



Sebastian
Meyer

Torben
Wichers

2. Auflage

Objective-C 2.0

Programmierung für Mac OS X und iPhone

Danksagung

Bücher entstehen aus Ideen, aber nicht von alleine. Wir möchten uns bei all jenen bedanken, die uns ermutigt und unterstützt haben, dieses Buch zu schreiben und es zu dem zu machen, was es jetzt ist.

Wir wollen zuerst den Studenten danken, die an unserem Seminar zu Objective-C und Cocoa im Wintersemester 2008/2009 teilgenommen haben. Ohne sie wäre dieses Buch nie entstanden. Wir freuen uns, dass ein Seminar in diesem Bereich auch über die Grenzen der Leibniz Universität Hannover hinaus so viel Interesse hervorrufen konnte.

Unser Dank gilt weiterhin Prof. Parchmann und Prof. Schneider, die uns in vielen anregenden Gesprächen ermutigt und mit wertvollen Tipps unterstützt haben, dieses Buch zu schreiben. Ebenso allen Kollegen, die ebenfalls ermutigende Worte für uns gefunden haben.

Nicht vergessen werden sollen all jene, die in unzähligen Stunden unsere Entwürfe auf Korrektheit und Sinnhaftigkeit überprüft haben.

In diesem Sinne gilt schließlich unser besonderer Dank Verena Oesterlein für ihre Bereitschaft, uns über diese lange Zeit hinweg stets zu unterstützen sowie für ihre dafür aufgebrachte Geduld.

Über die Autoren

Sebastian Meyer und Torben Wichers haben an der Leibniz Universität Hannover Informatik studiert. Nach ihrem Abschluss als Master of Science arbeiten sie am Institut für Praktische Informatik in den Fachgebieten Software Engineering und Programmiersprachen und Übersetzer als wissenschaftliche Mitarbeiter. Zusammen veranstalten sie Seminare und Labore zu Themen der Compilerkonstruktion und Programmiersprachen wie Objective-C.

Einleitung

In diesem Buch wollen wir uns mit der Programmiersprache Objective-C sowie ihrem Einsatz im Zusammenhang mit dem Cocoa-Framework auf dem Mac und auf iOS-Geräten wie dem iPhone und dem iPad beschäftigen. Die Sprache an sich ist schon verhältnismäßig alt. Jedoch ist sie in der breiten Öffentlichkeit bislang nie richtig in Erscheinung getreten. Dies mag daran liegen, dass Objective-C so gut wie auf den Mac beschränkt ist. Mit dem wachsenden Erfolg der Macs und der Einführung des iPhones stehen die Vorzeichen für Objective-C gut. Auf beiden Plattformen, also sowohl Mac OS X als auch iOS, ist Objective-C die Standardsprache. Für das iOS ist sie sogar die einzige Sprache. Mit der Veröffentlichung von Mac OS X 10.5 (Leopard) hat Apple die Sprache auf die symbolisch bedeutsame Versionsnummer 2.0 angehoben. Für das iOS steht entsprechend auch die neue Version 2.0 der Sprache zur Verfügung. Objective-C 2.0 fügt einige neue Konzepte und Sprachmittel ein, die die Entwicklung mit dieser Sprache noch effektiver und angenehmer machen.

Insgesamt lässt sich festhalten, dass Objective-C anders ist als besser bekannte Sprachen wie Java oder C++. Anders heißt in diesem Fall jedoch nicht schlecht. Eher möchte man sagen, Objective-C ist erfrischend anders. Denn Objective-C bietet uns Programmierern in der Sprache enthaltene Hilfsmittel, die wir in anderen Sprachen so nicht besitzen. Dies liegt zum einen am Alter der Sprache. Zur damaligen Zeit gab es noch nicht so viele Vorbilder für objektorientierte Programmiersprachen. Eigentlich nur eine: SmallTalk. So ist auch C++ zur selben Zeit entstanden. Das Hauptziel von Objective-C war, die in SmallTalk entwickelten Konzepte in eine C-ähnliche effiziente Sprache zu integrieren. Und dies ist damals wirklich gelungen, wodurch sich das sehr dynamische Verhalten von SmallTalk in Objective-C sehr gut wiederfindet und sehr viel Spaß beim Programmieren bringt.

Als zweiter Glücksfall für Objective-C erweist sich die Aktualisierung der Konzepte in Objective-C 2.0. Diese Sprachneuerungen wurden im Verhältnis zum Alter der Sprache selbst recht spät eingeführt. Dies ist aber durchaus ein Vorteil, da man erst nach all der Zeit allmählich versteht, was eine objektorientierte Sprache ausmacht und welche Sprachmittel man wirklich benötigt. Sehen Sie sich zum Vergleich einmal an, was gerade im Umfeld der Java Virtual Machine (JVM) geschieht. Dort entstehen eine Menge neuer Sprachen, die alle versuchen, die Unzulänglichkeiten und veralteten Paradigmen von Java zum Teil zu umgehen oder zu ersetzen, um eine moderne Sprache zu schaffen. Hier hat Apple mit

Objective-C 2.0 durchaus ein glückliches Händchen gehabt und behutsam neue Aspekte in die Sprache eingeführt. Dabei ist es jedoch gelungen, die Sprache weiterhin so wirken zu lassen, als wäre sie aus einem Guss.

Die Konzepte, die Objective-C umsetzt, wirken von vorne bis hinten durchdacht. Sehr selten nur ist man gezwungen, einen Umweg zu gehen, um schließlich zum Ziel zu kommen. So ist zum Beispiel die Verwendung von Methoden mit benannten Parametern etwas, was man aus den meisten objektorientierten Sprachen nicht kennt. Man erkennt nach kurzer Einarbeitungszeit sofort, was eine Methode macht, ohne zuerst die Parameter nachschlagen und zuordnen zu müssen.

