# Inhaltsverzeichnis

Vorw	ort	25
TEIL	. I Swift	
1	Hello World!	29
1.1	»Hello World« im Playground	30
	Voraussetzungen	30
	Apple Developer Program	30
	Den Playground starten	30
	Hello World!	31
	Grafische Darstellung von Daten	32
	Darstellung von Objekten	33
	Kommentare	35
	Playgrounds mit mehreren Dateien	36
1.2	»Hello World« als Terminal-App	37
	Xcode kennenlernen	37
	Wo ist die App?	40
	Mehr als nur »Hello World!«	41
	Den Swift-Interpreter und -Compiler direkt aufrufen	42
1.3	Swift-Crashkurs	45
	Elementare Syntaxregeln	45
	Kommentare	46
	Markdown-Kommentare	46
	Variablen und Konstanten	47
	Zahlen und Zeichenketten	47
	Datentypen und Optionals	48
	Tupel, Arrays und Dictionaries	50
	Schleifen	51
	Verzweigungen	52
	Funktionen	52
	Closures	54

	Klassen und Datenstrukturen	54
	Fehlerabsicherung	55
1.4	Xcode-Crashkurs	56
	Navigator	57
	Editor	58
	Assistenzeditor	58
	Maus- bzw. Trackpad-Kürzel	60
	Tastenkürzel und Editoreinstellungen	60
	Werkzeugleiste (Inspector und Library Pane)	61
	Code-Snippets	62
	Speichern	63
	Versionsverwaltung (Git)	63
	Vorhandenen Code für eine neue Swift-Version anpassen	66

## 2 Operatoren

2.1	Zuweisungs- und Rechenoperatoren	69
	Einfache Zuweisung	69
	Wert- versus Referenztypen	70
	Elementare Rechenoperatoren	71
	Zeichenketten aneinanderfügen	72
	Inkrement und Dekrement	73
	Rechnen mit Bits	73
	Kombinierte Rechen- und Zuweisungsoperationen	74
2.2	Vergleichsoperatoren und logische Operatoren	74
	Vergleichsoperatoren	74
	== versus ===	75
	Vergleiche mit ~=	76
	Datentyp-Vergleich (»is«)	77
	Casting-Operator (»as«)	77
	Logische Operatoren	78
2.3	Range-Operatoren	78
	Interval-Operatoren	79
2.4	Operatoren für Fortgeschrittene	80
	Ternärer Operator	80
	Unwrapping- und Nil-Coalescing-Operator	81
	Optional Chaining	82
	Operator-Präferenz	82
2.5	Operator Overloading	83
	Vergleichsoperator für Zeichenketten	84

# 3 Variablenverwaltung und Datentypen

-	-
v	-
~	-
~	-

3.1	Variablen und Konstanten	85
	Deklaration von Variablen	85
	Regeln für Variablennamen	86
	Konstanten	87
	Eigenschaften	88
	Enumerationen (Enums)	89
3.2	Zahlen und boolesche Werte	91
	Ganze Zahlen	91
	Fließkommazahlen	92
	Typumwandlungen	93
	Zufallszahlen	93
	Double-Zufallszahlen	94
	Boolesche Werte	94
3.3	Zeichenketten	94
	String-Eigenschaften	95
	Syntax	96
	Funktionen und Methoden zur Bearbeitung von Zeichenketten	98
	Länge von Zeichenketten ermitteln	98
	Vergleichen und sortieren	99
	Suchen und ersetzen	100
	Reguläre Ausdrücke	102
	Bestandteile von Zeichenketten	104
	split und join	105
	Zeichenketten manipulieren	106
	Palindrom-Test	107
	Teilzeichenketten extrahieren	108
	Teilzeichenketten komfortabler auslesen	109
	Zahlen formatieren	111
	Zahlen mit dem NSNumberFormatter formatieren	112
	Zeichenketten in Zahlen umwandeln (parsen)	113
	Zahlen mit dem NSNumberFormatter parsen	114
3.4	Datum und Uhrzeit	115
3.5	Optionals	117
	Optionals deklarieren	118
	Optionals auslesen	118
	nil-Test und if-let	119
	Optional Chaining	120
	Nil Coalescing	121

3.6	Interna der Variablenverwaltung	122
	Wert- versus Referenztypen	122
	Datentypen	123
	Syntaktischer Zucker	124
	Typen-Aliase	125
	Datentyp ermitteln und ändern (Casting)	125
	Reflection	126
	Speicherverwaltung	127
	weak und unowned	129
	weak-Beispiel	129

# 4 Arrays, Dictionaries, Sets und Tupel

4.1	Arrays	133
	Arrays initialisieren	134
	Array-Elemente auslesen	135
	Arrays manipulieren	137
	Arrays sortieren	138
	Interna und Geschwindigkeitsüberlegungen	139
	Array-Elemente verarbeiten	139
	Array-Algorithmen	141
	Mehrdimensionale Arrays	142
4.2	Dictionaries	143
	Dictionaries deklarieren und initialisieren	143
	Zugriff auf Dictionary-Elemente	144
4.3	Sets	145
4.4	Option-Sets (OptionSetType)	146
	Anwendungsbeispiel	147
	Eigene Option-Sets definieren	148
4.5	Tupel	149
	Anwendungen	150
5	Verzweigungen und Schleifen	153

5.1	Verzweigungen mit if	153
	if	153
	if-let-Kombination für Optionals	154
	if-let-Kombination mit where	155

6	Funktionen und Closures	173
	Version 2: Swift zeigt, was es kann	170
	Einige Benchmarktests	169
	Version 1: elegant, aber langsann	107
JT	Version 1. elegant aber langcam	167
5.4	Lottosimulator	166
	continue	166
	break	165
	repeat-while	165
	while-let-Kombination	164
	while	164
	for-in	163
	for	162
5.3	Schleifen	162
	case-let-Kombination mit where	161
	switch für Tupel	160
5.2	Verzweigungen mit switch	158
	Versionsabhängige Code-Teile	158
	Inverse Logik mit guard	156

6.1	Funktionen definieren und ausführen	173
	Benannte Parameter	174
	Rückgabewerte	175
	Aufräumarbeiten automatisch ausführen (defer)	177
	Funktionsnamen	179
	Gültigkeitsebenen	179
	Verschachtelte Funktionen	180
	Rekursion	181
6.2	Parameter	182
	Gewöhnliche Parameter	183
	Veränderliche Parameter	184
	Inout-Parameter	184
	Benannte Parameter	185
	Differenzierung zwischen externen und internen Parameternamen	185
	Auch den ersten Parameter benennen	186
	Unbenannten Parameter erzwingen	187
	Optionale Parameter und Defaultwerte	187
	Variable Parameteranzahl (Variadics)	189

6.3	Standardfunktionen	189
	Sequenzen bearbeiten	191
	prefix und suffix	191
	dropFirst und dropLast	191
	zip	192
	advance und distance	192
	split und join	193
	lazy	194
	Sonstige Funktionen	194
	Mathematische Funktionen	195
6.4	Standardmethoden und Standardeigenschaften	196
	Von der globalen Funktion zur Methode	196
	count	197
	first und last	197
	startsWith, contains und indexOf	198
	filter, map und reduce	199
	forEach	200
	sort und reverse	201
6.5	Funktionale Programmierung	201
	Funktionen als eigener Datentyp	202
	Funktionen als Parameter und Rückgabeergebnisse	203
6.6	Closures	206
	Syntax	206
	Auto-Closures	200
	RPN-Rechner	210
	Capturing Values	211
	Gefahr von Memory Leaks (Canture Lists)	213
	Closure-Speicherung und -Weitergabe verhindern (@noescape)	214
7	Objektorientierte Programmierung I	217
71	Klassen und Strukturen	218
/.=	Auch Enumerationen sind Datentypen	210
	Auch Endmerationen sind Datentypen:	219
		219
	Das Schlüsselwort »self«	220 222
	Zugriffsehenen und Zugriffssteuerung	222 222
	Andifizierer	∠∠∠ 221
	Verschachtelte Klassen Strukturen und Enumerationen	∠∠+ ))/
	Code-Dateien	∠∠4 ))⊑
		223

225

	Datentypen und Protokolle für Enumerationen	226
	Zuordnung von Zusatzdaten (Associated Values)	227
	Rekursive bzw. indirekte Enumerationen	228
7.3	Eigenschaften	230
	Verzögerte Initialisierung von Eigenschaften (Lazy Properties)	231
	Eigenschaften beobachten (willSet, didSet)	232
	Statische Eigenschaften	234
	Computed Properties (get und set)	235
	Temperaturumrechnung mit Computed Properties	235
	Read-Only-Eigenschaften	236
	Beispiel: Rectangle-Struktur	237
	Beispiel: ChessFigure-Struktur	239
7.4	Init- und Delnit-Funktion	241
	Syntax für Init-Funktionen	242
	Parameterliste	242
	Code-Reihenfolge in Init-Funktionen	243
	Overloading	243
	Designated versus Convenience Init	244
	Init-Funktion als Optional (Failable Init Functions)	245
	Deinit-Funktion	246
7.5	Methoden	246
	Instanzmethoden	247
	Mutating Methods	248
	Statische Methoden	250
	Benannte Parameter	251
	Benannte Parameter in Init-Funktionen und Methoden	253
	Signaturen von Methoden	253
7.6	Subscripts	255
	Beispiel: Schachbrett	256
8	Objektorientierte Programmierung II	259
8.1	Vererbung	259
	Mehrfachvererbung	260
	Vererbung versus Protokolle versus Extensions	260
	Das Schlüsselwort override	261
	Das Schlüsselwort super	263
	Das Schlüsselwort final	264
	Initialisierung	265

Enumerationen

7.2

	Das Schlüsselwort required	266
	Redundanz in Init-Funktionen vermeiden	267
	Generalisierung, Polymorphie und Casting	268
8.2	Generics	270
	Syntax	271
	Generics in der Swift-Standardbibliothek	271
	Regeln für generische Typen (Type Constraints)	273
8.3	Protokolle	274
	Vorhandene Protokolle implementieren	275
	Selbst Protokolle definieren	276
	Protokolle sind Datentypen	278
	Beispiel	278
	Optionale Protokollanforderungen	280
	Generische Protokolle mit »typealias«	281
8.4	Standardprotokolle	283
	CustomStringConvertible (ehemals Printable)	283
	Hashable und Equatable	285
	Comparable	286
	Any und AnyObject	287
	AnyClass	289
	StringLiteralConvertible	289
8.5	Extensions	291
	Syntax	292
	Übersichtlicherer Code durch Extensions	294
	Beispiel: Rechnen mit Kilo-, Mega- und Gigabyte	295
8.6	Protokollerweiterungen	296
	Bedingte Protokollerweiterungen	297
	Beispiel: Die uniqueElements-Methode	298
	Beispiel: Die uniqueSet-Methode	300
8.7	Metatypen	301
8.8	Header-Code einer eigenen Bibliothek erzeugen	302
9	Fehlerabsicherung und Spezialfunktionen	305
9.1	Fehlerabsicherung (try/catch)	305
	Swifts Verhalten beim Auftreten von Fehlern	305
	try-catch-Syntax	306
	Einführungsbeispiel	307
	Reaktion auf Fehler mit catch	308

	Selbst Fehler auslösen (throws und throw)	310
	Fehler in Init-Funktionen auslösen	311
	Fehler in Computed Properties	313
	Das ErrorType-Protokoll	313
	try ohne do-catch	315
	try! für Optimisten	315
	Parameterabsicherung mit guard	315
	Aufräumarbeiten mit defer	316
	assert	317
9.2	Fehlerabsicherung von API-Methoden (NSError)	318
	Die NSError-Klasse	319
	Die NSException-Klasse	320
9.3	Module, Frameworks und Importe	320
	Selbst Frameworks erzeugen	321
9.4	Attribute	322
9.5	Systemfunktionen aufrufen	323

## TEIL II iOS

### 10 Hello iOS-World!

10.1	Projektstart	328
10.2	Gestaltung der App	329
	Mini-Glossar	329
	Steuerelemente einfügen	330
	Ein erster Test mit dem iOS-Simulator	332
10.3	Steuerung der App durch Code	334
	Den Button mit einer Methode verbinden (Actions)	334
	Zugriff auf das Textfeld über eine Eigenschaft (Outlets)	336
	Endlich eigener Code	337
10.4	Actions und Outlets für Fortgeschrittene	339
	Eine Action für mehrere Steuerelemente	339
	Ein Outlet für mehrere Steuerelemente (Outlet Collections)	339
	Actions oder Outlets umbenennen	340
	Steuerelemente kopieren	341
10.5	Layout optimieren	341
	Layoutregeln	341
	Layoutregeln für den »Hello-World«-Button	342

	Layoutregeln für das Textfeld	344
	Wenn es Probleme gibt	345
10.6	Textgröße mit einem Slider einstellen	346
	Das Slider-Steuerelement hinzufügen	346
	Den Slider mit einer Methode verbinden	347
10.7	Apps auf dem eigenen iPhone/iPad ausführen	348
	Apple Developer Program	349
10.8	Komponenten und Dateien eines Xcode-Projekts	350
	Weitere Dateien	351
	Test- und Produktgruppe	352

# 11 iOS-Grundlagen

11.1	Model-View-Controller (MVC)	353
	Kommunikation in MVC-Apps	355
	MVC bei Apps mit mehreren Dialogen	356
11.2	Klassenhierarchie einer App-Ansicht	357
11.3	Die UIViewController-Klasse	361
	Lebenszyklus eines View-Controllers	361
	Init-Funktion	362
	viewDidLoad-Methode	363
11.4	Phasen einer iOS-App	365
	Die AppDelegate-Klasse	365
	Zugriff auf den Root-View-Controller und das AppDelegate-Objekt	367
11.5	Auto Layout	368
	Grundeinstellungen	368
	Viele Wege führen zum Ziel	369
	Live-Vorschau in der Preview-Ansicht	371
	Layoutregeln im Storyboard-Editor einstellen	372
	Layoutregeln manuell einstellen	374
	Layoutregeln aus der aktuellen Position und Größe ableiten	375
	Regeln ändern und löschen, Steuerelemente neu positionieren	376
	Layoutprobleme in der Document-Outline-Seitenleiste beheben	377
	Layoutregeln im Size Inspector bearbeiten	379
	Layoutdetails im Attributinspektor modifizieren	380
	Layouts für verschiedene iOS-Geräteklassen (Size Classes)	381
	Tipps und Tricks	383
	Layoutregeln mit Code definieren	384

11.6	Steuerelemente in einer Stack-View anordnen	386
	Funktionsprinzip	387
	Beispiel	388
	Content Compression Resistance Priority	389
11.7	Daten persistent speichern	390
	User-Defaults	391
	Umgang mit Dateien	393
	Zugriff auf Bundle-Dateien	394
	Beispiel	395
11.8	Mehrsprachige Apps	398
	Localization versus Internationalization	398
	Defaulteinstellungen in Xcode	399
	Deutsch als primäre Sprache einstellen	400
	Sprache hinzufügen	401
	Lokalisierungsdateien exportieren	402
	Lokalisierungsdateien bearbeiten	402
	Übersetzte Dateien wieder importieren	404
	Die App in verschiedenen Lokalisierungen ausprobieren	404
	Internationalisierung im Code	405
11.9	iOS-Crashlogs	407

# Apps mit mehreren Ansichten

12.1	Storyboard und Controller-Klassen verbinden	409
12.2	Ansichten durch Segues verbinden	411
	Welcher Segue-Typ ist der richtige?	412
	Zurück an den Start mit Unwind	413
12.3	Segues mit Datenübertragung	415
	Segue-Code für »View 1«	417
	Segue-Code für »View 2«	418
	Segues per Code auslösen	419
12.4	Tastatureingaben mit Delegation verarbeiten	420
	Beispiel	421
12.5	Navigation-Controller	422
	Funktionsweise	422
	Einstellungen	423
	Steuerung per Code	424
	Beispiel	425

