mitp

Holger
Hinzberg

3. Auflage

Modern Objective-C und Cocoa

Praxiseinstieg

Programmierung für Mac® OS X und iPhone

Holger Hinzberg

Modern Objective-C und Cocoa

Praxiseinstieg

Programmierung für Mac OS X und iPhone

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-8266-9907-8
3. Auflage 2014

www.mitp.de
E-Mail: kundenservice@hjr-verlag.de
Telefon: +49 6221 / 489 -555
Telefax: +49 6221 / 489 -410

Lektorat: Sabine Schulz
Korrektorat: Manfred Buchholz
Coverbild: © Max Krasnov
Satz: III-satz, Husby, www.drei-satz.de

Inhaltsverzeichnis

Danksagung

Als ich mit der Arbeit zur dritten Auflage dieses Buches begonnen habe, war mir nicht bewusst, dass mich diese Aufgabe mehrere Monate beschäftigen würde, aber die Geschwindigkeit, mit der sich Objective-C in den letzten Jahren weiterentwickelt hat, ist bemerkenswert. Funktionen, wie die automatische Speicherverwaltung und der Punktoperator, die es in anderen Programmiersprachen schon lange gibt, haben endlich auch ihren Weg zu den Plattformen der Firma Apple gefunden. Somit gab es für mich viele Themen, die neu erklärt werden mussten. Zu meinem Glück hatte ich dabei wieder tatkräftige und kompetente Unterstützung.

An erster Stelle geht mein Dank an Horst Andreas Roemer, denn dieses Buch enthält sehr viel von seinen Erfahrungen aus dem Alltag eines Softwareentwicklers. Ich bin sicher, alle Leser werden ihm für die zusätzliche Praxis in diesem Praxiseinstieg danken.

Dank geht an meinen Freund Peter Mekelburg, der schon seit vielen Jahren bereitwillig alle meine Texte prüft und mich davor bewahrt, mit zu vielen fehlenden Satzzeichen unangenehm aufzufallen.

Stellvertretend für alle Mitarbeiter des Verlags danke ich meiner geduldigen Lektorin Sabine Schulz, die mir erneut die Gelegenheit gab, meine Ideen und Erfahrungen als Buch zu veröffentlichen. In der Korrektur sorgte Manfred Buchholz wieder dafür, dass (im besten Fall) kein Fehler unentdeckt blieb.

Mein besonderer Dank geht an alle Leser meiner Bücher, von denen ich einige inzwischen persönlich auf Konferenzen und Seminaren kennenlernen durfte. Die Softwareentwicklung bleibt ein spannendes Thema und so konnte ich auch dieses Mal nicht alle eure Wünsche erfüllen. Vielleicht klappt es bei der nächsten Auflage!

Holger Hinzberg
Holzwickede, Mai 2014

Einleitung

Es ist vielleicht schwer zu glauben, aber bis vor wenigen Jahren war die Programmiersprache Objective-C nur wenigen Entwicklern vertraut. Der Marktanteil der Firma Apple war klein und auf anderen Plattformen konnte sich Objective-C als Sprache nie erfolgreich durchsetzen. Seitdem ist jedoch viel passiert. Die steigende Popularität von Mac- und iOS-Geräten führte zu einer regelrechten Wiederentdeckung der Sprache und viele Menschen fanden Gefallen an der manchmal etwas ungewöhnlichen Art der Softwareentwicklung für Computer und portable Geräte der Firma Apple.

War Literatur zur Softwareentwicklung für OS X und iOS am Anfang schwer zu finden und schwer zu verstehen, so gibt es inzwischen eine große Flut an Büchern zu diesen Themen. Sehr viele davon sind jedoch zu oberflächlich oder stellen zu hohe Ansprüche an die Leser – zwei Fehler, die ich mit diesem Buch nicht wiederholen möchte. So brachte mich die Unzufriedenheit mit den vorhandenen Büchern dazu, selbst tätig zu werden.