Schließlich bleibt anzumerken, dass man mit Objective-C in Kombination mit dem Cocoa-Framework in der Lage ist, sowohl schnell und einfach zu einem ersten Ergebnis zu kommen, als auch bis ins letzte Detail Einfluss zu nehmen. Dies gilt sowohl für den Mac als auch für iOS, viel mehr, als wir das von anderen Sprachen und anderen Systemen kennen. Wenn Sie einmal mit den Werkzeugen, die Apple bereitstellt, gearbeitet und das Cocoa-Framework ein bisschen besser kennen gelernt haben, werden Sie feststellen, dass es sich einfach gut anfühlt, damit zu arbeiten. Denn dies sind die Tools und die Sprachmittel und Bibliotheken, die auch Apple verwendet. Sie halten sie aber nicht in Cupertino hinter verschlossener Tür, sondern teilen, zumindest das meiste, mit uns Entwicklern. Was im Gegenzug den Effekt hat, dass Mac-Programmierer sich meist deutlich stärker mit ihren Programmen beschäftigen und identifizieren und dadurch immer noch ein wenig mehr polieren als ihre Kollegen auf anderen Plattformen.

Aber lassen Sie sich selbst in die Welt von Objective-C und der Programmierung für den Mac und iOS führen. Wir sind überzeugt davon, dass Sie es lieben und sich auch schnell als Mac-Entwickler fühlen werden. Wir hoffen, dass Sie beim Lesen dieses Buches genauso viel Spaß haben wie wir beim Schreiben und wünschen Ihnen viel Erfolg bei Ihrem nächsten oder vielleicht auch ersten Programm, das Sie in Objective-C schreiben.

An wen richtet sich das Buch?

Viele Objective-C-Bücher sind in der ersten Linie auch Bücher über die Sprache C. Dies wollen wir nicht. Wir wollen die Programmiersprache Objective-C in der Version 2.0 in den Mittelpunkt stellen. Dies hat allerdings zur Folge, dass wir davon ausgehen, dass Sie bereits mit der Programmiersprache C oder auch C++ programmiert haben. Ist dies nicht der Fall, können Sie dieses Buch natürlich trotzdem lesen, da die Beispiele so einfach wie möglich gehalten wurden.

Viele andere Teile gehen jedoch auch sehr in die Tiefe, wodurch ein gutes Verständnis der Sprache C vonnöten ist. In diesem Fall würden wir empfehlen, sich

vorweg ein wenig mit der Programmierung in C auseinanderzusetzen und vor allem zu verstehen, wie nah man sich mit C an der Programmierung der Hardware befindet und welche Möglichkeiten dieses dem Programmierer durch die etwas »einfachere« Typprüfung bietet. Im Grunde hat die Programmiersprache C eigentlich keine echte Typprüfung, sondern nur eine Feststellung, wie viel Platz eine Variable im Speicher einnimmt. Für alle Interessierten und insbesondere Programmierer, die wenig bis keine Kenntnisse in C besitzen, gibt es aus diesem Grund im Anhang B einen kurzen C-Kurs, der Ihnen einen Einblick in die C-Programmierung gewährt.

Erfahrung mit der objektorientierten Programmierung müssen Sie jedoch nicht besitzen. Wir erklären das Konzept der objektorientierten Programmierung in diesem Buch von Grund auf. Denn die meisten objektorientierten Programmiersprachen sind gar keine richtigen reinen objektorientierten Sprachen, wodurch es sehr schwer ist, den Kern von objektorientierter Programmierung (OOP) zu verstehen. Dies trifft im Grunde auch auf Objective-C zu. Allerdings ist der Grad zwischen strukturierter prozeduraler Programmierung und objektorientierter Programmierung in Objective-C sehr gut gelungen. Also auch für Programmierer, die bereits eine objektorientierte Programmiersprache verwendet haben, macht es Sinn, zu kontrollieren, ob sie das OOP-Konzept auf die gleiche Weise bereits verstanden haben, wie es in Objective-C umgesetzt ist.

Aufbau des Buches

Dieses Buch teilt sich in zwei Teile. Der erste Teil beschäftigt sich mit der Programmiersprache Objective-C 2.0 und der objektorientierten Programmierung. Eine Einführung in die Entwicklungswerkzeuge von Mac OS X haben wir mit Absicht nicht in den ersten Teil des Buches aufgenommen, diese werden in Kapitel 8 im zweiten Teil vorgestellt. Lesen Sie sich erst einmal in die Programmiersprache Objective-C ein. Wenn Sie dann das Gefühl haben, genug verstanden zu haben, ist zu empfehlen, sich zunächst einmal mit den Entwicklungsumgebungen auseinanderzusetzen. Den Zeitpunkt sollten jedoch Sie bestimmen. Zum weiteren Verständnis gibt es für jedes Kapitel einen Abschnitt mit Übungsaufgaben. Die entsprechenden Lösungshinweise finden Sie gesammelt in Anhang A.

Wir beschäftigen uns in diesem Buch ausschließlich mit der Version 2.0 von Objective-C. Diese Version steht erst ab Mac OS X 10.5 zur Verfügung. Daher sind auch alle Beispiele für eine Mac-OS-X-Version ab 10.5 ausgelegt. Die Blöcke aus Kapitel 7 sind erst ab Mac OS X Version 10.6 verfügbar. Für die Leser, die noch unter Mac OS 10.4 programmieren, haben wir die Features der Sprache und des Cocoa-Frameworks gekennzeichnet, die unter 10.4 oder Objective-C 1 nicht zur Verfügung stehen.

Dies sind insbesondere:

- Properties (Kapitel 4)
- Optionale Methoden in Protokollen (Abschnitt 5.4)
- Blöcke (Kapitel 7) erst ab Mac OS X 10.6
- Das Entwurfsmuster der schnellen Iteration (Abschnitt 14.2)
- Garbage Collection (Abschnitt 9.4)

Die restlichen Teile sind, soweit nicht anders vermerkt, auch für den Einsatz unter Mac OS X 10.4 geeignet. An einigen Stellen kann es jedoch sein, dass eine benutzte Methode erst in 10.5 vorhanden ist. In diesen Fällen können Sie jedoch dem Text entnehmen, was die Aufgabe dieser Methode ist. Die Dokumentation von Apple bietet in diesen seltenen Fällen einen guten Anlaufpunkt, um eine äquivalente Methode für ältere Cocoa-Versionen zu finden.