12.6	Tab-Bar-Controller	426
	Tab-Bar-Items	427
	Kombination aus Tab-Bar- und Navigation-Controller	429
	Programmierung	430
12.7	Bild-Management in Images.xcasset	432
	Zugriff auf Images.xcasset per Code	434
	App-lcon	434
	App-Icon	434
	App-Icon	434

435

# 13 GPS- und Kompassfunktionen

13.1	Hello MapView!	435
	MapKit-Framework	435
	Um Erlaubnis fragen	436
	Info.plist-Einstellungen	437
	Erste Tests	438
	Kartenfunktionen im iOS-Simulator	439
13.2	Wegstrecke aufzeichnen	440
	Programmaufbau und Auto Layout	440
	Die ViewController-Klasse	441
	Initialisierung in viewDidLoad	442
	locationManager-Delegate	443
	Die mapView-Methode	446
	Erweiterungsmöglichkeiten	446
13.3	Kompassfunktionen	447
	Kompasskalibrierung	448
	Grafische Darstellung eines Kompasses	449
13.4	Eigene Steuerelemente mit Grafikfunktionen	449
	Eine Klasse für ein neues Steuerelement	449
	Grafikprogrammierung	450
	Das Steuerelement verwenden	451
	Eine richtige CompassView	453
	Automatischer Redraw bei Größenänderung	455
	Kompassnadel einstellen	456
	Den Kompass an die Ausrichtung des Geräts anpassen	456
	Xcode-Integration mit IBDesignable und IBInspectable	458

### 14 To-do-Listen

14.1	Popups	461
	Hello Popup!	461
	Popups auch auf dem iPhone	463
	Größe des Popups einstellen	464
	Popup-Richtung festlegen	466
	Popups per Code anzeigen und entfernen	467
	Unwind für Popups	469
	dismissViewControllerAnimated-Methode	470
14.2	Ja-Nein-Dialoge (UIAlertController)	470
14.3	Listen (UITableView)	472
	Hello UITableView!	473
	Listenzellen mit Bild und Zusatzinformationen	478
14.4	Individuelle Gestaltung der Listenelemente (UITableViewCell)	481
	Detailansicht zu Listeneinträgen	484
14.5	Veränderliche Listen	487
	Die Methode reloadData	487
	Edit-Modus	488
	Gestaltung der Benutzeroberfläche	488
	Beispiele	488
14.6	To-do-App	489
	Auto Layout	489
	Datenmodell	490
	Outlets und Initialisierung	491
	Button- und Gesture-Methoden	493
	Popup-Dialog anzeigen	494
	Listeneintrag hinzufügen oder ändern	495
	DataSource-Methoden	496
	Popup-View-Controller	498

# 15 Schatzsuche

15.1	Aufbau der App	501
	Aufbau und Storyboard	503
	Layout-Regeln	504
	Funktion zur Darstellung geografischer Daten	504
	Projekteinstellungen	505

15.2	Datenmodell	505
	Basisklasse und Protokolle	506
	description-Eigenschaft	506
	Das Protokoll NSCoding	507
	Array speichern und wieder einlesen (NSKeyedArchiver)	507
15.3	Location Manager selbst gemacht	509
	Die Init-Funktion	510
	Kommunikation über das Notification Center	510
15.4	Steuerelement zur Richtungsanzeige (UIBezierPath)	513
15.5	Hauptansicht mit Listenfeld	514
	Outlets, Eigenschaften und Initialisierung	514
	DataSource-Anbindung	515
	Segues vom und zum Speichern-Popup, neuen Eintrag speichern	516
	Segue vom und zum Detaildialog	518
15.6	Popup-Dialog zum Speichern	519
15.7	Detailansicht mit Richtungspfeil	520
	Auto Layout	521
	Initialisierung der Controller-Klasse	522
	Abstand und Richtung zum Zielpunkt errechnen	523
	Listeneintrag löschen bzw. ändern	524

# 16 Währungskalkulator

16.1	App-Überblick	527
	Storyboard und Klassen	529
	Auto Layout in der Umrechnungsansicht	529
	Auto Layout in der Einstellungsansicht	531
	Layout-Variante mit Stack-Views	532
	Bildkataloge	534
	Erweiterungsmöglichkeiten	534
16.2	XML-Dokumente lesen	535
	XML-Datei herunterladen	536
	Die SWXMLHash-Bibliothek	538
16.3	Das Datenmodell der App	540
	Init-Funktion	540
	Wechselkurse im Cache-Verzeichnis speichern	541
	Kursumrechnung	543
	Länderkürzel aus Währungskürzeln extrahieren	543

16.4	Umrechnungsansicht	
	Property Observer für die Währungskürzel	544
	Initialisierung in viewDidLoad	545
	Das CurCalc-Objekt initialisieren und das Datum der Kurse anzeigen	546
	Ungültige Tastatureingaben vermeiden	547
	Tap Gesture Recognizer	548
	Währungsumrechnung bei der Texteingabe	549
16.5	Einstellungsansicht	550
	Picker-Views (UIPickerView-Klasse)	550
	Outlets, Eigenschaften und viewDidLoad	550
	Picker-View mit Daten füllen	552
	Auswahl eines Picker-View-Elements	554
16.6	Startansicht (Launch Screen)	555
16.7	App-lcon	556
	App-Name	557
16.8	Internationalisierung und Lokalisierung	557
16.9	App im App Store einreichen	558
	App-Store-Regeln	559
	Bundle-ID (Xcode)	560
	App-ID erzeugen (Apple Developer)	560
	App einrichten (iTunes Connect)	561
	App-Daten ausfüllen (iTunes Connect)	563
	Distribution Provisioning Profile erzeugen (Apple Developer)	564
	App-Upload (Xcode)	566
	Warten auf das Okay von Apple	566

# 17 Fünf Gewinnt

17.1	Einfache Animationen	569
	Hello World!	569
	Fade-In-Effekt	571
	Steuerelemente animiert erscheinen und verschwinden lassen	572
17.2	Die App »5 Gewinnt«	574
	Hintergründe zum Spiel	576
	Programmaufbau	576
	Storyboard und Auto-Layout-Regeln	576
17.3	Enumerationen und globale Funktionen (Globals.swift)	577
	Feld- und Spielstatus (Piece und GameStatus)	577
	Spielbrettgrößen (BoardSize)	578

	2D-Arrays erzeugen	579
	Farben aufhellen bzw. abdunkeln	580
	Rechteck rund um Mittelpunkt erzeugen	580
	Code verzögert ausführen	581
17.4	Die Spiellogik (FiveWins.swift)	582
	Spielfeld speichern	582
	Zug ausführen und rückgängig machen	583
	Sieg-Test	584
	Der Spielalgorithmus	586
	Den Wert einer Linie berechnen	588
	Wert aller Linien berechnen	589
	Den Wert eines Spielfelds berechnen	590
	Den besten Zug auswählen	592
17.5	Darstellung des Spielbretts und der Steine (BoardView.swift)	593
	Eigenschaften	593
	Reset	596
	Spielbrett zeichnen	596
	Zug ausführen	597
	Einen Spielstein als (animierte) View darstellen	598
	Steine neu positionieren	600
	Benutzereingaben feststellen und weiterleiten	601
17.6	Steuerung des Spielablaufs (ViewController.swift)	603
	Ein neues Spiel starten	604
	Warten auf den nächsten Zug	605
	Aktualisierung des Labels und der Statusvariablen	606
	Undo	607
	Popup-Aufruf	607
17.7	Der Popup-Dialog (PopupVC.swift)	609
	Neues Spiel starten, Zug rückgängig machen	610
	Einstellungen ändern	610
	Dialog schließen unter iOS 7	611
17.8	Erweiterungsmöglichkeiten	611
	Spielstand automatisch speichern	611
	Spielstärke	612
	Animationen und Audio-Effekte	612
	Optische Gestaltung	613
	Geld verdienen	613

# TEIL III OS X

18	Hello OS-X-World!	617
18.1	Von iOS zu OS X	617
	Gemeinsamkeiten	617
	Unterschiede	618
	Window- versus View-Controller	618
	Storyboards	619
18.2	Lottozahlengenerator (Storyboard-Variante)	620
	Projekt einrichten	621
	Gestaltung der Benutzeroberfläche	621
	Effizient arbeiten mit Kopieren und Einfügen	622
	Fenstergröße und Fenstertitel einstellen	623
	viewDidLoad und representedObject	625
	Outlets und Actions	626
	Die Lotto-Klasse	627
	Die View-Controller-Klasse	627
	Programmende	629
18.3	Lottozahlengenerator (XIB/AppDelegate-Variante)	630
	XIB-Dateien	630
	Organisation des Projekts	630
	Die AppDelegate-Klasse	631
	Eigener Code	632
	Programmende	634
18.4	Lottozahlengenerator (XIB/WindowController-Variante)	635
	Window-Controller mit XIB-Datei	635
	Das Fenster in der AppDelegate-Klasse anzeigen	636
	Windows-Controller-Code	637
	Programmende	638
18.5	Lottozahlengenerator (XIB/ViewController-Variante)	639
	Das Fenster mit dem View-Controller verbinden	640
	Controller-Code	641

## 19 OS-X-Grundlagen

19.1	Programme mit mehreren Fenstern	643
	Segues	645
	Datenübergabe mit der Methode prepareForSegue	646
	Fenstergröße fixieren	648

	Window-Eigenschaften des Ziel-Controllers einstellen	649
	Ansichten/Fenster schließen	649
	Segues per Code ausführen	650
	Fenster per Code erzeugen	650
19.2	Tab-View-Controller	651
	Storyboard und Tab-View-Controller-Einstellungen	653
	Dialogblattgröße	654
	Segues	654
	Splitter-Steuerelement	655
	Klassen	655
	Application Defaults mit den User-Defaults verbinden (AppDelegate.swift)	655
	Textgröße aus den User-Defaults lesen (ViewController.swift)	657
	Einstellungen ändern (SettingsGeneralVC.swift)	658
	User-Defaults-Interna	659
19.3	Standarddialoge	660
	Nachrichten anzeigen und Ja/Nein-Entscheidungen treffen	661
	Datei- und Verzeichnisauswahl	661
	Schrift einstellen	662
	Farbe einstellen	663
19.4	Maus	664
19.4	Maus Mausereignisse	664 665
19.4	Maus Mausereignisse Koordinatensysteme, Bounds und Frames	664 665 665
19.4	Maus Mausereignisse Koordinatensysteme, Bounds und Frames Mausposition ergründen	664 665 665 666
19.4	Maus Mausereignisse Koordinatensysteme, Bounds und Frames Mausposition ergründen Statustasten	664 665 665 666 667
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm	664 665 665 666 667 667
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse	664 665 665 666 667 667 668
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode	664 665 665 666 667 667 668 669
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode	664 665 665 666 667 667 668 669 671
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur	664 665 666 667 667 668 669 671 672
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur         Die NSResponder-Klasse	664 665 666 667 667 668 669 671 672 673
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Die NSResponder-Klasse         Tastatur         Dia NSResponder-Klasse	664 665 666 667 667 668 669 671 672 673 673
19.4	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Die NSResponder-Klasse         Tastaturereignisse         Beispielprogramm	664 665 666 667 667 668 669 671 672 673 674 675
19.4 19.5 19.5	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur         Die NSResponder-Klasse         Beispielprogramm         Mausereignisse         Beispielprogramm	664 665 666 667 667 668 669 671 672 673 674 675 678
19.4 19.5 19.6	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur         Die NSResponder-Klasse         Beispielprogramm         Mause         Die NSResponder-Klasse         Die Responder-Kette	664 665 666 667 667 668 669 671 672 673 674 675 678 678
19.4 19.5 19.6	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur         Die NSResponder-Klasse         Beispielprogramm         Mause         Die NSResponder-Klasse         Beispielprogramm         Menüs         Die Responder-Kette         Gestaltung der Menüleiste	664 665 666 667 668 669 671 672 673 674 675 678 678 678 678
19.4 19.5 19.6	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Die NSResponder-Klasse         Tastatur         Die NSResponder-Klasse         Beispielprogramm         Gestaltung der Menüleiste         Responder-Aktionen	664 665 666 667 667 668 669 671 672 673 674 675 678 678 678 680 681
19.4 19.5 19.6	Maus         Mausereignisse         Koordinatensysteme, Bounds und Frames         Mausposition ergründen         Statustasten         Beispielprogramm         Die MyView-Klasse         Die drawRect-Methode         Die mouseDown-Methode         Tastatur         Die NSResponder-Klasse         Beispielprogramm         Mause         Menüs         Die Responder-Kette         Gestaltung der Menüleiste         Responder-Aktionen         Menüaktionen in der AppDelegate-Klasse	664 665 666 667 667 668 669 671 672 673 674 675 678 678 678 678 680 681 681

	Veränderung von Menüeinträgen per Code	684
	Kontextmenüs	685
19.7	Programme ohne Menü	686
	Menubar-Apps	687
	Die AppDelegate-Klasse	687
	View-Controller	689
19.8	Bindings	689
	Hello Bindings!	690
	Sonderfälle	691

### 20 Icon-Resizer

20.1	Tabellen (NSTableView)	693
	Hello NSTableView!	694
	Table-View mit eigenen Views	697
	Programmaufbau und Country-Klasse	699
	Table-View-Code	700
	Die Tabelle mit Daten füllen	700
	Tabelle sortieren	703
	Auswahl einer Zeile	704
20.2	Drag & Drop	704
	Drag-Operationen empfangen (NSDraggingDestination)	705
	Drag-Operationen initiieren (NSDraggingSource)	706
	Beispielprogramm	706
	Projektaufbau	707
	View-Controller	707
	Die MyView-Klasse	708
	Drag & Drop initiieren	710
	Drag & Drop-Empfang zulassen	713
	Drag & Drop-Empfang verarbeiten	716
20.3	Icon-Resizer	717
	Programmaufbau	719
	Das Split-View-Steuerelement	719
	Layoutregeln für das Hauptfenster	720
	Popup-Menü	721
	Erweiterungsmöglichkeiten	722
20.4	Arbeiten mit Bitmaps (IconSize-Struktur)	722
	Enumerationen	722
	IconSize-Struktur	723
	Initialisierung von IconSize-Arrays	723

	Bitmaps skalieren	724
	Bitmaps im PNG-Format speichern	727
20.5	Hauptfenster (ViewController.swift)	728
	viewDidLoad mit dem Aufruf von unregisterDraggedTypes	729
	Popup- und Speicher-Buttons, Programmende	730
	Split-View-Delegation	732
	Table-View-Datenquelle	732
20.6	Drag & Drop-Quelle für Icons (IconCellView.swift)	735
20.7	Drag & Drop-Empfänger für Icons (OriginalIconView.swift)	736
	Drag & Drop einer Bilddatei empfangen	737
	Dateiauswahldialog für die Bilddatei	738
20.8	Popup-Menü (IconChoiceVC.swift)	739
20.9	Temporäres Verzeichnis erstellen und löschen (AppDelegate.swift)	740
	Ein eigenes temporäres Verzeichnis einstellen	741
	Temporäres Verzeichnis löschen	742
20.10	OS-X-Programme weitergeben	742
	Programme signieren und archivieren	743
	Programme in einem DMG-Image verpacken	745
Index		747

# Vorwort

Als im Juni 2014 das alljährliche Apple-Entwicklertreffen stattfand, übertrafen sich die Medien wie üblich mit Spekulationen darüber, welche Produkte Apple diesmal aus dem Hut zaubern würde: die damals noch sagenumwobene Apple Watch? Ein neues iPhone? Doch Apple konzentrierte sich auf die Software und präsentierte – selbst für Insider überraschend – eine neue Programmiersprache: Swift.