Die Beispiele in diesem Buch sind einfach gehalten. Einfach, das bedeutet in manchen Fällen nicht sehr lebensnah, aber vielleicht genau deshalb besser dazu geeignet, Grundlagen wirklich zu erklären. Der Name »Praxiseinstieg« bedeutet auch, dass Sie mit diesem Buch wirklich selbst Programme entwickeln und die Seiten mit Programmcode und weniger mit Klassendiagrammen gefüllt sind. Sie sollen erfahren, wie die Dinge funktionieren, und Sie werden lernen, wie man eine Anwendung erstellt und die nötigen Werkzeuge bedient. Auch gibt es in diesem Buch kein Projekt, das sich vom ersten bis zum letzten Kapitel entwickelt, sondern viele kleine Bausteine, die gezielt Themen behandeln, welche auf den ersten Blick wenig miteinander zu tun haben, aber dennoch in vielen Anwendungen zum Einsatz kommen.

Die Firma Apple bietet mit iOS App Store und Mac App Store inzwischen zwei Plattformen, die es selbst Programmieranfängern ermöglichen, ihre Software einem internationalen Publikum anzubieten. Alles, was Sie benötigen, ist eine gute Idee und vielleicht ein wenig Talent, denn wie Sie auf den folgenden Seiten erfahren werden, ist die Entwicklung eigener Programme gar nicht so kompliziert, wenn man den richtigen Einstieg findet.

Einen Einstieg für Objective-C und Cocoa zu schreiben stellt jedoch für jeden Autor eine Herausforderung dar, denn es ist schwierig, einen Anfang und ein

Ende zu finden. Nicht weil die Themen schwierig oder kompliziert sind, sondern wegen der Vielzahl an Möglichkeiten, die OS X und iOS inzwischen bieten. Die Mitarbeiter von Apple haben bei der Entwicklung beider Systeme hervorragende Arbeit geleistet und man sollte es dem Unternehmen hoch anrechnen, dass es seine Klassenbibliotheken und Entwicklerwerkzeuge der Öffentlichkeit, also auch Ihnen und mir, zur Verfügung stellt.

Dieses Buch heißt Praxiseinstieg und es ist daher die Praxis, die im Vordergrund stehen soll. Die Beispiele sind so ausgelegt, dass sie sehr leicht programmiert werden können, und das ist auch nötig, um Objective-C und Cocoa wirklich kennenzulernen. In den ersten beiden Kapiteln werden Sie noch keinen eigenen Programmcode schreiben, dafür erfahren Sie Wissenswertes über die Entstehung und den Aufbau der Sprache Objective-C, das CocoaFramework und einige andere Dinge, die Sie vielleicht überraschen werden.

An wen richtet sich das Buch?

Trotz aller guten Vorsätze, eine wirklich einfache Einführung zu schreiben, fängt auch dieses Buch leider nicht bei null an. Dafür reichen die zur Verfügung stehenden Seiten nicht aus. Wenn Sie als Leser aber schon ein wenig Erfahrung mit einer anderen objektorientierten Programmiersprache haben und jetzt auch OS X oder iOS für sich entdeckt haben, dann ist es das richtige Buch für Sie und die Beispiele sollten keine zu großen Hürden sein. Dabei ist es unerheblich, ob Sie Vollzeitentwickler oder nur Hobbyprogrammierer sind. Sind Begriffe wie Compiler, Klasse und Vererbung für Sie keine Fremdwörter, dann spricht nichts dagegen, dass Sie ebenfalls ein erfolgreicher Programmierer für die Apple-Plattformen werden.

Wenn Sie allerdings noch gar keine Erfahrungen mit einer Programmiersprache gemacht haben, dann ist alternativ das Buch »Mac-Programmierung für Kids« empfehlenswert, für das keinerlei Vorkenntnisse vorausgesetzt werden.

Aufbau des Buches

Ein Überblick über die Entwicklungsumgebung Xcode in der Version 5 und ein grober Überblick über die Grundlagen der Programmiersprache Objective-C sind der erste Teil des Buches. Hier lernen Sie Ihr Handwerkszeug und die ersten Befehle kennen, um schon eigene kleine Programme zu schreiben. Auf eine grafische Oberfläche wird in diesem Teil noch verzichtet, um nicht zu sehr von den Grundlagen abzulenken und das Lernen von vielen neuen Dingen nicht unnötig zu komplizieren. Trotzdem erfahren Sie, wie man mithilfe verschiedener Werkzeuge den Programmablauf verfolgen kann und den Debugger einsetzt, um Fehler zu finden.