■ Kapitel 1: Objekte und Klassen

Wir steigen aus diesem Grund im ersten Teil des Buches mit Kapitel 1 sofort bei der objektorientierten Programmierung und ihrer Realisierung in Objective-C mit Objekten und Klassen ein. Es gibt vorweg auch keine Einleitung in den C-Teil von Objective-C, da wir davon ausgehen, dass dieser bereits bekannt ist. Für alle Interessierten gibt es dennoch eine kurze Einführung in die Programmiersprache C in Anhang B.

■ Kapitel 2: Wie werden Nachrichten verarbeitet?

Um ein bisschen besser zu verstehen, wie Objective-C funktioniert, werden wir uns in Kapitel 2 anschauen, wie die objektorientierten Elemente von Objective-C in die Sprache C abgebildet werden können.

In den darauffolgenden Kapiteln 3 bis 7 des ersten Teils wollen wir uns einige zusätzliche Features von Objective-C ansehen, die zum Teil erst mit Objective-C 2.0 hinzugekommen sind. Bei diesen Features handelt es sich um einige sehr nützliche Details, die einem Programmierer in vielen Situationen das Leben stark erleichtern. Ein sehr nützliches Feature ist zum Beispiel, dass wir sogar bereits in einem Framework kompilierte Klassen durch Kategorien erweitern können.

■ Kapitel 3: Kategorien

Wie Kategorien eingesetzt werden können und was dabei zu beachten ist, untersuchen wir in Kapitel 3.

■ Kapitel 4: Properties

In Kapitel 4 befassen wir uns mit Properties, die eine bestimmte Art von Methoden, die sehr häufig benötigt werden, automatisch generieren. Jeder, der bereits eine objektorientierte Programmiersprache verwendet hat, die keine Properties besitzt, wird sie lieben. Sie erleichtern uns eigentlich in fast jeder Klasse das Leben.

■ Kapitel 5: Protokolle

Ebenso interessant sind die Protokolle aus Kapitel 5, die den Interfaces in anderen Sprachen sehr ähneln, allerdings einige interessante Eigenschaften mitbringen, die so in anderen Sprachen eher selten existieren.

■ Kapitel 6: Ausnahmebehandlung

In Kapitel 6 schließt dann der erste Teil des Buches mit den Ausnahmebehandlungen ab. Ausnahmen sind in modernen Programmiersprachen mittlerweile Pflicht geworden, um in einem Fehlerfall vom normalen Programmfluss abzuweichen und den Programmcode für den fehlerfreien Programmablauf von der Fehlerbehandlung zu entkoppeln. Hierdurch müssen nicht mehr ständig die Rückgabewerte von Funktionen für die Fehlerbehandlung verwendet werden und die Übersichtlichkeit von Programmen steigt enorm.

■ Kapitel 7: Blöcke

Mit Kapitel 7 und den Blöcken schließt dann der erste Teil des Buches ab. Blöcke sind eine Art von anonymen Prozeduren. Sie bieten im Vergleich zu normalen Funktionen jedoch einige Vorteile, die zu sehr interessanten Möglichkeiten und neuen Programmiervarianten führen.

Der zweite Teil des Buches

Im zweiten Teil des Buches wollen wir uns dann ansehen, wie die Programmiersprache Objective-C im Mac-Umfeld, also unter Mac OS X oder iOS, eingesetzt werden kann. Es handelt sich um einen Überblick über die dort vorhandenen Entwicklungswerkzeuge und mitgelieferten Frameworks. Dies ist beim Erlernen moderner Programmiersprachen meist der wichtigste Teil, da die mitgelieferten Frameworks immer größer werden. Das Lernen der eigentlichen Programmiersprache ist im Vergleich zum Einleben in ein neues Framework verhältnismäßig leicht. Dies gilt auch für Objective-C und den Umfang des Cocoa-Frameworks, dem wichtigsten Framework unter Mac OS X, das etwa 300 Klassen umfasst. Allerdings macht es immer wieder Spaß, mit dem Cocoa-Framework zu arbeiten, da die Konzepte sehr klar sind und alles wie aus einem Guss wirkt. Aus diesem Grund wollen wir uns in diesem Buch auch auf die Teile der Bibliothek konzentrieren, die infolge der Anwendungen mit grafischer Benutzeroberfläche selten im Vordergrund stehen, jedoch beim täglichen Gebrauch von Objective-C ständig benötigt werden. Ein weiterer Grund für diese Ausrichtung ist, dass es mit dem Buch *Cocoa – Programmierung für Mac OS X* von Aaron Hillegass bereits ein sehr gutes Buch gibt, das nur jedem empfohlen werden kann. Hillegass konzentriert sich sehr stark auf die Entwicklung von grafischen Oberflächen. In diesem Buch sollen deshalb jene Teile des Frameworks betrachtet werden, die im Hintergrund hinter der grafischen Oberfläche stehen, jedoch fundamental wichtig für ein gutes Programm sind. Der zweite Teil soll deshalb auch eher als eine Art Nachschlagewerk angesehen werden und nicht als ein »Roman«, der von vorne bis hinten

sequenziell durchgelesen wird. Lesen Sie lieber die Teile, die Sie besonders interessieren und die Sie in Ihrer Situation gerade brauchen. Wir wollen uns jedoch einmal genau anschauen, welche Inhalte in den einzelnen Kapiteln vermittelt werden sollen:

■ **Kapitel 8: Die Cocoa-Umgebung**

In diesem Kapitel wollen wir einen Überblick über die Cocoa-Entwicklungsumgebung unter Mac OS X und iOS geben. Dazu stellen wir die dort zur Verfügung gestellten Entwicklungstools vor und zeigen, wie man mit ihnen sowohl für Mac OS X als auch für iOS Software entwickeln kann.