In ersten Kommentaren konnten selbst Apple-Fans Ihre Skepsis nicht verbergen: Brauchen wir wirklich eine neue Programmiersprache? Doch je mehr Details Apple auf der World Wide Developers Conference (WWDC) verriet, desto größer wurde die Begeisterung der teilnehmenden Entwickler und der Fachpresse. Swift war zum Zeitpunkt der Ankündigung bereits ein nahezu fertiges Produkt, an dem Apple im Geheimen seit mehreren Jahren gearbeitet hatte.

Mit der Freigabe von Swift 1.0 blieb Apple aber nicht stehen. Jeweils im Abstand weniger Monate folgten die Versionen 1.1, 1.2 und zur WWDC 2015 auch schon die Version 2.0, auf der dieses Buch basiert. Damit hat Apple Swift im ersten Jahr stärker verändert als Objective C in einem ganzen Jahrzehnt!

#### Warum Swift?

Swift ist für Apple ein Befreiungsschlag: Objective C dient dem Apple-Universum seit vielen Jahren als Fundament. Das ändert aber nichts daran, dass Objective C eine Programmiersprache aus den 1980er-Jahren ist, die in keinerlei Hinsicht mit modernen Programmiersprachen mithalten kann.

Swift ist dagegen ein sauberer Neuanfang. Bei der Vorstellung wurde Swift auch *Objective C without the C* genannt. Natürlich ist Swift von Objective C beeinflusst – schließlich musste Swift kompatibel zu den Bibliotheken für iOS und OS X sein. Neben eigenen Ideen greift Swift aber auch Konzepte von C#, Haskell, Java, Python und anderen Programmiersprachen auf. Daraus ergeben sich mehrere Vorteile:

- Swift zählt zu den modernsten Programmiersprachen, die es momentan gibt.
- ► Code lässt sich in Swift syntaktisch eleganter formulieren als in Objective C.
- ► Der resultierende Code ist besser lesbar und wartbar.
- Swift ist für Programmierer, die schon Erfahrung mit anderen modernen Sprachen gesammelt haben, wesentlich leichter zu erlernen als Objective C. Vorhandenes Know-how lässt sich einfacher auf Swift als auf Objective C übertragen.

Wer mit Apple-Produkten zu tun hat, erwartet Perfektion bis ins letzte Detail. Bei aller Euphorie für Swift will ich Ihnen nicht verschweigen, dass dies für Swift momentan nicht ganz zutrifft:

- ► Die Integration in Xcode ist gut, aber nicht perfekt. Beispielsweise fehlen in Xcode noch Refactoring-Funktionen für Swift.
- Trotz der Fertigstellung von Version 2.0 ist zu erwarten, dass Apple weiter intensiv an Swift feilen wird. So wünschenswert jede Verbesserung ist, so ärgerlich sind inkompatible Neuerungen, wenn Sie gerade an einer App arbeiten.

Allen Kinderkrankheiten zum Trotz vereinfacht Swift den Einstieg in die App-Entwicklung enorm. Es ist zu erwarten, dass Swift in wenigen Jahren *die* Programmiersprache der Apple-Welt sein wird und Objective C in dieser Rolle ablöst. In naher Zukunft wird an Swift also kein Weg vorbeiführen.

#### Was bietet dieses Buch?

Dieses Buch vermittelt einen kompakten Einstieg in die Programmiersprache Swift (Version 2 / Xcode 7). Während es im ersten Teil des Buchs primär um die Syntax geht, demonstrieren die weiteren Kapitel die Entwicklung von Apps für iOS und OS X. Sie lernen Swift also in Theorie und Praxis kennen, wobei die Praxis klar im Vordergrund steht. Nebenbei gibt das Buch Ihnen auch einen Einstieg in den Umgang mit Xcode, in die Anwendung elementarer Cocoa- bzw. Cocoa-Touch-Klassen sowie in grundlegende Techniken und Konzepte der App-Entwicklung.

Um von diesem Buch maximal zu profitieren, benötigen Sie weder Vorkenntnisse in Xcode noch in der App-Entwicklung. Ich setze aber voraus, dass Sie bereits Erfahrungen mit einer Programmiersprache gesammelt haben. Ich erkläre Ihnen in diesem Buch also, wie Sie in Swift mit Variablen umgehen, Schleifen programmieren und Klassen entwickeln, aber nicht, was Variablen sind, wozu Schleifen dienen und warum Klassen das Fundament objektorientierter Programmierung sind. So kann ich Swift kompakt und ohne viel Overhead beschreiben und den weiteren Schwerpunkt auf die konkrete Anwendung legen.

Wenn Sie in die Welt der App-Entwicklung für iOS oder OS X eintauchen und dabei auf eine der modernsten verfügbaren Programmiersprachen setzen möchten, dann schafft dieses Buch ein solides Fundament. Bei Ihrer Reise durch die neue Welt der Swift-Programmierung wünsche ich Ihnen viel Spaß und Erfolg!

#### Michael Kofler (https://kofler.info)

PS: Ausdrücklich bedanken möchte ich mich bei Clemens Wagner, der eine Menge Zeit investiert hat, um das Manuskript zu lesen. Viele Korrekturen und Verbesserungen gehen auf sein Konto.

# Kapitel 2 Operatoren

Im Ausdruck a = b + c gelten die Zeichen = und + als Operatoren. Dieses Kapitel stellt Ihnen alle Swift-Operatoren vor - von den simplen Operatoren für die Grundrechenarten bis hin zu Swift-Spezialitäten wie dem Range-Operator n1...n2.

#### Leerzeichen vor oder nach Operatoren

Normalerweise ist es nicht notwendig, vor oder nach einem Operator ein Leerzeichen zu schreiben. x=x+7 funktioniert genauso gut wie x = x + 7. Aber wie so oft bestätigen Ausnahmen die Regel: Swift kennt bereits standardmäßig ungewöhnlich viele Operatoren, und wenige Zeilen Code reichen aus, um weitere zu definieren.

Das führt mitunter dazu, dass der Compiler nicht eindeutig erkennen kann, wo der eine Operator endet und wo der nächste beginnt. Spätestens dann *müssen* Sie ein Leerzeichen setzen – und dann sollten Sie es vor *und* nach dem Operator setzen! Andernfalls glaubt der Compiler nämlich, Sie wollten ihn explizit darauf hinweisen, dass es sich um einen Präfix- oder Postfix-Operator handelt. Im Detail sind diese Feinheiten in »The Swift Programming Language« dokumentiert:

https://developer.apple.com/library/ios/documentation/Swift/Conceptual/ Swift\_Programming\_Language/LexicalStructure.html (beim Punkt »Operators»)

### 2.1 Zuweisungs- und Rechenoperatoren

Dieser Abschnitt erläutert die zahlreichen Rechen- und Zuweisungsoperatoren. Swift kennt dabei auch Mischformen. Beispielsweise entspricht x+=3 der Anweisung x=x+3.

#### **Einfache Zuweisung**

Der Zuweisungsoperator = speichert speichert in einer Variablen oder Konstanten das Ergebnis des Ausdrucks:

variable = ausdruck

Vor der ersten Zuweisung an eine Variable bzw. Konstante muss diese mit var bzw. let als solche deklariert werden.

```
var i = 17
i = i * 2
let pi = 3.1415927
```

Nicht zulässig sind Mehrfachzuweisungen in der Art a=b=3. Dafür können mehrere Variablen als Tupel geschrieben und gleichzeitig verändert werden:

var (a, b, c) = (1, 7, 12)

Das funktioniert auch bei komplexeren Ausdrücken:

var (\_, a, (b, c)) = (1, 2, ("x", "y"))
// entspricht var a=2; var b="x"; var c="y"

Der Unterstrich \_ ist hier ein *Wildcard Pattern*. Es trifft auf jeden Ausdruck und verhindert im obigen Beispiel dessen weitere Verarbeitung.

#### Wert- versus Referenztypen

Swift unterscheidet bei Zuweisungen zwischen zwei grundlegenden Datentypen:

- Werttypen (Value Types): Dazu zählen Zahlen, Zeichenketten, Tupel, Arrays, Dictionaries sowie struct- und enum-Daten. Bei einer Zuweisung werden die Daten kopiert. Die ursprünglichen Daten und die Kopie sind vollkommen unabhängig voneinander.
- Referenztypen: Objekte, also Instanzen von Klassen, sind Referenztypen. Bei einer Zuweisung wird eine weitere Referenz auf die bereits vorhandenen Daten erstellt. Es zeigen nun zwei (oder mehr) Variablen auf dieselben Daten.

Die folgenden beiden Beispiele verdeutlichen den Unterschied. Im ersten Beispiel werden in x und y ganze Zahlen gespeichert, also Werttypen:

```
var x = 3
var y = x
x=4
print(y) // y ist unverändert 3
```

Für das zweite Beispiel definieren wir zuerst die Mini-Klasse SimpleClass. In a wird eine Instanz dieser Klasse gespeichert. Bei der Zuweisung b = a wird die Instanz *nicht kopiert*, stattdessen verweist nun b auf dasselbe Objekt wie a. (In C würde man sagen, a und b sind Zeiger.) Jede Veränderung des Objekts betrifft deswegen a gleichermaßen wie b:

```
class SimpleClass {
  var data=0
}
var a = SimpleClass()
var b = a // a und b zeigen auf die gleichen Daten
a.data = 17
print(b.data) // deswegen ist auch b.data 17
```

#### Arrays, Dictionarys und Zeichenketten sind Werttypen!

Die Unterscheidung zwischen Wert- und Referenztypen gibt es bei den meisten Programmiersprachen. Beachten Sie aber, dass Arrays und Zeichenketten in Swift Werttypen sind und nicht, wie in vielen anderen Sprachen, Referenztypen!

#### **Elementare Rechenoperatoren**

Die meisten Rechenoperatoren sind aus dem täglichen Leben bekannt (siehe <u>Tabelle 2.1</u>). Der Operator % liefert den Rest einer ganzzahligen Division: 13 % 5 ergibt also 3, da 2  $\ast$  5 + 3 = 13. Bei Fließkommazahlen wird der Rest zum ganzzahligen Ergebnis ermittelt. 1.0 % 0.4 ergibt 0.2, da 2  $\ast$  0.4 + 0.2 = 1.0

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Restwert einer ganzzahligen Division
&+	Integer-Addition ohne Überlaufkontrolle
8-	Integer-Subtraktion ohne Überlaufkontrolle
&*	Integer-Multiplikation ohne Überlaufkontrolle

Tabelle 2.1 Rechenoperatoren

Alle Operatoren setzen voraus, dass links und rechts von ihnen jeweils gleichartige Datentypen verwendet werden! Im Gegensatz zu anderen Programmiersprachen erfolgen Typumwandlungen nicht automatisch.

Wenn Sie die Summe von a plus b ausrechnen möchten, müssen Sie explizit den Datentyp einer der beiden Operatoren anpassen. Int rundet dabei immer ab, d. h., aus 1.7 wird 1.

var c1 = a + Int(b) // c1 = 4 var c2 = Double(a) + b // c2 = 4.7

#### **Division durch null**

Bei einer Fließkommadivision durch 0.0 lautet das Ergebnis einfach +infinity bzw. -infinity. Wenn Sie hingegen mit Integerzahlen arbeiten, löst eine Division durch 0 einen Fehler aus.

Eine Besonderheit von Swift sind die Operatoren &+, &- und &+: Sie führen die Grundrechenarten für Integerzahlen ohne Überlaufkontrolle durch. Das ermöglicht die Programmierung besonders effizienter Algorithmen. Sollte allerdings doch ein Überlauf eintreten, dann ist das Ergebnis falsch!

Swift kennt keinen Operator zum Potenzieren. a<sup>b</sup> müssen Sie unter Zuhilfenahme der Funktion pow berechnen. Diese Funktion ist in der Foundation-Bibliothek definiert. Sie steht nur zur Verfügung, wenn Ihr Code import Foundation enthält oder eine andere Bibliothek importiert, die auf die Foundation zurückgreift. Das trifft unter anderem für Cocoa und UIKit zu.

```
var a = 7.0
var b = pow(a, 3.0) // 7 * 7 * 7 = 343.0
```

#### Zeichenketten aneinanderfügen

Der Operator + addiert nicht nur zwei Zahlen, sondern fügt auch Zeichenketten aneinander:

```
var s1 = "Hello"
var s2 = "World!"
var hw = s1 + " " + s2 // "Hello World!"
```

#### **Inkrement und Dekrement**

Wie viele andere Programmiersprachen kennt Swift Inkrement- und Dekrement-Operatoren ++ und --. Sie vergrößern bzw. verkleinern eine numerische Variable um 1. Die Operatoren dürfen auch auf Double-Variablen angewendet werden.

Diese Inkrement- und Dekrement-Operatoren können wahlweise nach oder vor dem Variablennamen angegeben werden (Postfix- bzw. Präfix-Notation). Wie das folgende Beispiel beweist, hat dies zwar keinen Einfluss auf die betroffene Variable – diese wird in jedem Fall um 1 verändert; allerdings wird bei der Postfix-Notation zuerst der ursprüngliche Wert weiterverarbeitet und die Variable erst später verändert. Bei der Präfix-Notation wird die Variable hingegen sofort geändert; der neue Wert wird dann für den Ausdruck ausgewertet.

```
var daten = [0, 1, 2, 3, 4, 5]
print(daten[2]) // Ausgabe 2
var n=3 // Postfix-Inkrement
print(daten[n++]) // Ausgabe 3
print(n) // Ausgabe 4
n=3 // Präfix-Inkrement
print(daten[++n]) // Ausgabe 4
print(n) // Ausgabe 4
```

#### **Rechnen mit Bits**

Die bitweisen Operatoren &, |, ^ und ~ (AND, OR, XOR und NOT) verarbeiten ganze Zahlen bitweise. Das folgende Beispiel verwendet die Schreibweise 0b zur Kennzeichnung binärer Zahlen. String mit dem zusätzlichen Parameter radix:2 wandelt ganze Zahlen in eine Zeichenkette in binärer Darstellung um.

let a = 0b11100	// Wert 28
let b = 0b01111	// Wert 15
let result = a & b	// Wert 12
<pre>print(String(result, radix:2))</pre>	// Ausgabe 1100

>> verschiebt die Bits einer Zahl um n Bits nach links (entspricht einer Division durch 2<sup>n</sup>), << verschiebt entsprechend nach rechts (entspricht einer Multiplikation mit 2<sup>n</sup>).
>>> funktioniert wie >>, betrachtet die Zahl aber so, als wäre sie vorzeichenlos.

```
let a = 16
let b = a << 2 // entspricht b=a*4, Ergebnis 64
let c = a >> 1 // entspricht c=a/2, Ergebnis 8
```

Wenn Sie Daten bitweise verarbeiten, ist es oft zweckmäßig, anstelle gewöhnlicher Integer-Zahlen explizit Datentypen ohne Vorzeichen zu verwenden, z.B. UInt32 oder UInt16. Das folgende Beispiel verwendet 0x zur Kennzeichnung hexadezimaler Zahlen.

```
let rgb:UInt32 = 0x336688
let red:UInt8 = UInt8( (rgb & 0xff0000) >> 16 )
```

#### Kombinierte Rechen- und Zuweisungsoperationen

Alle bereits erwähnten Rechenoperatoren sowie die logischen Operatoren & und || können mit einer Zuweisung kombiniert werden. Dazu muss dem Operator das Zeichen = folgen. Details zu den logischen Operatoren folgen im nächsten Abschnitt.

```
X + = V
         // entspricht x = x + y
         // entspricht x = x - y
x - = y
X * = V
         // entspricht x = x * y
x / = y
         // entspricht x = x / y
x%=y
         // entspricht x = x \% y
        // entspricht x = x << y</pre>
x < < = y
x > > = y
         // entspricht x = x >> y
        // entspricht x = x & y
x&=y
        // entspricht x = x \&\& y
x&&=y
x | = y
         // entspricht x = x | y
        // entspricht x = x || y
x | | = y
x ^ = y
        // entspricht x = x^{y}
```

### 2.2 Vergleichsoperatoren und logische Operatoren

Um Bedingungen für Schleifen oder Verzweigungen zu formulieren, müssen Sie Variablen vergleichen und oft mehrere Vergleiche miteinander kombinieren. Dieser Abschnitt stellt Ihnen die dazu erforderlichen Operatoren vor.

