekte gelöscht werden, wenn noch Zeiger auf diese existieren. Bestehende Zeiger auf bereits gelöschte Objekte sind sehr gefährlich und daher "verboten".

Wichtig ist auch anzumerken, dass der Referenzzähler zuerst erhöht, und dann erniedrigt wird, denn wenn dasselbe Objekt übergeben würde das bereits schon gespeichert ist, bestünde die Gefahr, dass das Objekt durch das erste `release` bereits freigegeben wird, und so nicht mehr zur Verfügung steht.

3.2.3 Einstiegsdatei Main.m

```

/*
 * main.m
 */

// Klasse einbinden

```

```
#import "HelloWorld.h"

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

    // Objekt deklarieren
    HelloWorld *meinObjekt;    // statische Typisierung
    // id meinObjekt;          // dynamische Typisierung

    // Objekt instantiieren und initialisieren
    meinObjekt = [[HelloWorld alloc] init];

    // Mit dem Objekt arbeiten
    [meinObjekt setText:@"Das ist mein erstes
                        Objective-C Programm!\n"];
    [meinObjekt print];

    // Objekt aus dem Speicher loeschen
    [meinObjekt release];

    return 0;
}
```

3.2.4 Anmerkungen

Das obige Beispielprogramm verwendet elementare Teile des Cocoa-Frameworks. In Objective-C gibt es einen Operator @"", mit dem Objekte vom Typ NSString erzeugt werden können. NSString ist das Äquivalent zur Java-Klasse String. Für eine einfache Bildschirmausgabe wird hier analog zu System.out.println() die Funktion NSLog() verwendet. NSObject, NSString, NSLog und noch viele andere elementare Funktionen und Klassen werden bereits im Cocoa-Foundation-Framework bereitgestellt.

A Literaturangaben

Literatur

- [1] The C Programming Language (second edition, 1988) , Brian W. Kernighan and Dennis M. Ritchie, Prentice Hall
- [2] Objective-C - Einführung:
Learning Cocoa, O'Reilly 2001, Kapitel 3
- [3] Java SDK 1.4 Dokumentation unter
<http://java.sun.com/j2se/1.4/>

- [4] Ausführliche Dokumentationen über Objective-C und Cocoa unter <http://developer.apple.com/techpubs/macosx/Cocoa/CocoaTopics.html>
- [5] Die Dokumentation aus [4] ist größtenteils auch in der Xcode Hilfe enthalten.