Die Entwicklung von Anwendungen mit grafischen Benutzeroberflächen füllt den zweiten Teil des Buches. Hier lernen Sie, wie Textfelder, Schaltflächen und viele andere Steuerelemente funktionieren und wie so eine Anwendung aufgebaut ist. MVC-Architektur, Delegation und Sprachausgabe sind nur einige der behandelten Themen.

Der dritte Teil des Buches beschäftigt sich mit der Programmierung für das iOS-Betriebssystem, also für das iPhone, iPad und den iPod touch. Die Entwicklung einer »App« ist der eines OS-X-Programms ähnlich, allerdings bieten diese Geräte vollkommen andere Möglichkeiten der Bedienung. Mithilfe eines Simulators können programmierte Apps direkt auf dem Mac getestet werden, Sie benötigen kein iOS-Gerät für diese Beispiele.

Im Internet

Sollten sich Fehler im Buch finden oder Apple Anpassungen an seinen Entwicklungswerkzeugen vornehmen, werde ich versuchen, diese auf meiner Webseite zu beschreiben, damit Sie auch weiterhin problemlos mit dem Buch arbeiten können. Sie finden die Seite im Internet unter der Adresse:

`http://www.cocoa-coding.de/praxis`

Auf der Webseite finden Sie außerdem die einzelnen Beispielprojekte zum Download, aber auch Informationen zu anderen Themen, die es nicht in das Buch geschafft haben.

Bei Fragen zum Buch oder wenn Sie Fehler finden, können Sie sich gerne auch direkt an mich wenden, meine E-Mail-Adresse finden Sie ebenfalls auf der Webseite.

Teil I

Grundlagen von Objective-C

In diesem Teil:

Kapitel 1

Grundlagen

Hat man sich als Entwickler entschlossen, für die Apple-Plattform zu programmieren, gibt es verschiedene Möglichkeiten diesen Entschluss umzusetzen. Natürlich steht auf dem Mac die plattformübergreifende Sprache Java zur Verfügung und auch andere Programmiersprachen können eingesetzt werden. Möchte man aber alle Möglichkeiten des Systems ausnutzen und die Werkzeuge benutzen, wie sie auch bei der Firma Apple im Einsatz sind, fällt die Wahl schnell auf Objective-C und Cocoa. Zwei sehr verschiedene Dinge, die doch gemeinsam ein starkes Team bilden.

Objective-C ist eine Programmiersprache, und wie der Name es vermuten lässt, ist sie eine Weiterentwicklung der Sprache C. Ähnlich wie C++ ist auch sie eine Ergänzung von C um die Möglichkeit der objektorientierten Programmierung. Allerdings ist Objective-C eine Obermenge der Sprache C und Programmbausteine, die in ANSI-C geschrieben sind, lassen sich problemlos in Objective-C-Programmen verwenden. Auch liefert C den überwiegenden Teil des Sprachaufbaus. Variablendeklarationen, Schleifen, Fallunterscheidungen und vieles mehr entsprechen der C-Syntax und sind somit den entsprechenden Bestandteilen der meisten gängigen Programmiersprachen sehr ähnlich.

Als Sprache existiert Objective-C schon seit den frühen 1980er Jahren und wurde hauptsächlich von Brad Cox und Tom Love in der Firma Stepstone entwickelt. Beide hatten es sich zum Ziel gesetzt, die Sprache C durch Elemente der Sprache Smalltalk zu erweitern, um so eine größere Wiederverwertbarkeit von Programmcode zu erreichen. Trotzdem verbreitete sich Objective-C in den ersten Jahren nicht besonders erfolgreich und auch zu Apple kam die neue Programmiersprache erst über Umwege.