#### Vergleichsoperatoren

Die Vergleichsoperatoren ==, != (ungleich), <, <= sowie > und >= können gleichermaßen für Zahlen und für Zeichenketten eingesetzt werden. Wie bei anderen Operatoren ist es wichtig, dass auf beiden Seiten des Operators der gleiche Datentyp verwendet wird; Sie können also nicht eine ganze Zahl mit einer Fließkommazahl vergleichen!

```
1 == 2 // false
1 < 2 // true
"abc" == "abc" // true
"abc" == "Abc" // false
```

Zeichenketten gelten dann als gleich, wenn auch die Groß- und Kleinschreibung übereinstimmt. Etwas schwieriger ist die Interpretation von *größer* und *kleiner*. Grundsätzlich gelten Großbuchstaben als *kleiner* als Kleinbuchstaben, d. h., sie werden beim Sortieren vorne eingereiht. Internationale Zeichen werden auf der Basis der *Unicode Normalform D* verglichen. Die deutschen Buchstaben ä, ö oder ü werden dabei wie eine Kombination aus zwei Zeichen betrachtet, beispielsweise ä = a". Somit gilt:

"A" < "a" // true "a" < "ä" // true "ä" < "b" // true

Mehr Details zur Sortierordnung von Zeichenketten und zu Möglichkeiten, diese zu beeinflussen, folgen in <u>Abschnitt 3.3</u>, »Zeichenketten«.

#### == versus ===

Zum Vergleich von Objekten kennt Swift neben == und != auch die Varianten === und !==. Dabei testet a===b, ob die beiden Variablen a und b auf dieselbe Instanz einer Klasse zeigen. Hingegen überprüft a==b, ob a und b zwei Objekte mit übereinstimmenden Daten sind. Das ist nicht das Gleiche! Es ist ja durchaus möglich, dass zwei unterschiedliche Objekte die dieselben Daten enthalten.

#### Einschränkungen

Die Operatoren === und !== können nur auf Referenztypen angewendet werden, nicht auf Werttypen (wie Zahlen, Zeichenketten, Arrays, Dictionaries sowie sonstige Strukturen).

Umgekehrt können die Operatoren == und != bei selbst definierten Klassen nur verwendet werden, wenn Sie für diese Klassen den Operator == selbst implementieren (Protokoll Equatable, siehe <u>Abschnitt 8.4</u>, »Standardprotokolle«).

Die folgenden Zeilen definieren zuerst die Klasse Pt zur Speicherung eines Koordinatenpunkts und dann den Operator == zum Vergleich zweier Pt-Objekte. Damit ist das Beispiel gleich auch ein Vorgriff auf die Definition eigener Operatoren.

```
// Pt-Klasse
class Pt {
   var x:Double, y:Double
   // Init-Funktion
   init(x:Double, y:Double){
      self.x=x
      self.y=y
   }
}
```

```
// Operator zum Vergleich von zwei Pt-Objekten
func ==(left:Pt, right:Pt) -> Bool {
    return left.x==right.x && left.y==right.y
}
// == versus ==
var p1 = Pt(x: 1.0, y: 2.0)
var p2 = Pt(x: 1.0, y: 2.0)
p1 == p2 // true, weil die Objekte dieselben Daten enthalten
p1 === p2 // false, weil es unterschiedliche Objekte sind
```

#### Vergleiche mit ~=

Swift kennt mit ~= einen weiteren Vergleichsoperator mit recht wenigen Funktionen:

- ► Zwei Ausdrücke des gleichen Typs werden wie mit == verglichen.
- Außerdem kann getestet werden, ob eine ganze Zahl in einem durch den Range-Operator formulierter Zahlenbereich enthalten ist.

-2...2 ~= 1 // true -2...2 ~= -2 // true -2...2 ~= 2 // true -2...2 ~= 4 // false

Achten Sie darauf, dass Sie zuerst den Bereich und dann den Vergleichswert angeben müssen. Wenn Sie die Reihenfolge vertauschen, funktioniert der Operator nicht. Details zu Range-Operatoren folgen gleich.

Analog kann auch in switch-Ausdrücken mit case überprüft werden kann, ob sich ein ganzzahliger Ausdruck in einem vorgegebenen Bereich befindet:

```
let n = 12
switch n {
case (1...10):
    print("Zahl zwischen 1 und 10")
case(11...20):
    print("Zahl zwischen 11 und 20")
default:
    print("Andere Zahl")
}
```

#### Datentyp-Vergleich (»is«)

Mit dem Operator is testen Sie, ob eine Variable einem bestimmten Typ entspricht:

```
func f(obj:Any) {
  if obj is UInt32 {
    print("Datentyp UInt32")
    ...
  }
}
```

#### Casting-Operator (»as«)

Mit dem Operator as wandeln Sie, sofern möglich, einen Datentyp in einen anderen um. Der Operator hat in Swift drei Erscheinungsformen:

- as: In dieser Form eignet sich as nur, wenn der Compiler erkennen kann, dass die Umwandlung gefahrlos möglich ist. Das trifft auf alle Upcasts zu, also auf Umwandlungen in Instanzen einer übergeordneten Klasse (siehe <u>Abschnitt 8.1</u>), außerdem bei manchen Literalen (z. B. 12 as Float).
- as?: Der Downcast-Operator as? typ stellt vor der Typkonvertierung sicher, dass diese überhaupt möglich ist. Wenn das nicht der Fall ist, lautet das Ergebnis nil. Es tritt kein Fehler auf.

Mit if let varname = ausdruck as? typ können Sie den Typtest mit einer Zuweisung kombinieren. Das funktioniert gleichermaßen für Konstanten (let wie im folgenden Beispiel) wie auch für Variablen (var):

```
func f(obj:Any) {
    if let myint = obj as? UInt32 {
        // myint hat den Datentyp UInt32
        ...
    } else {
        print("falscher Datentyp")
    }
}
```

► as!: Mit einem nachgestellten Ausrufezeichen wird die Konvertierung auf jeden Fall versucht. Dabei kann es zu einem Fehler kommen, wenn der Datentyp nicht passt. Insofern ist diese Variante zumeist nur zweckmäßig, wenn die Typenüberprüfung im Voraus erfolgt.

```
let obj:Any = 123
if obj is UInt32 {
    // wird nicht ausgeführt, weil obj eine Int-Instanz enthält
    var myint = obj as! UInt32
}
```

#### Upcasts und Downcasts

Ein Grundprinzip der objektorientierten Programmierung ist die Vererbung. Damit können Klassen die Merkmale einer Basisklasse übernehmen und diese erweitern oder verändern. Ein Objekt einer abgeleiteten Klasse (im Klassendiagramm unten dargestellt) kann immer wie eines der Basisklasse verwendet werden (im Klassendiagramm oben). Das nennt man einen (impliziten) Upcast, also eine Umwandlung in der Klassenhierarchie nach oben.

Eine Konvertierung in die umgekehrte Richtung ist ein Downcast. Dieser funktioniert nur, wenn eine Variable vom Typ der Basisklasse tatsächlich ein Objekt der erforderlichen abgeleiteten Klasse enthält. Mehr Details zu diesem Thema finden Sie in Abschnitt 8.1, »Vererbung«.

#### Logische Operatoren

Logische Operatoren kombinieren Wahrheitswerte. *Wahr UND Wahr* liefert wieder *Wahr, Wahr UND Falsch* ergibt hingegen *Falsch*. In Swift gibt es wie in den meisten anderen Programmiersprachen die drei logischen Operatoren ! (Nicht), && (Und) sowie || (Oder):

&& und || führen eine sogenannte *Short-Circuit Evaluation* aus: Steht nach der Auswertung des ersten Operanden das Endergebnis bereits fest, wird auf die Auswertung des zweiten Ausdrucks verzichtet. Wenn im folgenden Beispiel a>b das Ergebnis false liefert, dann ruft Swift die Funktion calculate gar nicht auf; der logische Ausdruck ist in jedem Fall false, ganz egal, was calculate für ein Ergebnis liefern würde.

```
if a>b && calculate(a, b)==14 { ... }
```

#### 2.3 Range-Operatoren

In Swift gibt es zwei Operatoren, um Bereiche ganzer Zahlen auszudrücken:

- ► Der Closed-Range-Operator a...b beschreibt einen Bereich von a bis inklusive b.
- Der Half-Open-Range-Operator a...<b beschreibt hingegen einen Bereich von a bis exklusive b, entspricht also a...b-1.

a muss kleiner als b sein, sonst ist der Ausdruck ungültig. Mit den Range-Operatoren definierte Bereiche können in Schleifen, in switch-case-Ausdrücken sowie mit dem vorhin vorgestellten Operator ~= verarbeitet werden.

```
for i in 1...10 {
    print(i)
}
```

In der folgenden Schreibweise können Sie Zahlenbereiche zur Initialisierung eines Integer-Arrays verwenden:

```
var ar = [Int](1...10) // entspricht var ar = [1, 2, ..., 10]
```

Mit map können Sie auf einen Zahlenbereich direkt eine Funktion (Closure) anwenden:

```
(1...10).map { print($0) }
```

Die Operatoren ... und ... sind in Wirklichkeit nur eine Kurzschreibweise zur Erzeugung von Range-Elementen.

1<10	//	entspricht	<pre>Range &lt; Int &gt; ( start : 1,</pre>	end:10)
110	//	entspricht	<pre>Range &lt; Int &gt; ( start : 1,</pre>	end:11)

Range ist also eine Datenstruktur für einen Zahlenbereich. Die wichtigsten Eigenschaften sind startIndex und endIndex. Sie geben den Start- bzw. Endwert des Bereichs an:

```
var r = 1..<10 // entspricht r=Range<Int>(start:1, end:10)
r.startIndex // 1
r.endIndex // 10
```

Range-Elemente können nicht nur für Integer-Zahlen gebildet werden, sondern auch für andere Datentypen, die dem Protokoll ForwardIndexType entsprechen. Dazu zählen unter anderem auch String.Index-Elemente zur Positionsangabe in Zeichenketten. Double-Zahlen sind hingegen nicht geeignet, weil sie weder das ForwardIndexType-Protokoll noch dessen successor-Methode unterstützen.

#### Interval-Operatoren

Je nach Kontext können die Operatoren a...b und a..<b auch zur Formulierung eines Intervalls verwendet werden – z.B. als Vergleichsbasis für switch bzw. für den oben beschriebenen Operator ~=. In diesem Fall sind für a und b auch Fließkommazahlen oder einzelne Zeichen (Character) erlaubt.

110 ~=	8		//	true
1.7<2.9	~=	2.3	//	true
"a""z"	~=	"f"	//	true
"0""9"	$\sim =$	" x "	//	false

Intern werden hier durch ... bzw. ..< nicht Range-Elemente gebildet, sondern ein ClosedInterval bzw. ein HalfOpenInterval:

ClosedInterval(1.0, 3.4) // entspricht 1.0...3.4 HalfOpenInterval(1.0, 3.4) // entspricht 1.0..<3.4

### 2.4 Operatoren für Fortgeschrittene

Die wichtigsten Operatoren kennen Sie nun. Dieser Abschnitt ergänzt Ihr Wissen um Spezial- und Hintergrundinformationen. Am interessantesten ist dabei sicherlich die Möglichkeit, selbst eigene Operatoren zu definieren bzw. vorhandene Operatoren zu überschreiben (Operator Overloading).

#### **Ternärer Operator**

Swift kennt drei Typen von Operatoren:

- Unäre Operatoren (Unary Operators) verarbeiten nur einen Operanden. In Swift sind das neben dem positiven und negativen Vorzeichen und dem logischen NICHT (also !) die Inkrement- und Dekrement-Operatoren ++ und --.
- ► Binäre Operatoren verarbeiten zwei Operanden, also etwa das a \* b. Die Mehrheit der Swift-Operatoren fällt in diese Gruppe.
- ► Ternäre Operatoren verarbeiten drei Operanden. In Swift gibt es nur einen derartigen Vertreter, der daher einfach als *ternärer Operator* bezeichnet wird – so, als wären andere ternäre Operatoren undenkbar.

Die Syntax des ternären Operators sieht so aus:

a ? b : c

Wenn der boolesche Ausdruck a wahr ist, dann liefert der Ausdruck b, sonst c. Der ternäre Operator eignet sich dazu, einfache if-Verzweigungen zu verkürzen:

```
// if-Schreibweise
let result:String
let x = 3
if x<10 {
    result = "x kleiner 10"
} else {
    result = "x größer-gleich 10"
}
// verkürzte Schreibweise mit ternären Operator
let x = 3
let result = x<10 ? "x kleiner 10" : "x größer gleich 10"</pre>
```

#### Unwrapping- und Nil-Coalescing-Operator

Im Gegensatz zu den meisten anderen Programmiersprachen können mit einem Typ deklarierte Swift-Variablen nie den Zustand null im Sinne von »nicht initialisiert« annehmen. Swift bietet dafür die Möglichkeit, eine Variable explizit als *Optional* zu deklarieren (siehe <u>Abschnitt 3.5</u>, »Optionals«). Dazu geben Sie explizit den gewünschten Datentyp an, dem wiederum ein Fragezeichen oder ein Ausrufezeichen folgt:

```
var x:Int? = 3 // x enthält eine ganze Zahl oder nil
var y:Int! = 4 // y enthält eine ganze Zahl oder nil
var z:Int = 5 // z enthält immer eine ganze Zahl
x = nil // ok
y = nil // ok
z = nil // nicht erlaubt
```

Der Unterschied zwischen x und y besteht darin, dass das Auspacken (*Unwrapping*) des eigentlichen Werts bei y automatisch erfolgt, während es bei x durch ein nachgestelltes Ausrufezeichen – den Unwrapping-Operator – erzwungen werden muss. Beachten Sie, dass die folgenden Zeilen beide einen Fehler verursachen, wenn x bzw. y den Zustand nil aufweiset!

```
var i:Int = x! // explizites Unwrapping durch x!
var j:Int = y // automatisches Unwrapping
```

Die Variablen x und y können also nil enthalten. nil hat eine ähnliche Bedeutung wie bei anderen Sprachen null. Optionals sind aber gleichermaßen für Wert- und für Referenztypen vorgesehen, was in manchen anderen Programmiersprachen nicht oder nur auf Umwegen möglich ist. Allerdings können nur solche Klassen für Optionals verwendet werden, die das Protokoll NilLiteralConvertible einhalten.

