Holger
Hinzberg

iOS-Apps mit Swift 2

Das umfassende Praxis-Handbuch

Mit Referenzkarte

Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Holger Hinzberg

iOS-Apps mit Swift 2

Das umfassende Praxis-Handbuch

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 978-3-95845-222-0
1. Auflage 2016

www.mitp.de
E-Mail: mitp-verlag@sigloch.de
Telefon: +49 7953 / 7189 - 079
Telefax: +49 7953 / 7189 - 082

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Coverbild: © Max Krasnov @ fotolia.de
Satz: III-satz, Husby, www.drei-satz.de

Referenzkarte

Swift 2

mitp

Variablen und Konstanten

```
// Variablen und Konstanten
var variableValue = 3.1415
variableValue = 3.0

let constValue = 42
// Neue Zuweisung nicht möglich
// constValue = 50
```

Fallunterscheidungen

```
var salary = 10000

if salary > 15000
{
    print("Zu viel")
}
else if salary < 10000
{
    print("Zu wenig")
}
else
{
    print("Ok")
}

var index = 4

switch index
{
    case 1:
        print("Index ist 1")
    case 2:
        print("Index ist 2")
    case 3...5:
        print("Index liegt zwischen 3 und 5")
    case 6:
        print("Index ist 6")
        fallthrough
    default:
        print("Index liegt nicht zwischen 1 bis 5")
}
```

Schleifen

```
for var index = 0; index <= 5; index++
{
    print(index) // Ausgabe von 0 bis 5
}

for index in 0...5 // Closed Range
{
    print(index) // Ausgabe von 0 bis 5
}

for index in 0..<5 // Half-Open Range
{
    print(index) // Ausgabe von 0 bis 4
}

for index in (5...10).reverse()
{
    print(index) // Ausgabe von 10 bis 5
}

var access = 1
repeat
{
    print(access) // Ausgabe von 1 bis 9
    access++
}
while access < 10
```

Referenzkarte zum Buch »iOS-Apps mit Swift 2« von Holger Hinzberg, ISBN 978-3-95845-221-3,
www.mitp.de/221, mitp-Verlag

Array

```
var cities:[String] = ["Rom","Paris","Berlin"]
cities.append("London")

for city in cities
{
    print(city) // Ausgabe aller Städtenamen
}

var count = cities.count // Anzahl der Elemente
```

Methoden/Funktionen

```
// Ohne Parameter und ohne Rückgabewert
func doSomething()
{
    print("Hello World!")
}

doSomething()

// Mit einem Double als Parameter und einem Double
// als Rückgabewert
func doSomething(value:Double) -> Double
{
    return value * 2.0
}

var twice = doSomething(4.5)

// Mit zwei benannten Parametern
// und einem Double als Rückgabewert
func doSomething(
  firstValue first:Double, secondValue second:Double) -> Double
{
   return first + second
}

var sum = doSomething(firstValue: 3.2, secondValue: 4.7)
```

Referenzkarte zum Buch »iOS-Apps mit Swift 2« von Holger Hinzberg, ISBN 978-3-95845-221-3,
www.mitp.de/221, mitp-Verlag

Referenzkarte

Swift 2

mitp

Dictionary

```
var animals:[Int: String] = [11:"Hund", 22:"Katze", 33:"Maus"]
animals[22] = "Elefant" // Element ersetzen
animals[99] = "Huhn" // Element anfügen

print(animals[22]) // Ausgabe "Optional (Elefant)"
print(animals[99]) // Ausgabe "Optional (Huhn)"

animals[11] = nil // Element entfernen
```

Zeichenketten

```
var firstName = "Mike"
var lastName = "Müller"
// String-Interpolation
print("Mein Name ist \(firstName) \(lastName).")

// Zahlenwerte formatieren
var numeric = 3.34983632
var num = String(format:"%.2f", numeric) // Ausgabe 3.35
```

Klassen

```
class Person
{
    // Eigenschaften
    var firstName:String
    var lastName:String

    init()
    {
        // Eigenschaften initialisieren
        firstName = ""
        lastName = ""
    }
}

// Klasseninstanz erzeugen und
// Eigenschaften zuweisen
var pers = Person()
pers.firstName = "Mike"
pers.lastName = "Müller"
```

Enumerationen

```
enum Color
{
    case Red
    case Green
    case Blue
}
var exterior:Color = Color.Green
```

Referenzkarte zum Buch »iOS-Apps mit Swift 2« von Holger Hinzberg, ISBN 978-3-95845-221-3, www.mitp.de/221, mitp-Verlag

Fehlerbehandlung mit try-catch

```
// Eigene Fehlertypen definieren
enum CalculationError : ErrorType
{
    case DivideByZero
    case OtherError
}

// Funktion kann einen Fehler zurückgeben
func divide(dividend:Int, divisor:Int) throws -> Int
{
    guard divisor > 0 else
    {
        throw CalculationError.DivideByZero
    }
    return dividend / divisor
}

// Funktionsaufruf und Fehlerbehandlung
do
{
    var result = try divide(100, divisor: 0)
}
catch(CalculationError.DivideByZero)
{
    print("Es wurde versucht, durch null zu teilen")
}
catch
{
    print("Ein anderer Fehler ist aufgetreten")
}
```