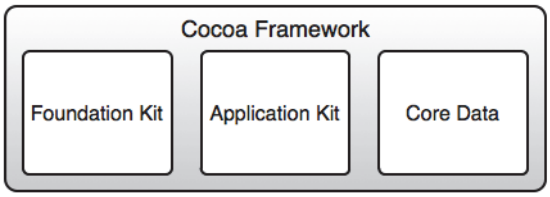
Nachdem Steve Jobs die Firma Apple, ein Unternehmen, das er mitbegründet hat, verlassen hatte, gründete er eine neue Firma mit dem Namen NeXT Computer. Bei NeXT setzte man auf Objective-C und veröffentlichte später sogar einen eigenen Compiler für diese Programmiersprache. Außerdem wurde Objective-C bei der Entwicklung des eigenen Betriebssystems NextStep (auch geschrieben als NeXTstep oder NeXTSTEP) verwendet, welches für seine Zeit schon sehr fortschrittlich war. Im Jahr 1996 übernahm Apple die Firma NeXT und aus NextStep entwickelte sich das heutige Betriebssystem OS X.

Viel wichtiger als die Programmiersprache, in der NextStep geschrieben wurde, waren für Apple aber wahrscheinlich die bei NeXT entwickelten Frameworks.

Unter einem Framework versteht man in der Softwareentwicklung eine Bibliothek aus mehreren Klassen, also eine Sammlung wiederverwendbarer Programmteile, die flexibel einsetzbar sind. Weil man bei Apple zu diesem Zeitpunkt schon fast verzweifelt nach einem neuen Betriebssystem suchte, war es wahrscheinlich keine schwierige Entscheidung aus NextStep das nächste Betriebssystem für Apple-Computer zu entwickeln. Die unterschiedlichen Frameworks, zu einem großen Teil bei NeXT entwickelt, wurden unter der Bezeichnung Cocoa zusammengefasst. Das Cocoa Framework, welches bis heute verwendet wird, kann seinen Ursprung bei NeXT nicht leugnen. Nahezu alle Klassennamen beginnen mit den Buchstaben NS für NextStep.

1.1 Das Cocoa Framework

Die häufig gebrauchte Bezeichnung Cocoa Framework ist ein wenig irreführend, denn es handelt sich bei Cocoa nicht um ein eigenständiges Produkt, sondern um den Zusammenschluss von drei einzelnen Frameworks, nämlich Foundation Kit, Application Kit und Core Data.



Das Foundation Kit Framework, oft auch nur als Foundation bezeichnet, enthält unzählige Klassen und Datenstrukturen, die es in der Sprache C nicht gibt, und ist somit die vielleicht wichtigste Erweiterung. Foundation ermöglicht spezielle Zeichenketten, Arrays, Dictionaries und viele weitere Typen. Zusätzlich übernimmt das Framework die Speicherverwaltung von Objekten.

Application Kit, abgekürzt AppKit, ist ein Framework für die Entwicklung grafischer Benutzeroberflächen. Es enthält Klassen für die unterschiedlichen Steuerelemente wie beispielsweise Fenster, Buttons oder Schieberegler. Eine weitere Aufgabe des Application Kits ist die Behandlung von Ereignissen, wie das Anklicken eines Buttons, das Verschieben eines Fensters oder die Eingabe von Text in ein grafisches Textfeld.

Core Data ist ein Framework speziell für die Verwaltung von Daten in relationalen Beziehungen.

Foundation und Application Kit wurden direkt von NeXT übernommen und in der ersten Version von OS X waren auch nur sie enthalten. Core Data wurde mit OS X

10.4 Tiger veröffentlicht, basierte aber auf dem Enterprise Objects Framework, das ebenfalls bereits bei NeXT entwickelt wurde.

Im Laufe der Jahre wurden von Apple zusätzliche Frameworks entwickelt, welche streng genommen kein Teil von Cocoa sind. Trotzdem können auch sie problemlos in Programmen verwendet werden, und sie funktionieren reibungslos mit den Teilen des Cocoa Frameworks. Zwei der bekanntesten Erweiterungen sind Core Animation und WebKit, letzteres ein Framework zur Darstellung von Internetseiten, das auch im Safari-Browser verwendet wird.

In einem Programm ist es nicht immer einfach, Objective-C und Cocoa zu trennen, denn beide sind sehr eng verzahnt. Objective-C ist als Programmiersprache für den Aufbau und die Syntax der Anweisungen verantwortlich, wie die Grammatik einer gesprochenen Sprache. Die Klassen und Objekte der Frameworks sind im Gegensatz dazu Vokabeln dieser Sprache, die zwar jede für sich eine Bedeutung haben, aber erst durch die Verwendung der richtigen Grammatik einen komplexeren Sinn ergeben.