Mit diesem Vorwissen kommen wir nun zum Nil-Coalescing-Operator a ?? b, der sich ebenso schwer aussprechen wie übersetzen lässt. Bei ihm handelt es sich um eine Kurzschreibweise des folgenden Ausdrucks:

a != nil ? a! : b

Wenn a initialisiert ist, also nicht nil ist, dann liefert a ?? b den Wert von a zurück, andernfalls den Wert von b. Damit eignet sich b zur Angabe eines Defaultwerts. In a! bewirkt das Ausrufezeichen das Auspacken (*Unwrapping*) des Optionals. Aus dem Optional Typ? wird also der reguläre Datentyp Typ.

var j = x ?? -1 // der Datentyp von j ist Int

#### **Optional Chaining**

Ebenfalls mit Optionals hat die Operatorkombination ?. zu tun. Sie testet, ob ein Ausdruck nil ergibt. Ist dies der Fall, lautet das Endergebnis nil. Andernfalls wird das Ergebnis ausgepackt und der nächste Ausdruck angewendet. Wenn dieser ebenfalls ein Optional liefert, kann auch diesem Ausdruck ein Fragezeichen hintangestellt werden. Swift führt einen weiteren nil-Test durch. Diese Verkettung von nil-Tests samt Auswertung, wenn der Ausdruck nicht nil ist, heißt in Swift »Optional Chaining«:

```
let a = optional?.method()?.property
let b = optional?.method1()?.method2()?.method3()
```

#### **Operator-Präferenz**

Beim Ausdruck a + b \* c rechnet Swift zuerst b\*c aus, bevor es summiert – so wie Sie es in der Schule gelernt haben. Generell gilt in Swift eine klare Hierarchie der Operatoren (siehe <u>Tabelle 2.2</u>). Um die Verarbeitungsreihenfolge zu verändern, können Sie natürlich jederzeit Klammern setzen – also beispielsweise (a+b)\*c.

Priorität	Operatoren	Gruppe
160	<< >>	Exponential-Funktionen
150 ←	* / % &*	Multiplikation
140 ←	+ - &+ &-   ^	Addition, Subtraktion
135	<	Range-Operatoren
132	is as	Casting
$131 \rightarrow$	??	Nil Coalescing
130	< <= > >= == != === !=== ~=	Vergleiche
120 ←	&&	Logisches Und
110 ←		Logisches Oder
100 →	?:	Ternärer Operator
90 →	= *= /= %= += -= <<= >>=	Zuweisungen
90 →	&= ^=  = &&=   =	Zuweisungen

 Tabelle 2.2
 Hierarchie der binären Operatoren

Die erste Spalte der Operatorentabelle gibt neben der Priorität auch die Assoziativität an: Sie bestimmt, ob gleichwertige Operatoren von links nach rechts oder von rechts
nach links verarbeitet werden sollen. Beispielsweise ist - (Minus) ein linksassoziativer Operator. Die Auswertung erfolgt von links nach rechts. 17 - 5 - 3 wird also in der Form (17 - 5) - 3 verarbeitet und ergibt 9. Falsch wäre 17 - (5 - 3) = 15!

Möglicherweise wundern Sie sich über die merkwürdigen Prioritätswerte. Diese sind aus der Deklaration der Swift-Bibliothek übernommen, die Sie in Xcode lesen können, wenn Sie ein Swift-Schlüsselwort wie Int mit 🕷 anklicken. Die Werte sind dann wichtig, wenn Sie eigene Operatoren definieren möchten: Dabei müssen Sie nämlich auch deren Priorität festlegen.

# 2.5 Operator Overloading

Swift bietet die Möglichkeit, Operatoren für bestimmte Datentypen neue Funktionen zuzuweisen. Das folgende Beispiel definiert zuerst die Datenstruktur Complex zur Speicherung komplexer Zahlen mit Real- und Imaginärteil. Die weiteren Zeilen zeigen die Implementierung der Operatoren + und \* zur Verarbeitung solcher Zahlen.

Operatoren sind aus der Sicht von Swift ein Sonderfall globaler Funktionen, wobei der Funktionsname aus Operatorzeichen besteht. Binäre Operatoren erwarten zwei, unäre Operatoren einen Parameter.

```
struct Complex {
 var re:Double, im:Double
 init(re:Double, im:Double) {
    self.re=re; self.im=im
  }
}
// Addition komplexer Zahlen
func + (left: Complex, right: Complex) -> Complex {
 return Complex(re:left.re + right.re, im:left.im + right.im)
}
// Multiplikation komplexer Zahlen
func * (left: Complex, right: Complex) -> Complex {
 return Complex(re:left.re*right.re - left.im * right.im,
                 im:left.re*right.im+left.im*right.re)
}
// Vergleich komplexer Zahlen
func == (left: Complex, right: Complex) -> Bool {
 return left.re==right.re && left.im==right.im;
}
func !=(left: Complex, right: Complex) -> Bool {
 return !(left==right);
}
```

```
// Operatoren anwenden
var a = Complex(re: 2, im: 1) // 2 + i
var b = Complex(re: 1, im: 3) // 1 + 3i
var c = a + b // 3 + 4i
var d = a * b // -1 + 7i
```

Bei der Definition unärer Operatoren muss mit dem Schlüsselwort prefix bzw. postfix angegeben werden, ob der Operator vor oder nach dem Operanden angegeben wird. Die Definition des Operators für negative Vorzeichen bei komplexen Zahlen sieht so aus:

```
// negatives Vorzeichen für komplexe Zahlen
prefix func - (op: Complex) -> Complex {
  return Complex(re: -op.re, im: -op.im)
}
```

Nicht überschrieben werden können der Zuweisungsoperator = und der ternäre Operator a ? b : c. Dafür können Sie für bisher nicht genutzte Sonderzeichenkombinationen selbst vollkommen neue Operatoren definieren. Dazu müssen Sie zuerst mit infix|prefix|postfix operator den Operator selbst definieren. Optional können Sie dabei auch die Assoziativität (associativity left|right) und einen Wert zur Festlegung der Operatorhierarchie angeben.

### Vergleichsoperator für Zeichenketten

Das folgende Beispiel definiert einen Vergleichsoperator für Zeichenketten, der nicht zwischen Groß- und Kleinschreibung unterscheidet. infix bezeichnet dabei einen Operator für zwei Operanden. Der Operator =~= erhält dieselbe Priorität wie die anderen Vergleichsoperatoren. Die Implementierung greift auf die compare-Methode zurück. Besser lesbar, aber weniger effizient wäre return left.lowercaseString == right.lowercaseString.

```
// neuen Vergleichsoperator definieren,
infix operator =~= { precedence 130 }
// implementieren,
func =~= (left:String, right:String) -> Bool {
  return
    left.compare(right, options:NSStringCompareOptions.
        CaseInsensitiveSearch) == NSComparisonResult.OrderedSame
}
// und ausprobieren
"abc" =~= "Abc" // true
```

# Kapitel 5 Verzweigungen und Schleifen

Verzweigungen mit if und Schleifen mit for sind in den vergangenen Kapiteln ja schon mehrfach vorgekommen. Dieses Kapitel geht der Steuerung des Programmflusses genauer nach. Sie lernen hier die if-Alternativen guard und switch sowie while-Schleifen kennen und erfahren, wie Sie Schleifen vorzeitig mit break verlassen bzw. teilweise mit continue überspringen können.

# 5.1 Verzweigungen mit if

»Verzweigungen« sind Code-Abschnitte, an denen bei der Ausführung des Programms in Abhängigkeit von einer Bedingung unterschiedliche Code-Pfade ausgeführt werden. In Swift werden Verzweigungen mit if oder switch gebildet. In besonders einfachen Fällen kann anstelle von if der ternäre Operator oder der Nil-Coalescing-Operator eingesetzt werden. Diese Operatoren wurden in <u>Kapitel 2</u> näher vorgestellt.

```
// das Ergebnis ist a, wenn die Bedingung erfüllt ist, sonst b
let ergebnis1 = bed ? a : b
// das Ergebnis lautet a, wenn das Optional a einen Wert
// enthält, sonst b
let ergebnis2 = a ?? b
```

# if

if ist das gängigste Konstrukt, um Verzweigungen zu formulieren. if funktioniert in Swift wie in den meisten anderen Programmiersprachen. Es gibt aber zwei syntaktische Feinheiten: Zum einen ist es nicht notwendig, die Bedingung in runde Klammern zu stellen, zum anderen *müssen* die resultierenden if- und else-Blöcke in geschwungene Klammern gesetzt werden – selbst dann, wenn sie nur aus einer einzigen Anweisung bestehen. Es sind beliebig viele else-if-Teile erlaubt. Der else-Block ist optional.

```
let n = Int(arc4random_uniform(100)) // Zahl zwischen 0 und 99
if n<10 {
    print("n ist kleiner 10")
} else if n<=50 {
    print("n liegt zwischen 10 und 50")
} else if n<75 && n%2==0 {
    print("n ist eine gerade Zahl zwischen 52 und 74")
} else {
    print("n ist eine andere Zahl: \(n)")
}</pre>
```

## if-let-Kombination für Optionals

Zur Verarbeitung von Optionals kennt Swift eine spezielle Kombination einer Zuweisung mit let oder var und einer Abfrage durch if:

```
if let x = optional {
    // dieser Code wird nur ausgeführt, wenn das
    // Optional nicht nil ist; x ist eine Konstante
}
if var y = optional {
    // dieser Code wird nur ausgeführt, wenn das
    // Optional nicht nil ist; y ist eine Variable
}
```

Wenn das Optional ungleich nil ist, dann speichert Swift den ausgepackten Inhalt des Optionals in x und führt den Code in den geschwungenen Klammern aus. Die Konstante x bzw. die Variable y ist nur bis zum Ende des if-Blocks gültig.

```
let someStringData = "10"
if let n = Int(someStringData) {
   // Int war erfolgreich, n ist eine Int-Zahl
   for var i=1; i<=n; i++ {
        // ...
   }
} // hier endet der Geltungsbereich von n</pre>
```

Das folgende Beispiel aus der Praxis illustriert diese Syntaxvariante. Die oft benötigte Funktion NSSearchPathForDirectoriesInDomains liefert ein Array mit Verzeichnissen zurück. first greift auf das erste Element dieses Arrays zurück. Diese Methode kann aber nil liefern, wenn das Array leer ist – daher die Absicherung durch if-let:

```
let pfd = // pfd hat den Datentyp [String]
NSSearchPathForDirectoriesInDomains(
    .DocumentDirectory, .UserDomainMask, true)
```

```
if let path = pfd.first {
    print(path) // path hat den Datentyp String
}
```

Ob Sie var oder let vorziehen, hängt davon ab, ob Sie die Variable im weiteren Verlauf ändern möchten oder ob dazu keine Notwendigkeit besteht. In der Praxis ist if-let die gängigere Kombination.

Sie können auch mehrere Variablen zugleich zuweisen. In diesem Fall gilt die if-Bedingung nur dann als erfüllt, wenn *alle* optionalen Ausdrücke ungleich nil sind:

```
if let a=opt1, b=opt2, c=opt3 { ... }
```

Vor let oder var können Sie noch eine optionale Bedingung angeben. Nur wenn diese erfüllt ist *und* die Optionals ungleich nil sind, wird der if-Block ausgeführt:

```
if condition, let/var a=opt1, b=opt2, c=opt3 {
    // dieser Code wird nur ausgeführt, wenn die Bedingung
    // zutrifft und opt1, opt2 und opt3 jeweils ungleich nil
    // sind
}
```

#### if-let ist nur mit »as?« zulässig

Wenn Sie in einer if-let-Kombination ein Casting durchführen, müssen Sie dazu as? verwenden. as! ist nicht erlaubt!

### if-let-Kombination mit where

Variablenzuweisungen mit if-let bzw. if-var lassen sich auch mit Bedingungen verknüpfen, die durch where formuliert werden. Im Vergleich zur vorhin beschriebenen Syntaxvariante if condition, let x=... kann die mit where formulierte Bedingung auf die gerade zugewiesenen Variablen zurückgreifen:

```
if let/var x=optional() where condition {
    // dieser Code wird nur ausgeführt, wenn optional() ein
    // Ergebnis ungleich nil liefert und die Bedingung
    // erfüllt ist
}
```

Das Schlüsselwort where darf nur einmal *nach* allen Zuweisungen verwendet werden. Mehrere Bedingungen können durch logische Operatoren wie && oder || verknüpft werden:

```
var opt1:Int? = 4
var opt2:Int? = 2
var opt3:Int? = 3
```

```
if let a=opt1, b=opt2, c=opt3 where a==b*b && c>2 {
   print("bingo")
}
```

Die folgenden Zeilen aus dem Beispielprogramm »Icon-Resizer« aus <u>Kapitel 20</u> zeigt eine komplexe if-let-Kombination aus der Praxis: if testet, ob die Anzahl der Drag & Drop-Objekte genau eins beträgt, ob die Methode readObjectsForClasses ein String-Array liefert und ob schließlich das erste Element dieses Arrays die Zeichenkette "xy" ist.

```
private func dragString(draginfo: NSDraggingInfo)
                      -> Bool
{
    let pboard = draginfo.draggingPasteboard()
    if draginfo.numberOfValidItemsForDrop == 1,
        let data = pboard.readObjectsForClasses(
                    [NSString.self], options: [:]) as? [String]
        where data.first == "xy"
    {
        return true
    }
    return false
}
```

### Inverse Logik mit guard

if-let-Konstruktionen bergen eine Tendenz zu unübersichtlichem Code in sich: Es passiert recht oft, dass zuerst umfangreicher Code für den positiven Fall formuliert wird; zum Ende der Funktion folgen dann kurze Einzeiler, die ausgeführt werden, wenn if-let nicht erfolgreich war. Das führt zu unnötig verschachteltem Code.

Im folgenden Beispiel ruft f zweimal die Funktion perhapsANumber auf und verarbeitet die Ergebnisse. Das folgende Listing zeigt die Implementierung von f ohne guard:

```
// Datei guard-test.playground
func perhapsANumber() -> Int? {
   let n = Int(arc4random_uniform(100))
   if n <= 50 {
      return n
   } else {
      return nil
   }
}</pre>
```

```
// ohne guard
func f() {
    if let a = perhapsANumber() {
        let n = 2 * a
        if let b = perhapsANumber() {
            print(n + b)
            // noch mehr Code, Teil 1
        } else {
            return
        }
    } else {
        return
    }
    // noch mehr Code, Teil 2
}
```

guard stellt die Funktion von if-let gewissermaßen auf den Kopf: Ist die Bedingung erfüllt, wird der Code *nach* dem else-Block ausgeführt, wobei alle mit let definierten Variablen weiterhin gültig sind. Ist die Bedingung hingegen nicht erfüllt, wird der else-Block ausgeführt. Mit guard lässt sich die gleiche Aufgabe wie in f wesentlich eleganter erledigen:

```
// mit guard
func g() {
  guard let a = perhapsANumber() else { return }
  let n = 2 * a
  guard let b = perhapsANumber() else { return }
  print(n + b)
  // noch mehr Code, Teil 1
  // noch mehr Code, Teil 2
}
```

Wie bei if-let können Sie in guard-Konstruktionen auch mehrere Zuweisungen durchführen, Variablen anstelle von Konstanten verwenden (guard var = ...), mit where weitere Bedingungen formulieren etc.:

```
guard let a = perhapsANumber(),
    var b = perhapsANumber() where a+b > 10
else {
    return
}
// Code, um a und b zu verarbeiten
```

### Versionsabhängige Code-Teile

Bei der Entwicklung von Apps, die unter mehreren Versionen von iOS oder OS X laufen sollen, stehen Sie vor einem Dilemma: Nutzen Sie die neuesten Features, dann läuft Ihre App nur auf Geräten, auf denen ebenfalls eine ganz aktuelle Version des Betriebssystems installiert ist. Orientieren Sie sich aber an einer älteren iOS- oder OS-X-Version, dann müssen Sie und Ihre Kunden auf neue und oft nützliche Funktionen verzichten.