Referenzkarte zum Buch »iOS-Apps mit Swift 2« von Holger Hinzberg, ISBN 978-3-95845-221-3,
www.mitp.de/221, mitp-Verlag

Inhaltsverzeichnis

Danksagung

Die Programmiersprache Swift und die iOS-Entwicklung sind aufregende Technologien und es gibt viele Themen, die ebenfalls ein Kapitel in diesem Buch verdient hätten. Als Autor habe ich die Erfahrung machen müssen, dass ein Buch nie wirklich fertig ist. Irgendwann muss man allerdings aufhören, zu schreiben, denn die Mitarbeiter vom Verlag möchten auch ihren Beitrag leisten.

Mein Dank geht an erster Stelle an meine geduldige Lektorin Sabine Schulz, die mir erneut die Gelegenheit gab, meine Ideen und Erfahrungen in einem Buch zu veröffentlichen. Ich danke ihr, auch stellvertretend für alle Mitwirkenden in der Korrektur, im Satz und in der Produktion.

Dank geht an meinen Freund Peter Mekelburg, der seit vielen Jahren bereitwillig alle meine Texte prüft und mich davor bewahrt, durch zu viele fehlende Satzzeichen und zu komplizierte Sätze unangenehm aufzufallen.

Nicht vergessen darf ich Horst Andreas Roemer, der ebenfalls gerne einen kritischen Blick auf meine Texte wirft. Unsere kreative Zusammenarbeit war wieder ein Spaß, den wir hoffentlich beim nächsten Buch fortsetzen können.

Mein besonderer Dank geht an alle Leser meiner Bücher, von denen ich einige inzwischen persönlich auf Konferenzen und Seminaren kennenlernen durfte. Ich hoffe, wir werden viele weitere Jahre zusammen Spaß haben!

Holger Hinzberg

Holzwickede, März 2016

Einleitung

Das iPhone – für viele technikbegeisterte Menschen ist dieser Name gleichbedeutend mit dem modernen Mobiltelefon, das inzwischen leistungsfähiger ist als die sperrigen Computer, die noch vor einigen Jahren auf vielen Schreibtischen zu finden waren. Auch Softwareentwickler zieht es seit Jahren zum iPhone. Zu verlockend ist die iOS-Plattform mit ihren Geräten. Neben leistungsfähigen Prozessoren verfügen das iPhone und seine Geschwister durch die eingebauten Sensoren und Kameras über Fähigkeiten, die ein klassischer Rechner nicht bieten kann. Eine Multi-Touch-Oberfläche, die es erlaubt, Programme direkt auf dem Bildschirm zu bedienen, bringt bisher unbekannte und aufregende Möglichkeiten. Verständlich ist der Wunsch vieler Programmierer, für diese Plattform zu entwickeln. Ein Einstieg war oft nicht leicht, denn seit über 20 Jahren wurde bei der Firma aus Kalifornien fast ausschließlich mit Objective-C programmiert – eine leistungsfähige Programmiersprache, die außerhalb von Apple aber kaum bekannt war.

Die Abkehr vom Vertrautem erfolgte am 1. Juni 2014, als Apple auf der WWDC-Konferenz in San Francisco die Programmiersprache Swift der Öffentlichkeit vorstellte. Für die Zuschauer war das eine Überraschung und bei vielen Entwicklern die Skepsis groß. Das Erlernen einer Programmiersprache braucht Zeit, und die ist für viele Programmierer, besonders jene, die mit der Softwareentwicklung ihren Lebensunterhalt verdienen, gleichbedeutend mit Geld. Dessen ungeachtet bringt Swift endlich Möglichkeiten für OS-X- und iOS-Projekte, die es in anderen Programmiersprachen schon seit vielen Jahren gibt. Typsicherheit, Closures und Optionals waren einige der Schlagwörter, mit denen man während der Präsentation das Herz der Programmierer schneller schlagen ließ. Ein Jahr später wurde Swift in der Version 2.0 vorgestellt. Ein untrügliches Zeichen, dass es Apple mit der neuen Sprache ernst meint. Inzwischen verwenden immer mehr Entwickler die Sprache Swift für neue Projekte. Verpassen Sie nicht die Gelegenheit, dabei zu sein!

An wen richtet sich das Buch?

Trotz meiner guten Vorsätze, eine wirklich einfache Einführung in die iOS-Entwicklung mit Swift zu schreiben, fängt auch dieses Buch leider nicht bei null an. Dafür reichen die zur Verfügung stehenden Seiten bedauernswerterweise nicht aus. Wenn Sie als Leser aber schon ein wenig Erfahrung in einer anderen objektorientierten Programmiersprache haben und jetzt iOS-Apps mit Swift schreiben

wollen, sollten die Beispiele für Sie keine großen Hürden sein. Dabei ist es auch unerheblich, ob Sie Vollzeitentwickler oder nur Hobbyprogrammierer sind. Wenn Begriffe wie Compiler, Objekte und Vererbung für Sie keine Fremdwörter sind, dann steht Ihnen der Weg offen, ein erfolgreicher iOS-Entwickler zu werden. Mit ein wenig Übung und einer guten Idee können Sie vielleicht schon bald Ihre erste App veröffentlichen!