■ **Kapitel 9: Memory Management**

Da es sich bei Objective-C im Prinzip um C handelt, spielt das Memory Management eine sehr große Rolle. Mit dem Cocoa-Framework werden für den Programmierer allerdings zwei Verfahren mitgeliefert, die das Memory Management stark vereinfachen: das Reference Counting und das Garbage Collecting. Diese beiden Verfahren wollen wir in diesem Kapitel genau erklären.

■ **Kapitel 10: Grundlegende Klassen**

Das Cocoa-Framework bietet eine große Auswahl an Klassen, die für die meisten Anwendungsfälle passende Lösungsansätze und Konzepte bereithalten. In einem normalen Programm werden Sie daher nicht die volle Spannweite dieser Bibliothek nutzen. Es gibt jedoch einige wenige Klassen, die so zentral sind, dass sie die Grundbausteine jedes Programms bilden. Mit diesen grundlegenden Klassen wollen wir uns in diesem Kapitel detailliert beschäftigen.

■ **Kapitel 11: Collections**

Collections sind Sammlungen von Objekten. Jede moderne Programmiersprache bringt in ihrer Bibliothek Klassen mit, die solche Sammlungen auf unterschiedliche Art ermöglichen. Objective-C macht hier mit dem Cocoa-Framework keine Ausnahme, geht aber durch die Trennung in veränderliche und unveränderliche Sammlungen noch einen Schritt weiter. In diesem Kapitel wollen wir uns daher genauer mit der Collections-API des Cocoa-Frameworks beschäftigen.

■ **Kapitel 12: Eingabe und Ausgabe**

Kaum ein Programm kommt heute ohne die Ein- und Ausgabe von Daten in Dateien oder über Netzwerkverbindungen aus. Dieses Kapitel beschäftigt sich daher mit den reichlichen Möglichkeiten der Ein- und Ausgabe durch das Cocoa-Framework und beleuchtet ausführlich sowohl die Dateiebene als auch die Kommunikation über Netzwerke.

■ **Kapitel 13: Introspektion**

Da es sich bei Objective-C um eine eher dynamische objektorientierte Programmiersprache handelt, gibt es mit der Objective-C-Runtime-API einige

beeindruckende Möglichkeiten, sich zur Laufzeit über das Programm zu informieren und eventuell sogar Änderungen am Programm durchzuführen. Wie weit diese Auskunft- und Änderungsmöglichkeiten reichen, soll in diesem Kapitel betrachtet werden.

■ Kapitel 14: Design Patterns für Objective-C

Design Patterns spielen beim Entwurf von Software heutzutage eine bedeutende Rolle. Sie fassen bewährte Implementierungen als eine Art Best Practice zusammen. In Kapitel 14 wollen wir uns vier Entwurfsmuster für die praktische Arbeit mit dem Cocoa-Framework ansehen.

■ Kapitel 15: Threading

Auf modernen Systemen wird es immer wichtiger, dass mehrere Programmteile parallel abgearbeitet werden können. Welche Möglichkeiten das Cocoa-Framework hierzu bietet und welche Risiken und Schwierigkeiten damit verbunden sind, wollen wir uns in diesem Kapitel ansehen.

■ Kapitel 16: Bundles

Anwendungen, die auf dem Mac oder unter iOS laufen, sind nicht nur eine Datei. Vielmehr sind Anwendungen in Wirklichkeit Verzeichnisse, die neben Programmcode auch Ressourcen wie zum Beispiel Bilder oder Texte enthalten können. In diesem Kapitel sehen wir uns an, wie wir diese Eigenschaft benutzen können, um ein Programm in verschiedenen Sprachen auszuliefern, oder auch, um ein Programm mit PlugIns nachträglich zu erweitern.

Beispiele und verwendete Werkzeuge

In den Kapiteln dieses Buches findet sich eine Vielzahl an Beispielen, um die Sachverhalte der einzelnen Kapitel zu verdeutlichen. Diese Beispiele sind mit Xcode 3.24 unter Mac OS X 10.6.6 entstanden und geprüft worden. Die Beispiele für iOS sind sowohl im Simulator als auch auf dem iPhone mit dem Betriebssystem iOS 4.2.1 getestet worden.

Die Werkzeuge zum Entwickeln von Software sind kostenloser Bestandteil des Betriebssystems Mac OS X. Die beiden wichtigsten Anwendungen sind Xcode und Interface Builder. Wie diese Tools zum Entwickeln von Software eingesetzt werden können, wollen wir uns in Kapitel 8 ansehen. Wer jedoch bereits mit einer modernen Programmiersprache und den dazugehörigen Entwicklungswerkzeugen gearbeitet hat, wird hier nicht sehr viele Überraschungen erleben und sich schnell einfinden.

Unter der folgenden Webseite können Sie auf Basis der Listingnummer die meisten Programmbeispiele herunterladen:

www.mitp.de/9132

In den meisten Fällen handelt es sich nur um Ausschnitte aus den vollständigen Programmen. Die Ausschnitte sind häufig auf den Kern reduziert. Viele der Beispiele müssen erst in den Kontext einer `main`-Funktion eingefügt werden, um ausgeführt werden zu können. Bei Beispielen, die auf mehrere Dateien aufgeteilt sind, wurde in den meisten Fällen auf die Angabe der `#import`-Anweisungen verzichtet. Es kann jedoch davon ausgegangen werden, dass Klassen immer in zwei Dateien abgelegt sind. Für eine Klasse mit dem Namen `MyClass` wäre es die Datei `MyClass.h` für das Klasseninterface und `MyClass.m` für die Implementierung.

Wir verwenden bereits zu Beginn des Buches das Cocoa-Framework, um darauf unsere Beispiele aufzubauen. Hierbei verwenden wir natürlich auch Klassen aus diesem Framework und nicht nur eigene Klassen. Im Normalfall versuchen wir, diese Klassen dann auch zu beschreiben. Jedoch gibt es für eigentlich jede Klasse eine sehr gute Klassendokumentation, in der alle unterstützten Methoden beschrieben werden.

Wenn nichts Genaueres dazu geschrieben wird, handelt es sich bei den Testanwendungen, in denen die Beispiele entstanden sind, um Cocoa-Konsolenanwendungen. Wie man diese erstellt und verwendet, wird in Kapitel 8 beschrieben.