1.2 Die typische Objective-C-Syntax

Trotz der engen Verwandtschaft mit C spricht Objective-C in der großen Familie der von C abstammenden Sprachen einen sehr eigenwilligen Dialekt. Besonders auffällig, und schon fast ein untrügliches Erkennungszeichen, ist die Verwendung von eckigen Klammern bei Befehlen, welche die objektorientierte Programmierung betreffen und bei denen es um die Kommunikation mit Objekten geht. Objective-C folgt dabei der von Alan Kay entwickelten Programmiersprache Smalltalk, die auf dem Konzept beruht, dass alle Objekte einer Anwendung voneinander unabhängig existieren und nur durch Nachrichten miteinander kommunizieren. Eine Nachricht besteht dabei aus bis zu drei Teilen: dem Empfänger, der Nachricht und eventuell zusätzlichen Parametern. Eine typische Anweisung könnte wie folgt aussehen:

```
[anObject doSomething:5];
```

Anweisungen in eckigen Klammern sind in einem Objective-C-Programm immer ein untrügliches Zeichen dafür, dass Nachrichten an Objekte gesendet werden. Im Beispielbefehl wird die Nachricht `doSomething` an das Objekt `anObject` gesendet. Die Nachricht ist gleichzeitig die Aufforderung an das Objekt, die vorhandene gleichnamige Methode `doSomething` abzuarbeiten. In dem Beispiel wird der Nachricht als Parameter die Zahl `5` mitgegeben, die ebenfalls an das Objekt gesendet wird. Eine vergleichbare Anweisung in C# sähe ein wenig anders aus:

```
anObject.doSomething(5);
```

Wichtig

Streng genommen handelt es sich bei den beiden Anweisungen um zwei verschiedene Dinge. Während die C#-Anweisung direkt ein Methodenaufruf in `anObject` ist, handelt es sich bei der Objective-C-Anweisung lediglich um das Absenden einer Nachricht an `anObject`, die dort als nächsten Schritt einen Methodenaufruf bewirkt. Um die Dinge aber nicht unnötig kompliziert zu machen, werden auch in Objective-C solche Anweisungen als Methodenaufrufe bezeichnet.

Objective-C folgt beim Aufbau der Nachrichten und der zugehörigen Methoden dem Konzept der Named Parameters, der benannten Parameter. Während in Sprachen wie C++ oder C# die Parameter als Reihe von Argumenten in einer Klammer nach dem Methodennamen übergeben werden, ist ein Objective-C-Aufruf anders aufgebaut. Die Parameter stehen direkt in der Anweisung.

```
[anObject addNumber:5 toNumber:9];
```

Durch die Benennung der Parameter bildet sich der vollständige Methodenname aus den einzelnen Bezeichnungen inklusive der Doppelpunkte und heißt somit: `addNumber:toNumber:`. Eine auf diese Nachricht passende Methode könnte so aussehen:

```
- (void)addNumber:(int)aValue toNumber:(int)bValue;
{
    c = aValue + bValue;
}
```

Diese Art der Namensgebung erlaubt sehr viele Freiheiten und ermöglicht es sogar, Methoden schon mit dem Namen gut zu beschreiben. Dem Programmierer bleibt die Aufgabe, die Parameter möglichst passend zu benennen.

Dass Nachrichten in Objective-C eine sehr große Rolle spielen, lässt sich auch an den Dateinamen der Klassen erkennen. Die Endung `.m` der Implementierungsdatei steht für Message, also Nachricht.

Wichtig

Beim Absenden einer Objective-C-Nachricht müssen die Parameter nicht zwangsläufig direkt nach dem Doppelpunkt folgen, der Compiler akzeptiert es durchaus, wenn dort Leerzeichen eingefügt werden. Besonders bei verschachtelten Anweisungen oder komplexen Parametern können Leerzeichen zur besseren Lesbarkeit beitragen.

1.3 Kontrollstrukturen

Die in Objective-C verwendeten Schleifen und Fallunterscheidungen sind den Kontrollstrukturen vieler anderer Programmiersprachen sehr ähnlich, da sie auch auf der Sprache C basieren, welche schon sehr oft als Vorbild gedient hat.