Aufbau des Buches

Swift ist vielen anderen Programmiersprachen sehr ähnlich, aber trotzdem werden wir uns in den ersten Kapiteln zunächst mit einfachen Dingen wie Datentypen und Kontrollstrukturen beschäftigen, denn selbst bei den Grundlagen der Sprache gibt es Besonderheiten. Ein Schwerpunkt liegt außerdem in der Verarbeitung und Manipulation von Zeichenketten, da fast kein Programm ohne die Ausgabe von Texten auskommt. Zusätzlich werden Sie Apples Entwicklungsumgebung Xcode kennenlernen, die in der Version 6 um einen »Spielplatz« erweitert wurde, mit dem man Swift besonders gut lernen kann. In den weiteren Kapiteln folgen dann ein wenig anspruchsvollere Themen wie Arrays, Dictionaries, Funktionen, Klassen und Methoden. Falls Ihnen Tuples, Optionals und Closures noch nicht vertraut sind, werden Sie diese ebenfalls kennenlernen. Nicht fehlen dürfen selbstverständlich die Themen Vererbung und Protokolle. Was wäre die objektorientierte Programmierung ohne sie?

Die Entwicklung von iOS-Apps füllt den zweiten Teil des Buches. Hier lernen Sie, wie Textfelder, Schaltflächen und viele andere Steuerelemente funktionieren und wie eine App aufgebaut ist. MVC-Architektur, Delegation und Grafik mit Multi-Touch sind nur einige der behandelten Themen. Allerdings gibt es in diesem Buch kein einzelnes Projekt, das über mehrere Kapitel hinweg erweitert wird, sondern mehrere unabhängige Beispiele, an denen gezielt spezielle Technologien und Anwendungsfälle erklärt werden. Zunächst sind die Projekte noch klein und überschaubar, und erst die umfangreichen Themen werden uns etwas länger beschäftigen. Obwohl die Firma Apple in ihrer Entwicklungsumgebung für viele verschiedene Arten von Programmen Vorlagen anbietet, werden wir davon nur wenig Gebrauch machen und die Dinge selbst programmieren. Das verlangt zwar einen größeren Aufwand, gibt uns aber Gelegenheit, mehr über die Hintergründe und den Aufbau einer iOS-App zu erfahren.

Teil I: Grundlagen der Sprache Swift

- Kapitel 1: Datentypen und Optionals
  Wir beginnen mit einer Einführung in die Entwicklungsumgebung Xcode und spielen ein wenig mit Variablen und Konstanten. Dann sehen wir uns an, wie

man Zahlen rundet und aus einer Gruppe von Werten den kleinsten oder den größten ermittelt. Optionals, eine Besonderheit von Swift, werden Sie ebenfalls in diesem Kapitel kennenlernen.

- Kapitel 2: Zeichenketten
  Ohne Ausgabe von Texten sind die meisten Programme wenig nützlich, und deshalb haben Zeichenketten ein eigenes Kapitel verdient. Swift arbeitet mit Unicode und dort gibt es einiges zu beachten.
- Kapitel 3: Arrays und Dictionaries
  Eine Liste von Werten ist immer interessanter als viele einzelne Werte. Man kann sie sortieren, filtern oder der Reihe nach ansehen. Wenn es schnell gehen muss, helfen Wörterbücher, einen Eintrag aufzuspüren.
- Kapitel 4: Fallunterscheidungen und Schleifen
  Wenn Ihnen der sequenzielle Programmablauf zu langweilig ist, sollten Sie Schleifen und Fallunterscheidungen ausprobieren. Swift bietet verschiedene Möglichkeiten, Anweisungen zu wiederholen oder nur bedingt auszuführen. Vorwärts oder rückwärts durch ein Array springen ist dann auch kein Problem mehr.
- Kapitel 5: Funktionen
  Lassen Sie uns wiederverwendbaren Code schreiben. Den Anfang machen Funktionen. Mit ihnen können wir Anweisungen zusammenfassen und strukturieren. Parameter und Rückgabewerte sind dann auch ganz schnell erklärt und wir haben Zeit für Tupel und Aliasse.
- Kapitel 6: Closures
  Stellen Sie sich vor, Sie könnten nicht nur Werte in Variablen ablegen, sondern auch Blöcke aus Programmcode. Mit Closures ist das möglich. Sie sind vielseitig einsetzbar und können ohne große Mühe sogar asynchron innerhalb einer App ausgeführt werden. Genau richtig also, wenn Vorgänge, wie der Download eines Bildes, mal wieder etwas länger dauern.
- Kapitel 7: Klassen und Objekte
  Die Grundlagen der objektorientierten Programmierung dürfen nicht fehlen. Wie wird aus einer Klasse ein Objekt und was sind die Eigenschaften einer Klasse? Dieses Kapitel verrät es!
- Kapitel 8: Methoden
  Methoden sind Funktionen, die zu einer Klasse gehören. Meinen Sie, damit wäre alles gesagt? Dann lassen Sie uns über Klassenmethoden, Failable Initializers und Methoden zur Deinitialisierung reden.
- Kapitel 9: Vererbung und Assoziationen
  Nicht immer alles neu programmieren, sondern auf dem Bewährten aufbauen und erweitern. Das ist ein weiterer Grundsatz der objektorientierten Program-

mierung. Wenn es dann nicht passt, lassen sich Methoden auch überschreiben. Assoziationen können zusätzlich helfen, die einzelnen Bausteine zusammenzusetzen.