In manchen Fällen bietet das Schlüsselwort #available einen Ausweg: Damit können Sie Code-Teile markieren, die nur dann ausgeführt werden, wenn bestimmte Voraussetzungen erfüllt sind. Die Syntax ist einfach:

```
if #available(iOS 8.2, OSX 10.10, *) {
    // ausführen, wenn zumindest iOS 8.2 oder OS X 10.10
    // zur Verfügung steht
} else {
    // bei älteren Versionen ausführen
}
```

An #available übergeben Sie eine Aufzählung von Betriebssystemnamen und Versionsnummern. Der letzte Parameter \* ist syntaktisch erforderlich und bedeutet, dass bei allen nicht explizit genannten Betriebssystemen (z. B. Watch OS) beliebige Versionsnummern akzeptiert werden.

Natürlich können Sie #available auch mit guard kombinieren:

```
func m() {
    // bei älteren Versionen als iOS 8.1 return ausführen
    guard #available(iOS 8.1, *) else { return }
    // wenn zumindest iOS 8.2 zur Verfügung steht,
    // den weiteren Code ausführen
}
```

#available klingt in der Theorie toll. In der Praxis scheitert der Einsatz des neuen Schlüsselwort aber oft daran, dass sich damit nur einzelne Anweisungen versionsabhängig implementieren lassen, nicht aber ganze Methoden, Funktionen oder Erweiterungen.

# 5.2 Verzweigungen mit switch

switch bietet sich vor allem dann als Alternative zu if an, wenn bei der Auswertung eines Ausdrucks viele unterschiedliche, klar definierte Fälle möglich sind. switch-Konstruktionen sind in solchen Fällen oft besser lesbar. Nachdem der erste zutreffende case-Block durchlaufen wurde, wird die gesamte Konstruktion verlassen. Sollten also mehrere Bedingungen zutreffen, wird nur der erste passende case-Block berücksichtigt.

Im folgenden Beispiel ermittelt eine switch-Konstruktion die Anzahl der Tage eines Monats:

```
let monat = "Februar", jahr = 2015
var tage:Int?
switch monat {
case "Januar", "März", "Mai", "Juli", "August",
    "Oktober", "Dezember":
  tage = 31
case "April", "Juni", "September", "November":
  tage = 30
case "Februar":
  if jahr%4 == 0 && (jahr%100 != 0 || jahr%400==0) {
   tage = 29
  } else {
    tage = 28
  }
default:
  print("Ungültiger Monatsname!")
}
```

Im Vergleich zu anderen Programmiersprachen gibt es in Swift mehrere Besonderheiten:

► Nach der Ausführung des ersten zutreffenden case-Blocks wird die switch-Konstruktion verlassen. Es ist also nicht wie in Java oder C ein break am Ende jedes case-Blocks erforderlich.

break ist aber durchaus ein zulässiges Schlüsselwort innerhalb eines case-Blocks. Mit break kann ein case-Block vorzeitig verlassen werden.

Wenn Sie möchten, dass sich switch in Swift so wie in C oder Java verhält, formulieren Sie am Ende jedes case-Blocks die Anweisung fallthrough. Damit wird auch der nächste case-Block durchlaufen, und zwar ohne die dort formulierte Bedingung zu überprüfen.

► Jede switch-Konstruktion muss einen default-Block aufweisen. Die einzige Ausnahme ist die Auswertung einer Enumeration. Wenn der Compiler erkennt, dass sämtliche Enumerationswerte abgefragt wurden, darf default entfallen. Sie müssen default also auch dann vorsehen, wenn Ihr Code gar keinen Bedarf dafür hat. Welche Anweisung führen Sie dann aber im default-Block aus? Für solche Fälle bietet sich wiederum break an. Die Zeile default: break bewirkt also, dass Swift ganz einfach nichts tut, wenn keine der Fallunterscheidungen zutrifft.

Zahlen- und Zeichenbereiche in case-Ausdrücken dürfen in der Range-Syntax n1... n2 oder n1..<n2 formuliert werden. Swift betrachtet die Operatoren ... und ... in diesem Kontext allerdings nicht als Range-Operatoren, sondern bildet vielmehr Intervalle (ClosedInterval bzw. HalfOpenInterval). Deswegen dürfen der Start- und der Endwert hier auch Fließkommazahlen bzw. einzelne Zeichen sein ("a"..."z").

```
// Intervall-Tests in switch
let d = 1.5
switch d {
  case 0.0...1.0:
    print("0 <= d <= 1")
  case 1.0...2.0:
    print("1 < d <= 2")
  default:
    print("anderer Wert")
}</pre>
```

# switch für Tupel

switch kann auch mehrere Ausdrücke gleichzeitig auswerten, die als Tupel übergeben werden. Daraus ergeben sich interessante Spielarten, die in <u>Abschnitt 4.5</u> schon erwähnt wurden:

```
// Tupel-Auswertung in switch
let pt = (0.0, 0.0)
switch pt {
  case (0, 0):
    print("Koordinatenursprung")
  case (_, 0):
    print("Auf der X-Achse")
  default:
    print("Sonstwo")
}
```

### case-let-Kombination mit where

Die Tupel-Auswertung lässt sich noch weiter perfektionieren. Ähnlich wie bei if können Sie den switch-Ausdruck in einem case-Block durch var oder let einer neuen Variablen oder Konstante zuweisen und diese Zuweisung außerdem an eine mit where formulierte Bedingung knüpfen:

```
// Tupel-Auswertung in switch
var pt = (0.0, 0.0)
switch pt {
case let (x, y) where x>0 && y>0:
    print("Im ersten Quadranten")
...
}
```

Mit where verknüpfte Zuweisungen sind auch für gewöhnliche switch-Ausdrücke zulässig:

```
let s = "bild.jpg"
switch s {
  case let jpg where jpg.hasSuffix(".jpg"):
    print("JPEG-Datei")

case let gif where gif.hasSuffix(".gif"):
    print("GIF-Datei")

default:
    print("eine andere Datei")
}
```

Die folgende Variante kommt sogar mit Dateinamen zurecht, die die Kennung .jpg in Großbuchstaben enthält:

```
case let jpg where jpg.lowercaseString.hasSuffix(".jpg"):
    print("JPEG-Datei")
```

#### »switch« für Enumerationen

Elementen von Swift-Enumerationen können Werte zugeordnet werden. Mit case let ... können Sie diese Werte in switch-Konstruktionen auslesen. Ein entsprechendes Beispiel finden Sie in Abschnitt 7.2.

# 5.3 Schleifen

Dieser Abschnitt stellt Ihnen im Schnelldurchgang vier Schleifenformen vor:

- die klassische for-Schleife
- die for-in-Schleife
- die while-Schleife
- die repeat-while-Schleife (ehemals do-while)

## for

Die klassische for-Schleife folgt der in C, Objective C und Java üblichen Syntax:

```
for initialisierung; bedingung; inkrement { schleifenkörper }
```

Ein typisches Beispiel für eine derartige Schleife sieht so aus:

```
for var i=1; i<=10; i++ {
    print(i)
} // Ausgabe: 1, 2, ..., 10</pre>
```

Beachten Sie, dass der Schleifenkörper einer for-Schleife nie durchlaufen wird, wenn die Schleifenbedingung beim ersten Test nicht erfüllt ist. Es ist zulässig, mehrere Variablen zu initialisieren und im Inkrementbereich zu verändern, wobei die Ausdrücke durch Kommata voneinander getrennt werden. Es ist aber immer nur eine Bedingung zulässig.

### Vorsicht beim Umgang mit Double-Variablen

Beim Rechnen mit Double-Zahlen kann es wie in jeder Programmiersprache zu Rundungsfehlern kommen. Mit etwas Pech wird die Schleife dann nicht so oft ausgeführt, wie Sie vielleicht beabsichtigt haben. Beispielsweise wird die folgende Schleife nur 10und nicht 11-mal durchlaufen. x hat zuletzt den Wert 1,90000000000008.

```
// Achtung, Rundungsfehler
for var x=1.0; x<=2.0; x+=0.1 {
    print(String(format:"%.16f", x))
}</pre>
```

```
// Ausgabe: 1.000000000000000
// 1.10000000000000
// ..
// 1.80000000000000
// 1.90000000000000
```

Um derartige Fehler zu vermeiden, formulieren Sie die Schleife mit einer Integer-Variablen oder berücksichtigen beim Schleifenendwert ein »Ungenauigkeits-Delta«:

```
// Schleife mit Integer-Variablen
for var i=0; i<=10; i++ {
   let x = Double(i) / 10.0
   print(x)
}
// Schleifenendwert plus Delta
let step = 0.1
let delta = step / 1000000
for var x=0.0; x <= 1.0 + delta; x+=step {
   print(x)
}</pre>
```

### for-in

Sie nutzen for-in, wenn Sie alle Elemente eines Arrays, eines Dictionary oder eines Objekts durchlaufen möchten, dessen Typ das SequenceType-Protokoll erfüllt. Zeichenketten erfüllen dieses Protokoll ab der Swift-Version 2.0 nicht mehr; Abhilfe schafft hier die Verwendung der Eigenschaft characters.

```
// i:Int
for i in 1...3 {
  print(i)
}
// Ausgabe: 1, 2, 3
let s = "abc"
for c in s.charachters { // c:Character
  print(c)
}
// Ausgabe: "a", "b", "c"
let dict = ["one":"eins", "two":"zwei"]
for (engl, germ) in dict { // engl:String, germ:String
  print("\(engl) -- \(germ)")
}
// Ausgabe: one -- eins
//
           two -- zwei
```

Die Schleifenvariable einer for-in-Schleife erhält automatisch den passenden Datentyp. for-in-Schleifen sind sehr bequem in der Handhabung, eignen sich aber in der Regel nur zum Auslesen, nicht zum Verändern der Elemente einer Aufzählung.

Wenn Sie eine Schleife n-mal ausführen möchten, aber an der Schleifenvariable gar nicht interessiert sind, können Sie an deren Stelle das Pattern-Zeichen \_ angeben:

```
// 10 Zufallszahlen zwischen 0 und 99 ausgeben
for _ in 1...10 {
    print(arc4random_uniform(100))
}
```

for var in a...b funktioniert nur für ganze Zahlen und nur für Bereiche, in denen a kleiner als b ist. for i in 3...1 ist zwar syntaktisch erlaubt, führt aber bei der Ausführung zu einem Fehler. Im Internet stoßen Sie vielleicht auf for i in reverse(1...3), aber davon sollten Sie ebenfalls Abstand nehmen: reverse erzeugt nämlich extra ein Array für die zu durchlaufenden Elemente, was ausgesprochen ineffizient ist. Die gute alte for-Schleife hat ihre Berechtigung also noch nicht verloren!

```
for var i = 3; i>=1; i-- {
    print(i)
}
```

### while

while-Schleifen werden so lange ausgeführt, wie die Bedingung erfüllt ist. Die folgende Schleife wird somit 10-mal durchlaufen:

```
var n=1
while n<=10 {
    print(n)
    n++
}
// Ausgabe: 1, 2, ..., 10</pre>
```

## while-let-Kombination

Auch while kann ähnlich wie if und switch-case mit einer Zuweisung verbunden werden. Die Zuweisung kann wahlweise mit let oder mit var durchgeführt werden, häufiger ist aber let. Die while-Bedingung gilt als erfüllt, wenn der Ausdruck ungleich nil ist. Diese Art der while-let-Kombination bietet sich besonders dann an, wenn eine Schleife so lange ausgeführt werden soll, bis eine Sequenz, Datei etc. abgearbeitet ist und die Lesemethode oder -funktion dementsprechend nil liefert.

Das folgende Beispiel bildet mit generate einen Generator aus einem Array. Die Methode next liefert das jeweils nächste Element – so lange, bis alle Elemente abgearbeitet sind. Die gleiche Art von Schleife bildet übrigens auch der Swift-Compiler, wenn Sie for i in ar schreiben!

```
var ar = [28, 34, 12]
var gen = ar.generate() // Generator für das Array
while let i = gen.next() { // i hat den Datentyp Int
    print(i)
}
```

### repeat-while

repeat-while-Schleifen (do-while-Schleifen in Swift 1.n) sehen auf den ersten Blick ganz ähnlich wie while-Schleifen aus. Der entscheidende Unterschied besteht darin, dass die Bedingung am Ende der Schleife angegeben ist. Deshalb wird der Ausdruck garantiert mindestens einmal durchlaufen. Bei allen anderen Schleifen kann es hingegen passieren, dass die Bedingung von Anfang an nicht erfüllt ist bzw. die Aufzählung keine Elemente aufweist und die Schleife daher sofort übersprungen wird.

```
var n=1
repeat {
    print(n)
    n++
} while n<=10
// Ausgabe: 1, 2, ..., 10</pre>
```

### break

verlässt eine Schleife vorzeitig, bricht also ihre Ausführung ab. Die folgenden Schleife endet daher nach der Ausgabe des Werts 5:

```
for i in 1...10 {
    print(i)
    if i==5 { break }
}
// Ausgabe: 1, 2, ..., 5
```

break kann gleichermaßen bei allen Schleifentypen verwendet werden. In switch-Konstruktionen bewirkt break ein vorzeitiges Verlassen des case-Blocks. Um mit break mehrere, ineinander verschachtelte Schleifen zu verlassen, müssen Sie vor der betreffenden Schleife ein sogenanntes Label platzieren. break label gilt dann für die so bezeichnete Schleife.

```
iloop: // Label für die nachfolgende Schleife
for i in 1...10 {
    for j in 1...10 {
        print("i=\(i), j=\(j)")
        if i+j > 15 {
            break iloop // die i-Schleife verlassen
        }
    }
}
// Ausgabe:
// i=1, j=1
// i=1, j=2
// ...
// i=6, j=10
```

## continue

Mit continue überspringen Sie den restlichen Schleifenkörper, setzen die Schleife dann aber fort, und zwar:

- ▶ beim klassischen for mit dem Inkrement-Ausdruck und der Bedingung
- ▶ bei for-in mit der Verarbeitung des nächsten Elements
- ▶ bei while bzw. repeat-while mit der Auswertung der Bedingung

continue bezieht sich normalerweise auf die innerste Schleife. Wie bei break können Sie aber mit einem Label die gewünschte Schleife auswählen.

```
for i in 1...10 {
    if i%2 == 0 { continue }
    print(i)
    if i>8 {break}
} // Ausgabe: 1, 3, 5, 7, 9
```

# 5.4 Lottosimulator

Das folgende Beispiel für den praktischen Einsatz von Schleifen beweist einmal mehr, dass Lotto-Spielen Geldverschwendung ist. Ursprünglich war das Beispiel eher dazu gedacht, etwas Abwechslung in dieses ein wenig eintönige Kapitel zu bringen. Rasch hat sich aber herausgestellt, dass das Beispiel auch einen anderen Aspekt der Swift-Programmierung ausgezeichnet beleuchtet: die Geschwindigkeitsoptimierung. Die erste Version des Programms hat sich nämlich als inakzeptabel langsam herausgestellt.

### Version 1: elegant, aber langsam

Das Ziel des Programms ist, so lange Lottoziehungen zu simulieren, bis die sechs gezogenen Zahlen mit den eigenen Zahlen übereinstimmen. Auf die Superzahl, die Erkennung von fünf übereinstimmenden Zahlen und andere Sonderfälle habe ich verzichtet.