Um die Programme so einfach wie möglich zu halten, wurde beim Memory Management auf das Garbage Collecting zurückgegriffen (siehe Kapitel 9). Unter iOS steht allerdings kein Garbage Collecting zur Verfügung. Aus diesem Grund sind die Anwendungen für iOS auf Basis von Reference Counting implementiert worden.

Für viele Beispiele sind Ausgaben als Bilder in den Kapiteln abgedruckt. Diese Beispiele sind Ausgaben der Programme, die als Bildschirmfotos aus der Anwendung *Konsole* von Mac OS X gewonnen wurden. Allerdings handelt es sich im Normalfall nur um einen Ausschnitt des gesamten Fensters.

Typografische Konventionen

Um Ihnen das Einfinden in die Texte dieses Buches zu erleichtern, haben wir einige typografische Konventionen festgelegt. Diese Konventionen wollen wir Ihnen nun kurz vorstellen.

Beispiele sind immer als ein hervorgehobener Block in einer Schrift mit festem Zeichenabstand wie folgt gekennzeichnet:

```
-(void)setBalance:(float)theBalance {
    accountBalance = theBalance;
}
```

Listing 1: Darstellung von Beispiel-Quellcode

Vollständige Methoden, Klassen oder längere Beispiele besitzen wie in diesem Fall (Listing 1) eigentlich immer eine Listingunterschrift mit Angabe einer eindeutigen Nummer.

Wird in den Beschreibungen zu den Beispielen auf Methoden oder Variablennamen Bezug genommen, so werden diese ebenfalls in einer Schriftart mit festem Zeichenabstand abgedruckt, wie zum Beispiel beim Methodennamen `setBalance`:. Ähnliches gilt für die Angabe von Dateinamen oder Internetadressen.

Namen von Bildschirmsteuerelementen oder Menübefehle werden in besonders dafür ausgezeichneter Schriftart abgedruckt (Beispiel: »File/New Project...«). Ebenfalls kursiv gedruckt sind Verweise auf externe Dokumente oder Bücher.

Unseres Erachtens nach wichtige Begriffe heben wir bei der ersten Verwendung fett hervor, zum Beispiel tritt in Kapitel 1 irgendwann in fett der Begriff **Polymorphie** auf.

Hinweis

Hinweise, Tipps oder Dinge, die Sie beachten sollten und die wir als besonders wichtig erachten, haben wir in eine solche graue hervorgehobene Box geschrieben, damit sie Ihnen besonders ins Auge fallen.

Historische Einordnung

Wenn man von Objective-C hört, denkt man in den heutigen Tagen sofort an Apple und Mac OS X. Die Programmiersprache Objective-C stammt jedoch bereits aus den frühen 1980ern, also aus einer Zeit etwa 20 Jahre vor dem Erscheinungsdatum von Mac OS X. Objective-C stellt eine Erweiterung der Programmiersprache C dar und wurde von Brad Cox und Tom Love in ihrem Unternehmen StepStone entwickelt. Objective-C ist damit etwa gleich alt wie die 1983 erschienene, sehr viel bekanntere objektorientierte Programmiersprache C++, die ebenfalls auf C basiert und von Bjarne Stroustrup bei AT&T entwickelt wurde.

Zu diesem Zeitpunkt basierten fast alle Software-Entwicklungen auf der strukturierten Programmierung. Durch die strukturierte Programmierung erhoffte man sich damals eine Möglichkeit, die immer größer werdenden Programme in kleinere handhabbarere Teile zerlegen zu können. Jedoch führten die immer größer werdenden Programme zu einem immensen Anstieg der notwendigen Prozeduren und der dazugehörigen Kontrollstrukturen, die entschieden, welche Prozeduren auszuführen waren. An eine effiziente Form der Wiederverwendung von Code war damit nicht zu denken.

Aus diesem Grund entstand die Idee der objektorientierten Programmierung. Die objektorientierte Programmierung versprach, das Heilmittel für die Zerlegung

großer Projekte zu sein, und vereinfachte die Codewiederverwendung erheblich. Als eigentlich erste objektorientierte Programmiersprache wurde in den 1970er Jahren die Programmiersprache SmallTalk am Xerox-PARC-Forschungszentrum von Alan Kay, Dan Ingalls und Adele Goldberg entwickelt.

Das Konzept von SmallTalk war damals revolutionär. So soll Alan Kay 1970 zu Beginn seiner Zeit bei Xerox auf die Frage »Was wird ihre wichtigste Errungenschaft hier sein?« geantwortet haben: »Der Personal Computer«, und er skizzierte auf seinem Notizbuch, »Ein Rechner dieser Größe, mit einem Bildschirm, einer Tastatur und genug Power, um ihre Post zu speichern und ihre Musik, Bilder und Bücher.« Mit dieser Vorstellung stieß Kay bei Xerox auf Unverständnis, sie ließen ihn jedoch an seinem Projekt arbeiten.

Durch dieses Unverständnis wurde das Potenzial der Sprache und der damit entstandenen Umgebung von Xerox völlig unterschätzt, wodurch es nie zu einem echten Durchbruch kam, obwohl einige sehr interessante Ansätze zusammen mit SmallTalk entstanden sind. So wurde auf Basis von SmallTalk die erste interaktive grafische Benutzeroberfläche mit Mausbedienung und Menüsteuerung entwickelt (siehe Abbildung 1).