Die `if`-Struktur erlaubt im Aufbau beliebig viele `else if`-Zweige sowie einen abschließenden `else`-Zweig, wobei die gängigsten Vergleichsoperatoren größer (>), kleiner (<), größer oder gleich (>=), kleiner oder gleich (<=), gleich (==), und ungleich (!=) sind. Sollen mehrere Bedingungen überprüft werden, können boolesche Operatoren wie UND (&&) sowie ODER (||) verwendet werden.

```
int i = 1;.

if (i == 5)
{
    // i ist 5.
}
else if (i > 10 && i != 20)
{
    // i ist größer als 10,
    // aber nicht gleich 20.
}
else
{
    // i ist kleiner als 10
    // oder gleich 20.
}
```

Objekte sollte man in Objective-C nicht mit den Vergleichsoperatoren vergleichen, da dabei in den meisten Fällen nur die Speicheradressen überprüft werden, was selten zum erwarteten Ergebnis führt. Einige Klassen bieten daher spezielle Methoden, um ihre Objekte auf Gleichheit zu prüfen. Das folgende Beispiel zeigt den Vergleich zweier Zeichenketten vom Typ `NSString`, mit denen sich Kapitel 4 näher beschäftigen wird.

```
NSString *text1 = @"Hallo Welt";
NSString *text2 = @"Hello World";

if ([text1 isEqualToString:text2] == YES)
{
    // Die Zeichenketten sind gleich.
}
else
{
    // Die Zeichenketten sind ungleich.
}
```

Sind besonders viele Zweige bei einer Fallunterscheidung erforderlich, resultiert der Einsatz von `switch` und `case` in einem oftmals besser lesbaren Code. Objective-C erlaubt es, bei einem `case`-Zweig auf die `break`-Anweisung zu verzichten und somit den Programmablauf an den folgenden Fall »durchzureichen«. Der `default`-Zweig, welcher nicht zwingend erforderlich ist, wird vom Programm verwendet, wenn keine der anderen Bedingungen erfüllt ist. Im Unterschied zur `if`-Struktur kann mit `switch` aber nur auf Werte und nicht auf Wertebereiche reagiert werden.

```
int i = 2;

switch (i)
{
    case 1:
    // i ist 1.
    break;

    case 2:
    // i ist 2.
    break;

    case 3:
    // i ist 3.
    break;

    default:
    // i ist kleiner als 1
    // oder größer als 3.
    break;
}
```

Sollen Anweisungen mehrfach ausgeführt werden, dann ist dies in vielen Situationen eine Aufgabe für eine `for`-Schleife, die in Objective-C so aufgebaut ist:

```
for (int i = 1; i <= 10; i++)
{
    // Anweisungen
}
```

Falls die Laufvariable, im Beispiel `i`, auch außerhalb des Gültigkeitsbereichs der Schleife benötigt wird, genügt eine Deklaration vor der Schleife, um den Wert zu erhalten.

```
int i;

for (i = 1; i <= 10; i++)
{
    // Anweisungen
}

// i kann hier noch verwendet werden.
```

Die `while`- und `do-while`-Schleifen funktionieren ebenfalls nach dem gleichen Prinzip wie in anderen Programmiersprachen. Die Anweisungen innerhalb der Schleife werden wiederholt, bis die Abbruchbedingung erreicht ist.

```
int i = 0;

do
{
    i++;
}
while (i < 10);
// Die Abbruchbedingung i ist 10 oder größer.
```

Während eine `do-while`-Schleife immer mindestens einen Durchlauf absolviert, wird bei der `while`-Schleife die Abbruchbedingung schon zu Anfang kontrolliert. Somit wird es möglich, die Schleife, bei bereits erfüllter Bedingung, ohne Durchlauf zu überspringen.

```
int i = 10;

// Die Abbruchbedingung ist bereits erfüllt.
while (i < 10)
{
    i++;
}
```

1.4 Namenskonventionen

Für einen erfahrenen Entwickler sollte es selbstverständlich sein, Instanzen einer Klasse `Person` nicht einfach nur als `p` zu bezeichnen, sondern einen aussagekräftigeren Namen zu verwenden. Aber auch darüber hinaus gibt es einige Regeln, an die man sich halten sollte. Nicht, weil die Anwendung sonst nicht funktioniert, sondern um einen gut lesbaren und leicht verständlichen Programmcode zu erhalten. Besonders bei der Zusammenarbeit mit anderen Programmierern sollte man diesen Punkt nicht unterschätzen.