Im folgenden Code läuft die äußere Schleife so lange, bis eine mit Zufallszahlen durchgeführte Ziehung den eigenen sechs »Glückszahlen« entspricht. Die innere Schleife simuliert Lotto-Ziehungen: Dabei werden so lange Zufallszahlen zwischen 1 und 49 in ein Set eingetragen, bis dieses aus sechs Elementen besteht. Zur einfacheren Vergleichbarkeit werden diese in ein sortiertes Array umgewandelt.

```
// Projekt cmd-lotto
import Foundation
// diese Zahlen müssen geordnet sein!
let meineZahlen = [1, 6, 12, 14, 25, 33]
var lotto: [Int]
var cnt=0
repeat {
  var set = Set <Int >() // leeres Set erzeugen
                              // Ziehung simulieren
  repeat {
    set.insert(Int(arc4random uniform(49))+1)
  } while(set.count<6)</pre>
  lotto = Array(set).sort(<)</pre>
  cnt++
  if cnt % 100 000 == 0 {
    print("Bisher \(cnt) Ziehungen")
  }
  // Schleife ausführen, bis beide Arrays übereinstimmen
} while meineZahlen != lotto
```

```
print("Sechs Richtige nach \(cnt) Versuchen")
```

Das Programm ist aus Geschwindigkeitsgründen für den Playground ungeeignet. Damit der Lotto-Simulator in einer angemessenen Geschwindigkeit ausgeführt wird, müssen Sie den Code in ein »richtiges« Projekt verpacken. Dazu erstellen Sie mit FILE • NEW • PROJECT ein OS X • APPLICATION • COMMAND LINE TOOL. Um dieses zu einer Release-Version zu kompilieren, führen Sie in Xcode PRODUCT • SCHEME • EDIT SCHEME aus, wählen den Eintrag RUN und das Dialogblatt INFO aus und stellen dort die BUILD CONFIGURATION auf RELEASE um (siehe Abbildung 5.1).

1 target	Info Arguments Options Diagnostics	
Release	Build Configuration Release	
F Test Debug	Executable 🛛 🔳 cmd-lotto 😜	
Horfile Release Debug	Debug executable  Debug Process As      Me (kofler)      Toot	
P Archive Release	Launch S Automatically Wait for executable to be launched	

Abbildung 5.1 Xcode-Einstellung für ein Release-Kompilat

Selbst in dieser Konfiguration und auf einem leistungsfähigen iMac dauert die Ausführung des Programms etwa eine halbe Minute, bis im Debug-Bereich von Xcode die Erfolgsmeldung erscheint (siehe <u>Abbildung 5.2</u>). Statistisch gesehen sind dafür rund 15 Millionen simulierte Ziehungen erforderlich.

•	• • • • • • • • • • • • • • • • • • •					
BB	< 🖒 🛓 🔤 cmd-lotto 🤇 🚞 cmd-lotto 🕽 📄 main.swift 🖯 No Selection					
123						
124	// Lottosimulator, ursprüngliche Variante					
125	<pre>func lottoSimulator1() {</pre>					
120	// diese Zahlen müssen geordnet sein!					
127	[tet methezanten = [1, 0, 12, 14, 25, 55]					
120						
130	var circ-o					
131	repeat {					
132	<pre>var set = Set<int>() // leeres Set erzeugen</int></pre>					
133	repeat { // Ziehung simulieren					
134	<pre>set.insert(Int(arc4random_uniform(49))+1)</pre>					
135	} while(set.count<6)					
136	// Keys des Dictionary in sortiertes Array umwandeln					
137	lotto = Array(set).sort(<)					
138						
139	iii + i = 1000000 - 0.5					
140	print ("Bisher ) (cnt) Ziehungen")					
	prince bisher ((enc) Lienangen /					
Bish Bish Bish Bish Bish Bish Bish Bish	ter 21400000 Ziehungen ter 2150000 Ziehungen ter 2150000 Ziehungen ter 2100000 Ziehungen ter 2100000 Ziehungen ter 22000000 Ziehungen ter 2200000 Ziehungen ter 22200000 Ziehungen ter 22200000 Ziehungen ter 22200000 Ziehungen ter 22500000 Ziehungen ter 2250000 Ziehungen ter 2250000 Ziehungen ter 2250000 Ziehungen					
All	Output 🗘	🕅   🗖 🛄				

Abbildung 5.2 Ein Sechser im Lotto-Simulator

### **Einige Benchmarktests**

Vergleichbare Aufgabenstellungen habe ich in der Vergangenheit auch schon mit anderen Programmiersprachen gelöst – und habe dabei weniger lange auf meinen simulierten Lotto-Sechser warten müssen. Warum ist das Programm so langsam?

Mein erster Verdacht richtete sich auf das Erzeugen der Zufallszahlen. Da Zufallszahlen in vielen Kryptografie-Algorithmen benötigt werden, legt arc4random\_uniform Wert darauf, möglichst zufällige Zahlen zu liefern. (Ganz kann das nie gelingen. Vom Computer generierte Zufallszahlen sind nie *wirklich* zufällig.) Aber ein kurzer Test bewies: Das Erzeugen von 100 Millionen Zufallszahlen dauert auf meinem Testrechner nicht einmal eineinhalb Sekunden.

```
// Projekt cmd-lotto, Datei main.swift
let start = NSDate()
var sum=0
for _ in 1...100_000_000 {
   sum += Int(arc4random_uniform(100))
}
let end = NSDate()
let seconds = end.timeIntervalSinceDate(start)
print("fertig nach \(seconds) Sekunden")
```

arc4random ist also unschuldig. Mein nächster Verdächtiger war das Set, das ich zum bequemen Erzeugen von Lottozahlen missbraucht habe. Das Set kommt hier zum Einsatz, weil es nur eindeutige Werte zulässt. Ohne Set muss beim Hinzufügen jeder neuen Lottozahl überprüft werden, ob es sich hierbei nicht um einen Doppelgänger handelt. Das erfordert eine zusätzliche Schleife und macht den Code unübersichtlicher – aber auch um den Faktor fünf schneller als mit dem ursprünglichen Set-Code!

```
for in 1...1 000 000 { // 1.000.000 Lottoziehungen simulieren
 var lotto = [Int](count:6, repeatedValue:0)
 var n=0
  repeat {
    lotto[n] = Int(arc4random uniform(49))+1
    // Doppelgängertest
    for i in 0..<n {</pre>
      if lotto[i] == lotto[n] {
            // die n-te Lottozahl ist ein Doppel-
        n - -
        break // gänger, neu erzeugen
      }
    }
    n++
  } while(n<6)</pre>
  lotto.sortInPlace(<)</pre>
}
```

Aber auch mit der verbesserten Variante war ich noch nicht glücklich. Das Erzeugen von 100.000.000 Zufallszahlen hat bewiesen, dass es noch schneller gehen müsste. Glücklicherweise ist mir da ins Auge gesprungen, dass ich das lotto-Array eigentlich nicht eine Million Mal neu erzeugen muss.

Ich habe die Zeile var lotto=... also oberhalb der for-Schleife angeordnet, ohne mir davon Wunder zu versprechen. Doch genau dieses Wunder traf ein: Die Rechenzeit sank auf meinem Testrechner von 0,32 auf 0,21 Sekunden, also auf circa zwei Drittel der ursprünglichen Zeit. Oder, anders formuliert: Im obigen Code beansprucht die Zeile var lotto = [Int](count:6, repeatedValue:0) beachtliche 30 Prozent der Rechenzeit!

```
// wie oben, aber mit Recycling des lotto-Arrays:
// um 30% schneller
var lotto = [Int](count:6, repeatedValue:0) // <-- neu platziert
for _ in 1...1_000_000 {
  var n=0
  repeat {
    // Code wie oben ...
  } while(n<6)
  lotto.sortInPlace(<)
}
```

Im Versuch, den Algorithmus weiter zu optimieren, habe ich den Datentyp des Arrays von Int auf UInt8 umgestellt – ohne Erfolg. Die Rechenzeit blieb im Rahmen der Messgenauigkeit unverändert.

Optimierungspotenzial habe ich dann nur noch bei sortInPlace gefunden: Ohne das Sortieren sinkt die Rechenzeit nochmals auf die Hälfte. Die Zahlen müssen aber sortiert werden, damit sie mit den ebenfalls geordneten Glückszahlen verglichen werden können. Wenn Sie das Programm also weiter optimieren wollten, könnten Sie versuchen, einen effizienteren Sortieralgorithmus zu implementieren oder den Zahlenvergleich ohne vorheriges Sortieren durchzuführen. Darauf habe ich verzichtet.

### Version 2: Swift zeigt, was es kann

Mit dem so gewonnenen Wissen habe ich dann die endgültige Version des Lottosimulators fertiggestellt. Der Code sieht so aus:

```
// Projekt cmd-lotto, Datei main.swift
// diese Zahlen müssen geordnet sein!
let meineZahlen = [1, 6, 12, 14, 25, 33]
var lotto = [Int](count:6, repeatedValue:0)
var cnt=0
```

```
repeat {
  // Lottoziehung simulieren
  var n=0
  repeat {
    lotto[n] = Int(arc4random uniform(49))+1
    // Doppelgängertest
    for i in 0.. < n {
      if lotto[i] == lotto[n] {
        n-- // die n-te Lottozahl ist ein Doppel-
        break // gänger, neu erzeugen
      }
    }
    n++
  } while(n<6)</pre>
  lotto.sortInPlace(<)</pre>
  cnt++
  if cnt % 100 000 == 0 {
    print("Bisher \(cnt) Ziehungen")
  }
 // Schleife ausführen, bis beide Arrays übereinstimmen
} while meineZahlen != lotto
print("Sechs Richtige nach \(cnt) Versuchen")
```

# Kapitel 10 Hello iOS-World!

Mittlerweile sollten Ihnen alle wesentlichen Sprachmerkmale von Swift vertraut sein. Mit diesem Kapitel beginnt nun (endlich) die konkrete iOS-Programmierung. In seinem Zentrum steht ein einfaches Hello-World-Programm. Es erfüllt keine andere Aufgabe, als Sie mit den Grundfunktionen von Xcode sowie mit einigen Konzepten der iOS-Programmentwicklung bekannt zu machen. Ein weiteres Thema dieses Kapitels besteht darin, die Hello-World-App auf dem eigenen iPhone oder iPad auszuführen. Dazu müssen Sie Mitglied im iOS-Entwicklerprogramm werden.

Bei der Lektüre dieses und auch des nächsten Kapitels werden Sie feststellen, dass Swift vorübergehend in den Hintergrund rückt. Natürlich müssen wir da und dort ein paar Zeilen Code schreiben; aber gerade bei den ersten iOS-Programmen geht es vielmehr darum, den effizienten Umgang mit Xcode zu erlernen, die grafische Gestaltung von Apps kennenzulernen und grundlegende Konzepte der iOS-Programmentwicklung zu verstehen.

### Xcode per Video kennenlernen

Ich wäre nicht seit 30 Jahren mit Begeisterung Autor, würde ich nicht aus vollster Überzeugung das Medium Buch lieben. Aber es heißt ja, ein Bild sagt mehr als tausend Worte, und man könnte diese Analogie noch fortführen: Ein Video sagt mehr als tausend Bilder. Mitunter stimmt das sogar, z. B. wenn es darum geht, die Bedienung von Xcode zu erlernen.

Ich will Sie hier nicht an die Konkurrenz verweisen, aber ich möchte Ihnen doch ein Video-Tutorial ans Herz legen, das qualitativ meilenweit über den vielen YouTube-Filmchen der Art »Jetzt lerne ich iOS-Programmierung« steht. Die renommierte Stanford Universität hat eine rund 18-stündige Vorlesung zum Thema »Developing iOS 8 Apps with Swift« kostenlos im iTunes-U-Programm verfügbar gemacht. Um die Videos ansehen zu können, benötigen Sie entweder iTunes auf Ihrem Mac oder die App *iTunesU* für das iPad. Grundlegende Englischkenntnisse reichen aus, um dem mit vielen Screencasts unterlegten Vortrag zu folgen. Die Videos basieren momentan auf Swift 1.0, es ist aber zu hoffen, dass es Anfang 2016 eine aktualisierte Fassung gibt.

https://itunes.apple.com/us/course/developing-ios-8-apps-swift/id961180099

# 10.1 Projektstart

Um eine App für ein iOS-Gerät, also für ein iPhone oder ein iPad, zu entwickeln, starten Sie in Xcode ein neues Projekt und wählen zuerst den Typ IOS APPLICATION • SINGLE VIEW APPLICATION. Das ist die einfachste Art von iOS-Apps. Anfänglich besteht eine derartige App aus einer einzigen Ansicht, also aus einem »Bildschirm«, aus einer »Seite«. Sie können aber später problemlos weitere Ansichten hinzufügen.

Im nächsten Dialogblatt geben Sie Ihrem Projekt einen Namen und wählen als Programmiersprache SWIFT sowie als Device UNIVERSAL aus. Das bedeutet, dass Sie die App später sowohl auf einem iPhone als auch auf einem iPad ausführen können.

Der Dialog enthält drei Optionen (siehe <u>Abbildung 10.1</u>), die Sie allesamt deaktiviert lassen können und die in diesem Buch nicht weiter behandelt werden:

- ► USE CORE DATA ist nur dann relevant, wenn Sie die Daten Ihrer App in einem Objekt-Relationen-Mapping-Modell speichern möchten.
- ► INCLUDE UNIT TESTS und INCLUDE UI TESTS fügt Ihrem Projekt jeweils ein eigenes Verzeichnis hinzu, in dem Sie Testfunktionen für Ihren Code bzw. für die Benutzeroberfläche unterbringen können.

Product Name:	Hello iOS	
Organization Name:	Michael Kofler	
Organization Identifier:	info.kofler	
Bundle Identifier:	info.kofler.Hello–iOS	
Language:	Swift	
Devices:	Universal	
	Use Core Data	
	Include Unit Tests	
	Include UI Tests	

Abbildung 10.1 Projektoptionen für das Hello-iOS-Projekt

Zuletzt müssen Sie noch das Verzeichnis angeben, in dem Xcode die Code-Dateien des neuen Projekts speichern soll.

Sie brauchen für diese Hello-World-App übrigens noch kein registriertes iOS-Entwicklergerät, sondern können die App im Simulator ausprobieren. Wie Sie Apps auf Ihrem eigenen iPhone oder iPad zum Laufen bringen, ist das Thema von Abschnitt 10.7, »Apps auf dem eigenen iPhone/iPad ausführen«.

# 10.2 Gestaltung der App

Das neue Projekt besteht aus einer ganzen Reihe von Dateien, deren Namen in der linken Spalte von Xcode im Projektnavigator zu sehen sind. Sollte Xcode den Projektnavigator nicht anzeigen, führen Sie VIEWS • NAVIGATOR • PROJECT NAVIGATOR aus oder drücken einfach (\*)+(1). Im Projektnavigator klicken Sie nun auf die Datei Main.storyboard.

Im Hauptbereich von Xcode sehen Sie nun das sogenannte »Storyboard«. Anfänglich besteht dieses Drehbuch für iOS-Apps aus nur einer Szene, also einer quadratischen Bildschirmansicht der App. Bei vielen Apps gesellen sich dazu später weitere Seiten, zwischen denen die Anwender dann navigieren können. Im Hello-World-Beispiel wird es aber bei einer Ansicht bleiben.

## Warum sind Ansichten im Storyboard quadratisch?

Das quadratische Format hat damit zu tun, dass Sie beim Entwurf von Apps losgelöst von den tatsächlichen physikalischen Maßen von iOS-Geräten denken sollen. Ihre App soll ja später auf unterschiedlichen Geräten mit unterschiedlichen Auflösungen laufen, hochkant oder im Querformat. In <u>Abschnitt 11.5</u>, »Auto Layout«, werden Sie lernen, wie Sie durch Regeln eine automatische Anpassung des Layouts Ihrer Apps je nach Größe und Ausrichtung erreichen können.