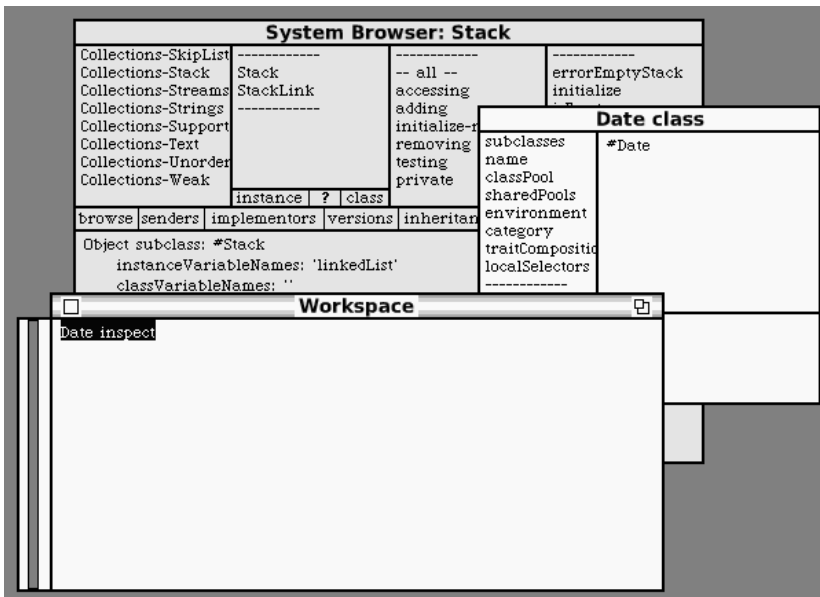


Abb. 1: Grafische Benutzeroberfläche in SmallTalk-80

Zu diesem Zeitpunkt kam auch das Unternehmen Apple mit Steve Jobs durch einen Besuch bei Xerox das erste Mal mit SmallTalk und den dort entstehenden Entwicklungen in Kontakt. Dieser Besuch war für Steve Jobs ein Wendepunkt, als er vernetzte Computer sah, die mit grafischer Benutzeroberfläche untereinander

E-Mails austauschten und mit einer objektorientierten Programmiersprache programmiert waren. Steve Jobs sagte viele Jahre später über seinen Besuch bei Xerox: *»Ich war total geblendet von dem ersten Ding, das sie mir zeigten: Die grafische Benutzeroberfläche. Ich dachte, das ist das beste Ding, was ich je in meinem Leben gesehen habe. Es hatte noch viele Schwächen. Was wir sahen, war unvollständig. Sie hatten eine ganze Reihe Sachen falsch gemacht. Aber zu der Zeit wussten wir das nicht. Aber dennoch: Sie hatten den Keim der Idee geschaffen, und sie hatten es sehr gut gemacht. Und innerhalb von zehn Minuten war mir klar, dass eines Tages alle Computer so arbeiten würden.«¹*

Aus diesem Besuch entstand für Apple die Vision eines Rechners mit grafischer Benutzeroberfläche nach dem Vorbild von Xerox, woraus bis zum Jahr 1984 der erste Macintosh entstand. In Abbildung 2 sehen Sie den Apple Macintosh SE/30, der 1989 eingeführt wurde und dem ersten Macintosh äußerlich sehr ähnlich war. Was Steve Jobs und Apple allerdings unterschätzten, war die objektorientierte Programmierung auf Basis der Programmiersprache SmallTalk. Die Entwicklung von objektorientierten Programmiersprachen ging aus diesem Grund viele Jahre an Apple vorüber. Steve Jobs sagte hierzu in demselben Interview wie oben über seinen Besuch bei Xerox: *» ... und sie zeigten mir wirklich drei Dinge, aber von dem ersten war ich so geblendet, dass ich die anderen beiden gar nicht richtig wahrgenommen habe. Eines der Dinge, die sie mir zeigten, war die objektorientierte Programmierung. Das haben sie mir gezeigt! Aber ich habe es nicht einmal wahrgenommen ...«*



Abb. 2: Apple Macintosh SE/30 von 1989

¹ Aus dem Film *Triumph of the Nerds*, <http://www.pbs.org/nerds/>

Der größte Nachteil an der Programmiersprache SmallTalk war damals ihre Geschwindigkeit. Zur Ausführung von SmallTalk diente eine virtuelle Maschine, die später unter anderem als Vorbild für die virtuelle Maschine der Programmiersprache Java diente. Da Rechner zum damaligen Zeitpunkt jedoch noch sehr langsam waren, war durch die zusätzliche virtuelle Maschine die Programmiersprache SmallTalk viel zu langsam, um den Durchbruch in den freien Markt zu schaffen.

Viele andere Programme basierten damals auf der Programmiersprache C. Diese war durch ihre Nähe zur Hardware extrem effizient zu programmieren, wodurch auch auf langsamer Hardware effiziente Programme entstehen konnten. Die Vision hinter den Programmiersprachen C++ von Bjarne Stroustrup und Objective-C von Brad Cox und Tom Love war damals, die Ideen der Programmierung von SmallTalk auf die Programmiersprache C mit ihrer effizienten Ausführung zu übertragen. Auf welche Art und Weise diese Kombination erzielt wurde, unterschied sich jedoch stark. Bei der Programmiersprache C++ stand die Effizienz im Vordergrund, was zu einer eher statischen objektorientierten Erweiterung führte. Diese Effizienz stellte jedoch den großen Vorteil von C++ dar. Bei Objective-C standen eher die dynamischen Aspekte von SmallTalk im Vordergrund. So wundert es nicht, dass sich SmallTalk und Objective-C, was die Syntax angeht, sehr viel stärker ähnelten. Dies hatte allerdings zur Folge, dass Objective-C ebenfalls eher ein Nischendasein fristete. Die erste echte Beschreibung der Programmiersprache in seiner originalen Form veröffentlichten Cox und Love in dem Buch *Object-Oriented Programming, An Evolutionary Approach* im Jahr 1986.

Im Jahr 1988 gründete Steve Jobs, nachdem er Apple verlassen hatte, das Unternehmen NeXT. Mit NeXT hoffte Steve Jobs, seine Visionen nach seinen eigenen Vorstellungen umsetzen zu können. Hierzu entwickelte er eigene Rechner zusammen mit dem objektorientierten Betriebssystem NeXTSTEP. Für die Entwicklung lizenzierte NeXT die Programmiersprache Objective-C von StepStone und entwickelte seinen eigenen Compiler. Auf Basis von Objective-C entstanden dann die objektorientierten NeXTSTEP-Bibliotheken für das eigene Betriebssystem.