Klassennamen sollten immer mit einem Großbuchstaben beginnen, zum Beispiel `Person`, `Container` oder `Player`. Besonders bei Entwicklung in größeren Teams kann es zudem sinnvoll sein, die eigenen Initialen als Präfix des Klassennamens zu verwenden, da andere Programmierer möglicherweise den gleichen Namen für ganz andere Aufgaben verwenden. In diesem konkreten Falle hießen die Klassen, die von mir geschrieben wurden, dann `HHPerson`, `HHContainer` oder `HHPlayer`. Für die Beispiele in diesem Buch wird aber auf ein Präfix verzichtet.

Wichtig

Da die Klassen der Foundation und des Application Kit Frameworks mit den Buchstaben NS beginnen, sollte man dieses Präfix bei der Benennung von eigenen Klassen vermeiden. Da Objective-C keine übergeordneten Namensräume (englisch: Namespaces) unterstützt, könnte es sonst zu Kollisionen mit bestehenden Klassen des Frameworks kommen. Zwar kann man ermitteln, ob ein Klassenname schon vergeben ist, zukünftige Erweiterungen lassen sich aber nicht voraussehen. Auch sollte man das Präfix UI meiden, da dieses von Klassen des Cocoa Touch Frameworks verwendet wird. UI steht als Abkürzung für User Interface.

Im Unterschied zu Klassennamen beginnen Variablenbezeichnungen immer mit einem Kleinbuchstaben. Dabei ist es egal, ob es sich um einen einfachen Typen der Sprache C handelt oder ein instanziiertes Objekt in Objective-C.

```
double salary;
NSString *clubName;
Person *aPerson;
```

Bei Objekten ist es nicht unüblich, auch den Namen der Klasse, aus der sie erzeugt wurden, mit in den Variablennamen einfließen zu lassen. In diesem Beispiel ist `aPerson`, aus dem Englischen übersetzt als »eine Person«, eine Instanz der Klasse `Person`. Es gibt allerdings Ausnahmen von dieser Regel. Die Klassen `NSString`, `NSNumber` und `NSArray` sind so weit verbreitet, dass eine explizite Benennung nicht nötig ist und oft nur stört. Auch muss der Variablenname nicht immer den kompletten Klassennamen enthalten, da dieser sehr lang sein kann. Der Name soll lediglich ein Hinweis sein, um welchen Objekttyp es sich handelt.

Etwas übertrieben ist eine Bezeichnung wie:

```
NSProgressIndicator *downloadNSProgressIndicator;
```

Kürzer, und vollkommen ausreichend ist:

```
NSProgressIndicator *downloadIndicator;
```

Wichtig

In Objective-C-Programmen gelten weiterhin die in C verwendeten Regeln für Variablennamen: Sie dürfen nicht mit einer Zahl beginnen, keine Leerzeichen und keine Sonderzeichen außer dem Unterstrich enthalten. Die in anderen Programmiersprachen übliche Verwendung eines Unterstrich-Präfixes für private Variablen, wie zum Beispiel in `_salary`, ist nicht zu empfehlen. Es gibt Situationen, in denen der Compiler Variablen, die dieser Namenskonvention folgen, automatisch erzeugt. Dies könnte zu Konflikten führen.

Auch Methodennamen beginnen mit Kleinbuchstaben, ebenso alle im Aufruf verwendeten Parameter.

```
[anObject addNumber:5 toNumber:9];
```

Wird ein Bezeichner aus mehr als einem Wort gebildet, sollte der erste Buchstabe der zusätzlichen Wörter immer ein Großbuchstabe sein. Wie gezeigt, bilden die Wörter `add` und `number` den Namen `addNumber`. In der Softwareentwicklung wird diese Schreibweise auch als CamelCase bezeichnet, da die Großbuchstaben wie die Höcker eines Kamels aus dem ansonsten kleingeschriebenen Text herausragen. Weil Klassennamen immer mit einem Großbuchstaben beginnen, verwenden sie die Variante UpperCamelCase, Variablen und Methoden hingegen den LowerCamelCase. Der CamelCase erlaubt es, auch lange Namen noch relativ einfach lesbar zu halten. Unterstriche zur Trennung von einzelnen Wörtern innerhalb von Bezeichnern werden in Objective-C selten verwendet.