- Kapitel 10: Protokolle und Extensions
  Die Kommunikation zwischen verschiedenen Partnern kann kompliziert sein und deshalb ist es wichtig, dass man sich zuvor auf ein gemeinsames Protokoll einigt. So ist es auch in der Softwareentwicklung.
  Extentions ermöglichen es, bestehende Klassen um zusätzliche Methoden zu erweitern. Das funktioniert in Swift sogar ohne Vererbung.
- Kapitel 11: Strukturen und Enumerationen
  Es müssen nicht immer Klassen sein. Strukturen sind vielseitig und können in Swift Aufgaben übernehmen, die ihnen in anderen Programmiersprachen verwehrt sind. Geschickt eingesetzt lassen sich mit ihnen einige Probleme vermeiden. Enumerationen sind auf dem Weg zu gutem Code ebenfalls eine große Hilfe.
- Kapitel 12: Sicherer Programmcode
  Soll ein Programm bei fehlerhaften Daten weiterhin zuverlässig arbeiten, muss der Code für jede Situation optimal vorbereitet werden. Kommt es trotzdem zu einem Fehler, sollte man auch damit ganz gelassen umgehen. In weiser Voraussicht haben die Entwickler von Apple einige Techniken für uns vorbereitet.
- Kapitel 13: Speicherverwaltung mit Referenzzähler
  Um die Speicherverwaltung muss man sich in Swift nur selten Gedanken machen, ganz ignorieren kann man aber das Thema nicht. Hier erfahren Sie die Dinge, die Sie wissen sollten.
- Kapitel 14: Robuste Anwendungen und automatisierte Tests
  Die eigenen Projekte nach Veränderungen automatisch testen zu lassen, klingt wie der Traum vieler Programmierer. Mit der aktuellen Version von Apples Entwicklungsumgebung ist das jetzt möglich.

Teil II: iOS-Apps

- Kapitel 15: Lotto – 6 aus 49
  In diesem Kapitel werden Sie endlich Ihre erste App entwickeln und dabei Schaltflächen und Bezeichnungsfelder kennenlernen. Zuvor ist allerdings ein wenig Theorie nötig und Sie erfahren, was es mit dem MVC-Entwurfsmuster auf sich hat.
- Kapitel 16: Ein Blick hinter die Kulissen
  Wie startet eine App? Wie bekommt das Programm ein Icon und warum dreht sich die Ansicht, wenn wir das Gerät drehen? Die Antworten erhalten Sie durch einen Blick auf die internen Vorgänge.

- Kapitel 17: Eingaben mit virtuellen Tastaturen
  iOS-Apps bringen ihre eigenen Bildschirmtastaturen mit und deshalb gibt es einige Besonderheiten zu beachten. Welche Tastatur ist für welchen Anwendungsfall die richtige und wie kann eine Tastatur wieder ausgeblendet werden? In diesem Kapitel finden Sie die Antworten.
- Kapitel 18: Schieberegler, Textformatierungen und unendliche Werte
  Eine App zur Berechnung des Kraftstoffverbrauchs ist optimal geeignet für den Einsatz eines Schieberegler-Steuerelements. Ein weiterer Schwerpunkt in diesem Kapitel liegt auf der Formatierung von Zahlen in Abhängigkeit von den aktuellen Einstellungen. Nicht überall auf der Welt wird ein Komma als Dezimaltrennzeichen verwendet.
- Kapitel 19: Storyboards – Mit dem Drehbuch durch die App
  Der Bildschirm eines iOS-Geräts ist klein und umfangreiche Informationen können nur schwer angezeigt werden. Da kann ein Storyboard hilfreich sein, um von einer Ansicht in eine andere zu wechseln. Im grafischen Editor der Entwicklungsumgebung ergeben sich ganz neue Möglichkeiten, eine App zu planen und zu gestalten.
- Kapitel 20: Storyboards mit Protokollen
  Die Kommunikation zwischen verschiedenen Klassen ist nicht immer einfach, doch Protokolle helfen, für eine reibungslose Verständigung zu sorgen. In einem Projekt sorgen sie zusätzlich für größere Flexibilität, mehr Sicherheit und bessere Wiederverwendbarkeit von zuvor programmierten Komponenten.
- Kapitel 21: Navigation mit einem Tableisten-Controller
  Eine Tableiste ist für den Nutzer einer App eine bequeme Möglichkeit, zwischen verschiedenen Ansichten umzuschalten, und für einen erfahrenen Entwickler auch schnell umzusetzen. Das gibt mir genug Gelegenheit, das Thema »Delegation« anzusprechen. Mit dieser Technik, die in fast allen Bereichen der iOS-Entwicklung eingesetzt wird, erhalten wir ganz neue Möglichkeiten in unseren Programmen.
- Kapitel 22: Grafische Oberflächen ohne Interface Builder
  Der grafische Editor der Entwicklungsumgebung ist ein mächtiges Werkzeug, doch wenn es sein muss, kann die grafische Oberfläche einer App auch durch Anweisungen im Programmcode erzeugt werden. Das schafft Flexibilität, und Sie erfahren, wie die Kommunikation mit grafischen Elementen funktioniert.
- Kapitel 23: Serialisierung – Aus dem Speicher in eine Datei
  In diesem Kapitel beschäftigen wir uns wieder mit Arrays und wie man sie aus dem Speicher des Geräts in eine Datei überträgt und wieder lädt. Außerdem erfahren Sie, wie die Nachrichtenzentrale verschiedenen Programmteilen dabei hilft, miteinander zu kommunizieren.