Nach dem Kauf von NeXT durch Apple im Jahr 1996 verwendete Apple unter der neuen Führung von Steve Jobs NeXTSTEP als Grundlage für die Entwicklung ihres neuen Betriebssystems Mac OS X. Hierzu gehörte auch die Programmiersprache Objective-C und die dazugehörigen Entwicklertools: **Project Builder**, aus dem später die Entwicklungsumgebung **Xcode** entstehen sollte, und der **Interface Builder**. Viele der Teile von NeXTSTEP finden sich heutzutage immer noch im Cocoa-Framework, der zentralen Bibliothek von Mac OS X, wieder. Mac OS X ist mit dem Cocoa-Framework damit die größte und ernst zu nehmendste Entwicklungsplattform für die Programmiersprache Objective-C.

Als Alternative zum Framework NeXTSTEP gibt es das auf dem OpenStep-Standard basierende Framework GNUstep. Auch dieses Framework greift auf die in

der GNU Compiler Collection (gcc) integrierte Version der Programmiersprache Objective-C zurück.

Mit Mac OS X 10.5 entwickelte Apple dann eine an die eigenen Bedürfnisse angepasste Version von Objective-C. Diese veröffentlichte Apple im Jahr 2006 auf der *Worldwide Developers Conference* unter dem Namen Objective-C 2.0 als eine Revision der Programmiersprache Objective-C zur zusätzlichen Unterstützung von Garbage Collecting als modernes Memory-Management-Verfahren und mit einigen Syntaxerweiterungen sowie einer verbesserten Performance. Hierfür stellte Apple einen eigenen Compiler zur Verfügung. Bis jetzt ist nicht bekannt, wann und ob die Sprache Objective-C 2.0 in die GNU Compiler Collection für andere Systeme als Mac OS X Einzug halten wird.

Seit dem 11. Juli 2008 stehen Objective-C 2.0 und eine spezielle Version des Cocoa-Frameworks auch als Programmierumgebung für das von Apple entwickelte iPhone der Öffentlichkeit zur Verfügung. Apple bietet mit dem iPhone SDK einen revolutionären Ansatz für die Programmierung von tragbaren Geräten. Zusammen mit der Möglichkeit, eigene Programme über den AppStore auf eine sehr einfache Art und Weise verkaufen zu können, hat Apple einen extremen Boom für die Programmierung auf einem Handy ausgelöst und damit der Programmiersprache Objective-C zu einem späten Siegeszug verholfen.

Mit der Vorstellung des iPads am 27. Januar 2010 ist ein weiteres Gerät für die Programmierung mit Objective-C und dem Cocoa-Framework hinzugekommen. In diesem Zuge wurde das Betriebssystem iPhone OS in iOS umbenannt und steht seitdem beiden Plattformen zur Verfügung. Auch das neue AppleTV der zweiten Generation basiert auf dem neuen iOS. Hiermit zeigt Apple, dass der Weg des neuen Betriebssystems und der Programmierung mit Objective-C erst begonnen hat und sie mit iOS und Objective-C noch viel vorhaben.

Aber auch auf dem Mac selbst hält das Konzept eines zentralen Marketplace Einzug. Seit dem 6.1.2011 steht auch unter Mac OS X ab 10.6.6 ein Mac AppStore zur Verfügung. Über diesen können Entwickler ihre Programme allen Mac-Benutzern zur Verfügung stellen. Dieser Schritt vereinheitlicht die Entwicklungsstrategie von Apple für alle Gerätefamilien und wird insgesamt sicherlich der Programmiersprache Objective-C sehr förderlich sein.

Objective-C als Erweiterung zu ANSI-C

Wie zuvor bereits erwähnt, handelt es sich bei der Programmiersprache Objective-C um eine Erweiterung des ANSI-Standards für die Programmiersprache C. Hierdurch sind alle für ANSI-C geschriebenen Programmteile auch in Objective-C einsetzbar, das heißt, Sprachkonstrukte wie Schleifen, Abfragen und die Definition von Funktionen oder Datentypen bleiben identisch. Viele der zusätzlich hinzuge-

fügten Sprachelemente dienen dazu, die objektorientierten Konzepte in C nachzupflegen. Es sind erstaunlich wenige zusätzliche Konstrukte notwendig, um eine objektorientierte Programmierung zu ermöglichen. Sie werden merken, dass die Programmierung in Objective-C für einen C-Programmierer sehr einfach zu erlernen ist. Mit der objektorientierten Programmierung und den hierzu hinzugefügten Sprachelementen werden wir uns in Kapitel 1 ausführlich beschäftigen.

Als Compiler verwendet man unter Mac OS X die GNU Compiler Collection (kurz gcc). In diese wurde Objective-C 2.0 von Apple integriert. Allerdings existiert die GNU Compiler Collection mit Objective-C-2.0-Unterstützung nur für Mac OS X. Für andere Systeme steht nur die einfache Version von Objective-C zur Verfügung. Aus diesem Grund müssen Sie sich auch nicht wundern, wenn ein Programm, das unter Mac OS X kompiliert wurde, sich unter Linux nicht übersetzen lässt.

Womit startet ein Objective-C-Programm?

Da jedes C-Programm auch ein Objective-C-Programm ist, beginnt die Ausführung eines Objective-C-Programms ebenfalls mit der `main`-Funktion. Diese Methode muss irgendwo in einer der Quelldateien, die übersetzt werden, zur Verfügung gestellt werden. Damit der Compiler erkennen kann, dass es sich um Objective-C-Quellcode handelt und nicht um einfaches ANSI-C, ist die Endung für die Quelldateien nicht `.c`, sondern `.m`. In Objective-C gibt es jedoch auch zusätzlich die Möglichkeit, Definitionen in andere Dateien auszulagern. Diese Dateien enden äquivalent zu C mit `.h`. In einer der `.m`-Dateien eines Objective-C-Programms muss also eine `main`-Funktion der folgenden Art definiert sein:

```
int main (int argc, const char * argv[]) {
    // Objective-C Code
    return 0;
}
```

Diese Methode wird dann mit dem Starten der Anwendung ausgeführt und damit der darin enthaltene und übersetzte Quelltext ausgeführt.