1.5 Kommentare

Objective-C erlaubt es, innerhalb des Programmcodes nicht nur Anweisungen, sondern auch Kommentare zu schreiben. Nicht alle Dokumentationen weisen darauf hin, aber Kommentare sind wichtig! Nicht nur für den Entwickler selbst, sondern auch für andere Menschen, die vielleicht in Zukunft mit dem Quelltext in Kontakt kommen. Natürlich sollte man es auch mit den Kommentaren nicht übertreiben. Der Programmcode sollte so geschrieben werden, dass er gut verständlich ist. Daher sollten Kommentare auch weniger genutzt werden, um zu dokumentieren »was« ein bestimmter Programmteil tut, sondern »warum«. Eine Ausnahme sind Bücher für Einsteiger, wovon Sie gerade eins lesen. Hier kann es nicht genug Kommentare geben. Sobald Ihnen die Befehle von Objective-C und Cocoa vertraut sind, werden Sie diese Art von Hilfe aber nicht mehr benötigen.

Es gibt zwei Arten Kommentare zu verwenden. Die erste und einfachste ist die Kommentarzeile, eingeleitet durch zwei Schrägstriche //.

```
// Dies ist ein Kommentar.
```

Kommentarzeilen werden oft verwendet, um Variablen zu beschreiben.

```
double salaryOfPerson; // Das Gehalt der Person
```

Möchte man mehr als eine Zeile Kommentar im Programm einfügen, können komplette Abschnitte als Kommentar gekennzeichnet werden. Mit der Zeichenfolge /* beginnt so ein Abschnitt, der mit */ wieder beendet wird. Kommentarzeilen und Kommentarabschnitte können während der Entwicklung verwendet werden, um Anweisungen vorübergehend aus dem fertigen Programm zu entfernen, ohne sie im Quelltext löschen zu müssen.

```
/*

Person *aPerson = [[Person alloc] init];

*/
```

Wie Sie später noch selbst sehen werden, sind Kommentare im Codeeditor immer in einer besonderen Farbe, in der Regel Grün, gekennzeichnet. So kann man gut erkennen, wenn Anweisungen auskommentiert wurden. Kommentare werden auch nicht in das zu erstellende Programm übernommen. Unabhängig davon, wie viele Kommentare Sie schreiben, haben diese keinerlei Einfluss auf die Dateigröße der fertigen Anwendung.

Wichtig

Vielleicht werden Sie an dieser Stelle denken, dass Sie auf Kommentare verzichten können, weil Sie immer genau wissen, was Ihre Programme tun – schließlich haben Sie diese ja selbst programmiert. Es würde ja auch niemand behaupten, dass Sie von Ihnen selbst verfasste Briefe in Zukunft nicht mehr selbst lesen könnten. Aber dies ist ein Irrtum und Sie werden es verstehen, wenn sich die ersten Dutzend unterschiedlicher Projekte auf Ihrem Rechner befinden. Dabei ist die Menge der Projekte gar nicht so ausschlaggebend, sondern viel mehr Ihr persönlicher Stil. Wenn Sie viel programmieren und dabei eine neue Programmiersprache erlernen, wird sich die Art, wie Sie Programme schreiben, ändern. Das ist ein natürlicher Prozess, denn Sie lernen ständig dazu. Komplexe Probleme, die zu Anfang noch viele Zeilen Code benötigen, werden Sie irgendwann mit wenigen Anweisungen lösen können, Ihr Code wird kürzer und besser, und Sie werden Schwierigkeiten haben, selbst geschriebene alte Programme zu verstehen. Dann werden Sie sich wundern, wie Sie nur auf all diese seltsamen Ideen gekommen sind und warum alles so kompliziert ist. Unterschätzen Sie deshalb die Macht der Kommentare nicht!