- Kapitel 24: Der Picker – ein ganz besonderes Steuerelement
  iOS verwendet viele grafische Objekte, die dem Anwender schon von Desktop-Anwendungen her vertraut sind und deshalb nur wenig Umgewöhnung verlangen. Neu in der mobilen Welt ist dagegen das Picker-Steuerelement, das auch in der Programmierung einen etwas ungewöhnlichen Weg einschlägt. Ein weiteres Mal treffen wir auf Protokolle und Delegation.
- Kapitel 25: Tabellen
  Tabellen mit Master- und Detailansicht sind unter iOS ebenfalls ein spannendes Thema, das ein eigenes Kapitel mehr als verdient hat. Sehr viele Apps verwenden Tabellen, auch wenn diese manchmal gar nicht mehr als solche zu erkennen sind.
- Kapitel 26: Eye Candy – Unsere App soll schöner werden
  Unsere App soll schöner werden, und Rahmen um die Bilder oder abgerundete Ecken sind nur zwei der möglichen Verbesserungen. Gefallen dem Anwender diese Stilmittel nicht, dann sollten wir ihm die Möglichkeit geben, sie in den Systemeinstellungen zu deaktivieren.
- Kapitel 27: Der Collection View
  Das Collection-View-Steuerelement ist eine Alternative zur Tabellenansicht und ermöglicht eine andere Darstellung von Informationen. Zusätzlich beschäftigen wir uns in diesem Kapitel mit den Themen Drucken und wie wir mit unseren Apps auf Twitter und Facebook Bilder und Texte veröffentlichen können.
- Kapitel 28: Zeichnen mit Core Graphics
  Trotz der Vielzahl an vordefinierten Steuerelementen ist es manchmal erforderlich, den digitalen Pinsel selbst in die Hand zu nehmen und Objekte oder Symbole, die man benötigt, selbst zu zeichnen. Das Core Graphics Framework bietet uns nahezu unbegrenzte Möglichkeiten, kreativ zu werden und Formen und Farbverläufe nach unseren Wünschen zu gestalten.
- Kapitel 29: Multi-Touch mit Gestenerkennung
  Die Möglichkeit, Berührungen auf dem Bildschirm zu erkennen und auszuwerten, ist eine der faszinierenden Eigenschaften von iPhone, iPad und iPod touch. Mit nur wenigen Anweisungen können Sie die Objekte auf dem Bildschirm mit den Fingern verschieben, drehen oder skalieren.
- Kapitel 30: Digitale und analoge Uhren mit Timern
  Soll eine bestimmte Funktion einer App immer wieder ausgeführt werden, greift der Entwickler schnell zu einem Timer. Damit ist es ein Leichtes, immer die aktuelle Uhrzeit anzuzeigen. In diesem Kapitel bietet sich Ihnen außerdem die Gelegenheit, zu zeigen, was Sie über Blöcke und Core Graphics inzwischen gelernt haben. Unsere Uhr bekommt ein Zifferblatt und sogar einen Sekundenzeiger.

- Kapitel 31: Lokale Benachrichtigungen
  Wird Ihre App momentan nicht ausgeführt? Sie wollen den Anwender aber trotzdem über wichtige Ereignisse informieren? Ein scheinbar unlösbares Problem – doch lokale Benachrichtigungen schaffen Abhilfe.
- Kapitel 32: Karten und Koordinaten
  Wo bin ich und was gibt es hier zu sehen? In diesem Kapitel beschäftigen wir uns mit Straßenkarten und wie Sie mehr über Ihren aktuellen Standort erfahren können. In den Frameworks Core Location und Map Kit finden sich einige sehr interessante Funktionen.
- Kapitel 33: Daten suchen und finden
  In einer großen Menge an Daten schnell eine gesuchte Information zu finden, ist für jeden Programmierer eine Herausforderung. Zum Glück gibt es im iOS bereits Funktionen, die einen Großteil der Arbeit für uns übernehmen. Eine Tabelle für die Erweiterung einer Suchleiste ist dann auch kein großes Problem mehr.