Die Präprozessordirektive »#import«

Es gibt allerdings auch noch einige sehr nützliche Erweiterungen in Objective-C, die das Programmieren in Objective-C an vielen Stellen vereinfachen. Das wichtigste Sprachelement in diesem Zusammenhang ist die Präprozessordirektive `#import`. Mit Hilfe von `#import` kann der Quelltext in eine andere Datei hinzugefügt werden:

```
#import "someFile.h"
```

Die Verwendung von `#import` verhält sich sehr ähnlich zu der Verwendung von `#include` aus der Programmiersprache C:

```
#include "someFile.h"
```

Jedoch wird beim mehrfachen Auftreten von `#import` eine Datei nicht mehrfach hinzugefügt. Beim Befehl `#include` musste dies in C manuell abgebildet werden, wodurch der Aufbau einer `.h`-Datei für C immer wie folgt aussieht:

```
#ifndef _SOMEFILE_H_
#define _SOMEFILE_H_
    // Eigentlicher Quelltext
#endif /* _SOMEFILE_H_ */
```

Bei Verwendung von `#import` anstelle von `#include` kann dieses Konstrukt vollständig entfallen und es ergibt sich exakt das gleiche Verhalten. Hiermit ist eine beliebte Fehlerquelle ausgeschaltet und die Lesbarkeit der `.h`-Dateien erhöht sich, da nur noch der relevante Quellcode erhalten bleiben muss.

Darstellung von booleschen Werten

Ein weiterer Punkt, der die Lesbarkeit von Objective-C-Programmen erhöht, ist die Einführung des Datentyps `BOOL` als Standarddatentyp zur Darstellung von booleschen Werten. Ein solcher Datentyp existiert in ANSI-C nicht. Im C99-Standard oder C++ sind ebenfalls Datentypen zur Darstellung von booleschen Werten hinzugefügt worden. Bei C99 ist es der Datentyp `_Bool` und in C++ `bool`. Allerdings unterscheidet sich die Realisierung in Objective-C ein wenig von den anderen untereinander sehr ähnlichen Realisierungen.

Beim Datentyp `BOOL` in Objective-C handelt es sich eigentlich nur um eine Typdefinition auf `signed char`:

```
typedef signed char BOOL;
```

Außerdem gibt es zwei Konstanten, die als Wahrheitswerte »Wahr« (`YES`) und »Falsch« (`NO`) dienen:

```
#define YES (BOOL)1
#define NO (BOOL)0
```

Diese beiden Konstanten können einer Variablen vom Typ `BOOL` zugewiesen und in booleschen Operationen verwendet werden:

```
BOOL var1 = YES;
BOOL var2 = NO;
BOOL var3 = (YES || var2) & (var1 && NO);
```

Da in C implizit davon ausgegangen wird, dass 0 den Wert »Falsch« besitzt und jeder andere Wert einem »Wahr« entspricht, funktioniert diese Definition zum Beispiel in einer `if`-Abfrage wie erwartet:

```
if (var3) { printf("Ja"); } else { printf("Nein"); }
```

Diese Form der Definition hat allerdings zur Folge, dass einem `BOOL` jeder Wert zugewiesen werden kann, der auch einer Variablen vom Typ `signed char` zugewiesen werden kann.

```
BOOL var3 = 5;
if (var3) { printf("Ja"); } else { printf("Nein"); }
```

Aus diesem Grund muss das Vergleichen von booleschen Werten mit Vorsicht genossen werden. So führt die folgende Konstruktion nicht immer zum erwarteten Ergebnis, obwohl sie aus boolescher Sicht äquivalent zur vorherigen `if`-Abfrage sein müsste:

```
if (var3 == YES) { printf("Ja"); } else { printf("Nein"); }
```

So würde die zweite Variante der Variablen `var3` mit dem obigen `if` zu der Ausgabe "Ja" führen, wohingegen die zweite `if`-Abfrage zu der Ausgabe "Nein" führt, obwohl man erwarten würde, dass beide Aussagen äquivalent sind.

Die beiden Datentypen `_Bool` und `bool` sind im Gegensatz zu `BOOL` allerdings fest in den Compiler eingebaut und funktionieren auch beim Objective-C-Compiler von Apple. Auch für diese beiden Datentypen gibt es Konstanten für die beiden Wahrheitswerte: `true` und `false`.

Dadurch, dass die beiden Datentypen jedoch fest in den Compiler integriert sind, besteht die Möglichkeit, mehr Semantik hinzuzufügen, als wenn davon ausgegangen wird, dass es sich nur um eine Typdefinition auf Basis von `signed char` handelt.

Im Prinzip werden auch die Werte von `_Bool` und `bool` wieder als Zahlen 0 und 1 mit der Länge von einem Byte kodiert. Allerdings können Variablen dieser beiden Typen keine anderen Werte annehmen. Hierzu folgendes Beispiel:

```
bool var1 = true;
bool var2 = false;
bool var3 = 5;
```

Die drei Variablen `var1`, `var2` und `var3` können hierdurch ebenfalls als Zahlenwerte interpretiert werden. Die Variable `var1` besitzt dann den Wert 1 und die Variable `var2` den Wert 0. Die Variable `var3` besitzt im Vergleich zu `BOOL` allerdings nicht den Wert 5, sondern den Wert 1, da die 5 kein gültiger Wert für `bool` ist.

Der Grund hierfür ist, dass hinter jeder Wertzuweisung an eine Variable vom Typ `bool` oder `_Bool` eigentlich eine Abfrage der folgenden Form steckt:

```
bool var3 = 5 ? true : false;
```

Diesen Unterschied sollte man als Programmierer in Objective-C immer im Hinterkopf behalten. Und man sollte darauf achten, dass man bei Funktionen, die ein Argument vom Typ `BOOL` erwarten oder einen `BOOL`-Wert zurückgeben, auch Variablen vom Typ `BOOL` zur Übergabe der Werte verwendet. Entsprechendes gilt für die anderen Datentypen.