Swift ist nicht Objective-C Version 3.0

Bevor es im nächsten Kapitel mit der Swift-Entwicklung losgehen kann, sind einige einführende Worte über die Programmiersprache nötig. Das gilt besonders für alle Entwickler, die bisher mit Objective-C auf den Apple-Plattformen gearbeitet haben. Einige Dinge, die Ihnen seit Jahren vertraut sind, funktionieren jetzt anders und müssen neu erlernt werden.

Sieht man sich als erfahrener Entwickler den Quellcode eines Swift-Programms an, bekommt man schnell den Eindruck, es mit einer weiteren auf C basierenden Sprache zu tun zu haben. Tatsächlich ist die Syntax von Swift anderen Sprachen der C-Familie in manchen Bereichen ähnlich, unterscheidet sich aber in anderen Teilen genauso oft. Swift ist zweifellos mit C verwandt, aber keineswegs mit der Sprache kompatibel. Das war bei Objective-C, der Sprache, mit der für die Apple-Plattformen bisher bevorzugt entwickelt wurde, anders. Sie können in Apples Xcode-Entwicklungsumgebung ein Objective-C-Projekt anlegen und dort ohne weitere Anpassungen 20 Jahre alten C-Code einfügen, ohne auf Probleme zu stoßen. Mit Swift funktioniert das nicht mehr. Die Sprache hat den Anspruch, sicher zu sein, und muss deshalb Anforderungen erfüllen, auf die man in der Vergangenheit weniger Wert legte. So ist es beispielsweise nicht mehr möglich, bei einer `if`-Struktur auf den Block aus geschweiften Klammern zu verzichten. Einige andere Programmiersprachen erlauben das, wenn nur ein Befehl bedingt ausgeführt werden soll, aber es ist dann auch nur genau eine Anweisung. Ist der Programmcode jedoch ungünstig formatiert, kann es bei dem Entwickler leicht zu Fehlinterpretationen kommen. Mit Swift hat Apple sich das Ziel gesetzt, solche

und ähnliche Situationen zu vermeiden. Nicht mehr schreiben muss man hingegen die runden Klammern, mit denen in vielen anderen Sprachen die Bedingung der `if`-Struktur eingeklammert wird. Im Gegensatz zu dem Block aus geschweiften Klammern erhöhen sie weder die Lesbarkeit noch die Sicherheit des Codes und auch sonst haben die Klammern keinen weiteren Nutzen.

```
var value = 10000

if value == 10000
{
   print("Der Wert ist 10.000!")
}
else
{
   print("Der Wert ist nicht 10.000.")
}
```

Ein Semikolon nach einer Anweisung ist in Swift ebenfalls nicht mehr erforderlich: Der Compiler kann das Ende eines Befehls ohne diese Markierung erkennen. Sollten Sie trotzdem weiterhin Semikolons schreiben, vielleicht weil Sie das aus jahrelanger Gewohnheit automatisch tun, wird Ihr Code trotzdem akzeptiert. Nötig wird ein Semikolon nur, wenn Sie mehrere Anweisungen in eine Zeile schreiben wollen. Um den Programmcode lesbar zu halten, sollte man dies aber vermeiden.

Wie Sie in den folgenden Beispielen sehen werden, benötigt ein Swift-Programm nicht zwingend eine `main`-Methode, um starten zu können. Natürlich ist es wichtig, den Startpunkt einer Anwendung genau festzulegen, allerdings ermöglicht Swift auch Programme, die anders arbeiten. Ein »Playground« kann ein vollwertiges Programm in einer Datei abbilden, die von oben nach unten abgearbeitet wird. Im nächsten Kapitel werden wir uns mit diesem Dokumententyp und seinen Möglichkeiten detailliert beschäftigen.

Sind Sie bereits Entwickler für die Apple-Plattformen, werden Sie vermutlich die eckigen Klammern vermissen, die in Objective-C immer zum Einsatz kamen, wenn mit Klassen und Objekten programmiert wurde. Auf diese besondere Syntax wurde in Swift ebenfalls verzichtet. Die Sprache verwendet stattdessen für Eigenschaften und Methodenaufrufe die Punktnotation, wie sie in Java und C# schon seit vielen Jahren üblich ist. Auch das Konzept, dass sämtliche Objekte im Speicher unabhängig voneinander existieren und nur durch Nachrichten miteinander kommunizieren, wird bei Swift nicht mehr angewendet. Dies führt zwar gegenüber Objective-C zu Einbußen in der Flexibilität, ermöglicht Swift im Gegenzug aber schnellere Methodenaufrufe.

Kommentare

Die Syntax der Schreibweise von Kommentaren hat sich im Vergleich zu C und Objective-C nicht verändert. Es gibt weiterhin zwei verschiedene Möglichkeiten, Anmerkungen im Programmcode zu hinterlegen. Die erste und einfachste ist die Kommentarzeile, eingeleitet durch zwei Schrägstriche.

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

Kommentarzeilen eignen sich gut, um Variablen zu beschreiben. Es ist sogar möglich, den Kommentar direkt hinter einer Anweisung einzufügen. Nach den zwei Schrägstrichen wird der noch folgende Text zu einem Kommentar.

```
// Das Alter der Person
var ageOfPerson:Int = 45
var salaryOfPerson:Int = 45000 // Das Gehalt der Person
```

Möchte man mehr als eine Kommentarzeile in den Code einfügen, können komplette Abschnitte als Kommentarblock ausgewiesen werden. Mit der Zeichenfolge /* beginnt ein solcher Block, der mit */ wieder beendet wird. Neben dem Einsatz zur Dokumentation können Kommentarzeilen und Kommentarblöcke verwendet werden, um Anweisungen vorübergehend aus einem Programm zu entfernen. So müssen die Befehle nicht gelöscht werden, falls man sie zu einem späteren Zeitpunkt wieder benötigt.

```
// Eine auskommentierte Anweisung
/*
var tax:Double = 0.19
*/
```

Wie Sie spätestens bei unserem ersten Projekt sehen werden, sind Kommentare im Editor der Entwicklungsumgebung immer durch eine spezielle Farbe gekennzeichnet, voreingestellt ist Grün. So lassen sich auskommentierte Anweisungen leicht erkennen.

Nicht alle Dokumente und Lehrbücher weisen darauf hin, aber Kommentare sind wichtig! Nicht nur für Sie, sondern auch für andere Entwickler, die vielleicht in Zukunft mit Ihrem Programmcode in Kontakt kommen. Wenn sich die Möglichkeit bietet, sollten Sie immer versuchen, Ihre Arbeit und Erfahrungen mit anderen Programmierern auszutauschen. Egal, wie viele Kommentare Sie schreiben, diese haben keinerlei Einfluss auf die Dateigröße oder die Geschwindigkeit der Anwendung. Es gibt also keinen Grund, auf Kommentare im Programmcode zu verzichten! Bei den Beispielen in diesem Buch werden Sie ebenfalls viele wichtige Anmerkungen in den Listings finden.

Stilmittel in den Listings

Das begrenzte Seitenformat dieses Buches macht es in manchen Situationen erforderlich, den Programmcode etwas anders zu formatieren, als es in der Xcode-Entwicklungsumgebung möglich wäre. Gelegentlich werden lange Befehle, die im Texteditor auf dem Computer in eine Zeile passen, für die Codebeispiele auf mehrere Zeilen verteilt. Unabhängig von der Schreibweise ergeben sich bei den abgedruckten Listings aber keine Unterschiede in der Funktion.

An manchen Stellen im Buch werden Ihnen im Programmcode drei Punkte begegnen, die weder eine Anweisung der Programmiersprache Swift noch ein geheimes Zeichen der Apple-Entwickler sind:

```
...
```

Diese Markierung dient ausschließlich als Hinweis für Sie, dass es sich bei den gezeigten Anweisungen nur um einen Ausschnitt eines schon zuvor gezeigten Listings handelt. Die Punkte werden als Stilmittel verwendet, um im Buch nicht wertvollen Platz mit Wiederholungen bekannten Codes zu füllen.

Wird im Laufe eines Kapitels vorhandener Programmcode geändert oder ersetzt, sind die neuen Anweisungen innerhalb des Listings **fett** gedruckt. Sie müssen dann nur diese Befehle bearbeiten. Der übrige Code bleibt unverändert.

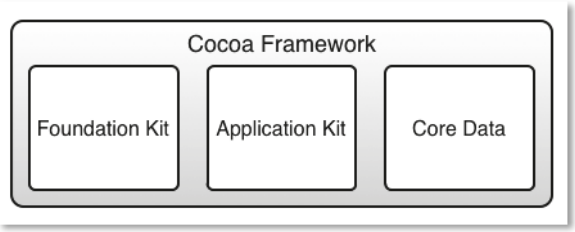
Die Frameworks der Apple-Plattformen

Unabhängig davon, für welches System man entwickeln möchte, ist neben dem Umgang mit der Programmiersprache oft eine umfangreiche Kenntnis der Systembibliotheken, den *Frameworks*, unverzichtbar.

Frameworks entbinden Entwickler von der mühevollen Aufgabe, sämtliche Komponenten für eine Anwendung eigenhändig programmieren zu müssen. Stattdessen ermöglichen sie den Zugriff auf bewährte Bausteine für nahezu sämtliche Anwendungsfälle. Wenn man sich mit der Entwicklung für die OS-X-Plattform beschäftigt, stößt man deshalb früher oder später unvermeidlich auf den Begriff *Cocoa Framework*. Der Begriff Framework ist in Verbindung mit Cocoa allerdings nicht sehr präzise, denn hinter dieser Bezeichnung verbirgt sich kein eigenes Framework, sondern der Zusammenschluss von drei einzelnen Bibliotheken: Application Kit, Core Data und Foundation.

Application Kit, oft abgekürzt »App Kit«, enthält die Bausteine für die Entwicklung grafischer Benutzeroberflächen. Es enthält Klassen für die unterschiedlichsten Steuerelemente wie beispielsweise Fenster, Schaltflächen und Schieberegler. Durch die Verwendung eines gemeinsamen Frameworks kann sichergestellt wer-

den, dass diese Elemente in allen Programmen nicht nur gleich aussehen, sondern sich auch identisch verhalten. 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.



Das Foundation Framework, gelegentlich nur als »Foundation« bezeichnet, enthält unzählige Klassen und Datenstrukturen, die es in der Sprache Swift nicht gibt, und ist somit die wichtigste Bibliothek. Foundation kann auf eine bewegte Geschichte zurückblicken, denn dieses Framework wurde von der Firma NeXT entwickelt, bevor das Unternehmen von Apple übernommen wurde. Zwar wurde Foundation seitdem mehrfach erweitert, seine Wurzeln sind aber noch immer leicht zu erkennen. Fast alle Klassen aus dieser Bibliothek beginnen mit den Buchstaben NS, die für das Betriebssystem *NeXT Step* stehen, das später zu Mac OS X wurde.

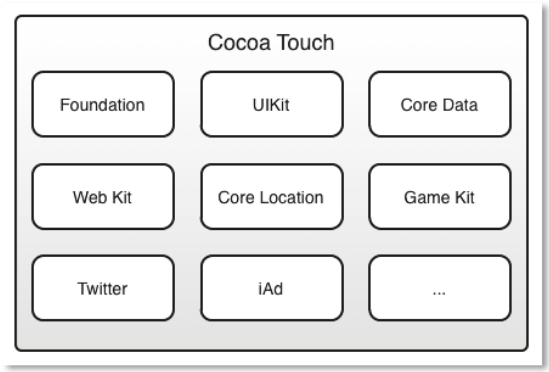
Einige Datentypen aus dem Foundation Framework braucht man in Swift nur noch selten. So gibt es für Zeichenketten jetzt den Typ `String` und man muss nicht länger die Klasse `NSString` verwenden, wie es bei der Objective-C-Programmierung noch der Fall ist. Nach Aussage von Apple sind die Swift-Klassen und die bewährten Klassen aus dem Foundation Framework jedoch austauschbar. Eine Methode, die einen `NSString` erwartet, würde sich problemlos mit einem `String` zufriedengeben. Das ist besonders beim Einsatz von Bibliotheken von Drittanbietern wichtig. Leider fehlen den neuen Typen einige Funktionen, die besonders von langjährigen Objective-C-Entwicklern sehr vermisst werden. Oft muss man neue Wege gehen, um an das gleiche Ziel zu gelangen. Es spricht aber nichts dagegen, in Swift weiterhin die aus Objective-C bekannten Klassen zu verwenden, falls das nötig sein sollte.

Im Laufe der Jahre wurden von Apple zusätzliche Frameworks entwickelt, die streng genommen kein Teil von Cocoa sind. Trotzdem können auch sie problemlos in Programmen eingebunden werden und funktionieren dort reibungslos mit den anderen Komponenten. Zwei der bekanntesten Erweiterungen sind Core Animation und WebKit. Letzteres ist eine Bibliothek zur Darstellung von Internetseiten, die auch im Safari-Browser verwendet wird.

Wendet man sich der iOS-Plattform zu, bekommt man es mit dem *Cocoa Touch Framework* zu tun, das ebenfalls einen Zusammenschluss einzelner Bibliotheken darstellt. Wie man am Namen schon gut erkennen kann, findet man dort spezielle Komponenten, um Programme durch Berührungen des Bildschirms mit den Fingern zu bedienen. Der Aufbau der grafischen Oberfläche wird deshalb nicht länger von App Kit übernommen, sondern wird zu einer Aufgabe des UIKit Frameworks. Dieses hat seine eigenen Steuerelemente, die bei einem Einsatz unter OS X und der Bedienung mit der Maus wenig nützlich wären.

Das Foundation Framework des Macs wurde vom iPhone übernommen, was sich in vielen Situationen als sehr praktisch erweist. Manche Klassen, die für OS-X-Anwendungen geschrieben wurden, können ohne Anpassungen in iOS-Apps erneut verwendet werden.

Im Gegensatz zu Cocoa wurde Cocoa Touch von Anfang an großzügiger geplant und enthält zusätzliche Frameworks. Map Kit und Core Location ermöglichen den Zugriff auf den GPS-Empfänger und auf Straßenkarten, während Game Kit Spielen den Informationsaustausch ermöglicht. Seit einigen Jahren kann sogar der ins Betriebssystem integrierte Twitter-Dienst von Entwicklern in eigenen Programmen genutzt werden.



Innerhalb eines Projekts ist es nicht immer einfach, zwischen Programmiersprache und Framework zu unterscheiden, denn die Bausteine sind sehr eng miteinander verzahnt. Ähnlich wie die Grammatik einer Fremdsprache ist Swift für den Aufbau und die Syntax der Anweisungen verantwortlich, während die Klassen der Bibliotheken das Vokabular der Sprache bilden.

Die Installation von Xcode

Das Werkzeug zur Swift-Programmierung, die Entwicklungsumgebung *Xcode*, wird von der Firma Apple kostenlos zum Download angeboten und kann direkt