Nichtsdestotrotz ist es beim Entwurf von Benutzeroberflächen natürlich oft praktisch, zumindest die ungefähre Form des Geräts vor Augen zu haben, für das Sie Ihre App entwickeln. Das ist kein Problem: Klicken Sie zuerst auf die Ansicht, und suchen Sie dann im Attributinspektor nach der Einstellungsgruppe SIMULATED METRICS. Dort haben Sie im Listenfeld Size die Wahl zwischen verschiedenen Display-Größen, z. B. IPHONE 4.7-INCH.

Alternativ können Sie bei den quadratischen Ansichten bleiben und im Assistenzeditor die PREVIEW-Ansicht öffnen. Dort können Sie – einen großen Monitor vorausgesetzt – das Aussehen Ihrer App auf unterschiedlichen iOS-Geräten parallel betrachten.

# Mini-Glossar

Bevor Sie damit beginnen, die Hello-World-App zu gestalten, sollten Sie zumindest die wichtigsten Grundbegriffe aus der iOS-Entwicklerwelt kennen: Eine App kann aus mehreren Ansichten bestehen. Solche Ansichten nennt Xcode **Szenen**. Jede Szene kann diverse Steuerelemente enthalten, die das Aussehen der Ansicht bestimmen. Alle Einstellungen aller Szenen sowie der darin enthaltenen Steuerelemente werden im **Storyboard** gespeichert. Jede Szene wird mit einer Code-Datei verbunden, dem **Controller**. Im Controller legen Sie fest, wie Ihre App auf Ereignisse reagiert. Der Controller bestimmt also, wie sich Ihre App verhalten soll, wenn ein Benutzer einen Button berührt, einen Slider verschiebt etc. Das dem Controller zugrunde liegende Programmiermuster, den *Model-View-Controller*, stelle ich Ihnen in Abschnitt 11.1 näher vor.

Standardmäßig enthält ein neues Projekt vom Typ SINGLE VIEW APPLICATION die Storyboard-Datei Main.storyboard mit der Szene »View-Controller« und dem zugeordneten Controller in der Datei ViewController.swift. Um später weitere Szenen hinzuzufügen, verschieben Sie einen VIEW-CONTROLLER aus der Xcode-Objektbibliothek in das Storyboard. Während im Storyboard beliebig viele Szenen gespeichert werden können, benötigen Sie im Regelfall zu jeder Szene eine eigene Swift-Code-Datei mit dem Controller. Mitunter können Sie auch eine Controller-Klasse mehreren ähnlichen Szenen zuordnen. Wie Sie selbst neue Controller-Klassen einrichten, beschreibt Abschnitt 12.1, »Storyboard und Controller-Klassen verbinden«.

### Vorsicht bei Umbenennungen!

Die grafische Darstellung des Storyboards in Xcode ist sehr ansprechend. Intern handelt es sich beim Storyboard aber um eine simple XML-Datei. Dabei ist es wichtig zu wissen, dass die Referenzen vom Storyboard auf Code-Elemente und Komponenten ausschließlich in Form von Zeichenketten erfolgen.

Oft ist die Versuchung groß, Outlets, Actions, Klassennamen etc. nachträglich zu verändern, wenn sich die ursprünglich gewählten Namen sich als irreführend oder zu wenig prägnant erweisen. Damit geraten Sie aber schnell in Teufelsküche. Xcode beklagt sich dann über unbekannte Objekte in den Interface-Builder-Dateien, die aus dem Storyboard generiert werden. Abhilfe schafft das Neueinfügen bzw. Neuverbinden von Actions und Outlets bzw. die Neuzuordnung von Klassen; mitunter ist es aber sehr schwierig, die wahre Ursache des Problems zu finden.

# Steuerelemente einfügen

Die Benutzeroberfläche unser Hello-World-App ist momentan leer. In die quadratische Ansicht sollen nun zwei Bestandteile (Steuerelemente) eingebaut werden: Ein Button mit der Aufschrift HELLO WORLD und ein anfänglich leeres Textfeld. Jedes Mal, wenn Sie den Button HELLO WORLD anklicken, soll im Textfeld eine neue Zeile mit dem aktuellen Datum und der Uhrzeit eingefügt werden (siehe Abbildung 10.2).

Xcode zeigt die zur Auswahl stehenden Steuerelemente normalerweise links unten in der Objektbibliothek an. Sollten die Steuerelemente bei Ihnen nicht sichtbar sein, blenden Sie diesen Bereich der Werkzeugleiste mit VIEW • UTILITIES • SHOW OBJECT LIBRARY bzw. mit (tr1) + (alt) + (w) + (3) ein.

🔘 😑 🔘 Simulator – il	Phone 6 – iPhone 6 / iOS 9.0 (1
Netzbetreiber 훅	09:56
Hello World!	
23.7.2015 9:56:41 23.7.2015 9:56:41 23.7.2015 9:56:45 23.7.2015 9:56:51	

### Abbildung 10.2 Die Hello-World-App im iOS-Simulator



Abbildung 10.3 Das Storyboard mit der einzigen Programmansicht der Hello-World-App

Xcode kann die Objektbibliothek in einer kompakten Icon-Ansicht oder in einer Listenansicht darstellen. Wechseln Sie gegebenenfalls mit dem Button am unteren Rand der Objektbibliothek in die übersichtlichere Listenansicht. Im daneben befindlichen Suchfeld können Sie in der Liste rasch die gewünschten Einträge ermitteln – suchen Sie nach *button* bzw. *text view*.

Klicken Sie zuerst den Button an und verschieben Ihn per Drag & Drop in den quadratischen View-Controller des Storyboards. Platzieren Sie den Button links oben und ändern Sie dann den Text von BUTTON zu HELLO WORLD (siehe <u>Abbildung 10.3</u>). Im Attributinspektor links oben in der Werkzeugleiste stellen Sie die Schrift größer ein. Den Attributinspektor blenden Sie bei Bedarf mit <u>alt</u>+**#**+4 ein.

Nun ist das Text-View-Steuerelement an der Reihe. Verschieben Sie es Steuerelement per Drag & Drop aus der Objektbibliothek in das Storyboard. Platzieren Sie das Steuerelement unterhalb des Buttons und stellen Sie die Größe so ein, dass es circa ein Viertel des quadratischen Storyboards füllt. Wie gesagt, mit den Feinheiten des Layouts beschäftigen wir uns später. Stellen Sie nun die gewünschte Textgröße ein, und löschen Sie den Blindtext »Lorem ipsum ... « im Attributinspektor.

Wenn Sie möchten, können Sie dem Textfeld auch noch eine andere Hintergrundfarbe zuweisen – dann erkennen Sie später bei der Programmausführung besser die tatsächlichen Ausmaße des Steuerelements. Zur Farbeinstellung scrollen Sie im Attributinspektor nach unten, bis Sie im Bereich VIEW das Feld BACKGROUND finden.

# Ein erster Test mit dem iOS-Simulator

Nachdem wir nun die minimalistische Oberfläche unserer App gestaltet haben, spricht nichts dagegen, das Programm ein erstes Mal zu starten. Dazu wählen Sie in der Symbolleiste des Xcode-Fensters zuerst das Gerät aus, das Sie simulieren möchten. Zur Auswahl steht die gängige iOS-Hardware-Palette – Mitte 2015 also iPad 2, Retina und Air sowie iPhone 4 bis 6.

Ein Klick auf den dreieckigen RUN-Button kompiliert das Programm und übergibt es zur Ausführung an den iOS-Simulator (siehe <u>Abbildung 10.2</u>). Dabei handelt es sich um ein eigenständiges Programm, das losgelöst von Xcode läuft. Wie der Name vermuten lässt, simuliert es ein iPhone bzw. iPad, wobei das zu testende Programm zur Ausführung ohne den Umweg über den App Store installiert wird. Der Simulator ist zumindest für erste Tests gut geeignet. Für »echte« Apps geht an Tests auf richtiger Hardware aber natürlich kein Weg vorbei.

### Spracheinstellungen im Simulator

Im Simulator gelten anfänglich englische Spracheinstellungen. Das lässt sich wie auf einem richtigen Gerät in den Systemeinstellungen beheben. Dazu beenden Sie die laufende Hello-World-App durch einen per Menü mit HARDWARE • HOME simulierten Druck auf den iPhone-Button. Anschließend starten Sie die vorinstallierte App SET-TINGS, navigieren in den Dialog GENERAL • LANGUAGE & REGION und stellen dort die Sprache DEUTSCH und die Region GERMANY ein.

Diesen Vorgang müssen Sie für jedes iOS-Gerät wiederholen, das Sie testen möchten. Die Einstellungen werden also nicht über alle simulierten Geräte geteilt.

Da die Veränderung der Spracheinstellungen in allen iOS-Geräten recht umständlich ist, bietet Xcode auch die Möglichkeit, mit PRODUCT • SCHEME • EDIT SCHEME festzulegen, in welcher Sprache die App ausgeführt werden soll. Details dazu und zur Entwicklung mehrsprachiger Apps folgen in Abschnitt 11.8.

Im iOS-Simulator können Sie nun den Button HELLO WORLD! anklicken. Das Programm wird darauf aber nicht reagieren, weil der Button ja noch nicht mit Code verbunden ist. Wenn Sie das Textfeld anklicken, können Sie über die Tastatur Ihres Computers sowie über die im Simulator eingeblendete Bildschirmtastatur Text eingeben. Die Test-App läuft, bis Sie sie in Xcode mit dem Stopp-Button beenden.

### Ärger mit der Tastatur

Sie können im iOS-Simulator mit der Tastatur Ihres Computers Eingaben durchführen. Das ist effizient, aber nicht realitätsnah: Der Simulator betrachtet Ihre Tastatur als mit dem iOS-Gerät verbunden und blendet deswegen die iOS-Bildschirmtastatur aus. Mit dem Kommando HARDWARE • KEYBOARD • CONNECT HARDWARE KEYBOARD im iOS-Simulator können Sie umstellen, ob Sie Ihre Computertastatur oder die Bildschirmtastatur von iOS verwenden möchten.

Erscheint die iOS-Bildschirmtastatur einmal im iOS-Simulator, tritt ein weiteres Problem auf: Es scheint keine Möglichkeit zu geben, die Tastatur wieder loszuwerden. Das ist das in iOS vorgesehene Verhalten – wenn die Eingabe abgeschlossen ist, muss die Tastatur per Code explizit ausgeblendet werden, in der Regel durch die Anweisung view.endEditing(true). Im Hello-World-Projekt verzichten wir darauf; konkrete Beispiele finden Sie aber in den meisten größeren Beispielprojekten dieses Buches – zum ersten Mal in <u>Abschnitt 12.4</u>, »Tastatureingaben mit Delegation verarbeiten«.

# 10.3 Steuerung der App durch Code

Damit das Berühren bzw. im Simulator das Anklicken des Buttons eine sichtbare Wirkung zeigt, soll jedes Mal in das Textfeld eine Zeile mit dem Datum und der aktuellen Uhrzeit hinzugefügt werden. Das erfordert einige Zeilen eigenen Swift-Code.

## Den Button mit einer Methode verbinden (Actions)

Mit der App-Ansicht (»Szene«) verbundener Code muss nun in die bereits vorgesehene Datei ViewController.swift eingetragen werden. Xcode unterstützt uns dabei, wenn wir vorher das Storyboard sowie ViewController.swift nebeneinander anzeigen. Genau das bewirkt der Button mit den »Hochzeitsringen«: Er blendet zur gerade aktiven Datei den sogenannten Assistenzeditor ein, also eine zugeordnete zweite Datei.

### Platz sparen in Xcode

Wenn Sie nicht gerade auf einem sehr großen Monitor arbeiten, wird der Platz in Xcode jetzt knapp. Links sitzt der Projektnavigator, in der Mitte das Storyboard und der Code des Controllers und rechts noch die Werkzeugleiste – das ist zu viel! Blenden Sie deswegen vorübergehend die beiden Seitenleisten aus. Die entsprechenden Buttons finden Sie rechts oben in der Xcode-Symbolleiste.

ViewController.swift enthält den Swift-Code für eine Klasse mit dem Namen View-Controller. Diese Klasse ist von der UIViewController-Klasse abgeleitet. Wie der Name vermuten lässt, steuert diese Klasse das App-Erscheinungsbild. Der View-Controller enthält bereits zwei leere Methoden. Diese sind für uns vorerst aber nicht von Interesse; wenn Sie möchten, können Sie sie löschen.

Dafür möchten wir nun eine Methode in den Code einbauen, die ausgeführt wird, wenn der Button HELLO WORLD angeklickt wird. Dazu drücken Sie die Taste <u>ctrl</u> und ziehen den Button vom Storyboard in den Assistenzeditor. Die Methode soll direkt in der Klasse, *nicht* innerhalb einer anderen Methode eingefügt werden. Achten Sie darauf, die Maus- bzw. Trackpad-Taste an der richtigen Stelle loszulassen! Bis dahin markiert eine blaue Linie die Verbindung, die Sie herstellen möchten.

Sobald Sie die Maus bzw. das Trackpad losgelassen haben, erscheint ein Dialog, in dem Sie die Details der Verbindung einstellen können (siehe <u>Abbildung 10.4</u>). Für uns sind folgende Einstellungen zweckmäßig:

 CONNECTION = ACTION: Wir wollen auf ein Ereignis reagieren. Der Begriff »Action« bezieht sich dabei auf das Target-Action-Entwurfsmuster, bei dem eine Methode in die eigene Controller-Klasse eingebaut wird. Sie gilt als »Ziel«, das angesprungen wird, wenn das entsprechende Ereignis auftritt.

- NAME: Hier geben wir der zu erstellenden Methode einen möglichst aussagekräftigen Namen. Ich habe mich bei diesem Beispiel für hwButtonTouch entschieden.
- ► TYPE = ANYOBJECT oder UIBUTTON: Diese Einstellung gibt an, in welcher Form Daten an die Methode übergeben werden sollen. Im Hello-World-Beispiel werten wir diese Daten aber ohnehin nicht aus – insofern ist die Einstellung egal. Wollten wir den Text des Buttons oder andere Eigenschaften auslesen, wäre es zweckmäßig, hier UIBUTTON einzustellen.
- EVENT = TOUCH UP INSIDE: Hier wird festgelegt, auf welches Ereignis wir reagieren möchten. Die Defaulteinstellung passt hier: Wenn der Button innen berührt wird, soll unser Code ausgeführt werden.
- ARGUMENTS = NONE: In der Auswahlliste können Sie festlegen, welche Daten an die Methode übergeben werden sollen. Zur Auswahl stehen SENDER, SENDER AND EVENT oder eben NONE. Wir haben nicht vor, diese Daten auszuwerten – insofern reicht NONE aus.



**Abbildung 10.4** Der Button des Storyboards (links) wird mit einer neuen Methode im ViewController-Code (rechts) verbunden.

Die hier so langatmig beschriebenen Einstellungen nehmen Sie mit etwas Übung in gerade mal 15 Sekunden vor. Xcode belohnt uns diese Arbeit mit einer aus zwei Zeilen Code bestehenden Definition einer Methode:

```
// Projekt hwios-world, Datei ViewController.swift
@IBAction func hwButtonTouch(sender: UIButton) {
    // todo: Code verfassen
}
```

### Der Code alleine reicht nicht!

Angesichts des doch recht umständlichen Mausgeklickes liegt es nahe, den wenigen Code einfach selbst einzutippen. Das ist aber keine gute Idee! Mit dem Code an sich ist es nämlich nicht getan. Hinter den Kulissen merkt sich Xcode auch, mit welchem Steuerelement und mit welchem Ereignis die Methode verknüpft ist. Diese Metadaten landen in der XML-Datei Main.storyboard. Die Verbindung stellt Cocoa Touch dynamisch zur Laufzeit her. Das kompilierte Storyboard enthält dafür eine Referenz auf den serialisierten View-Controller und den Selektor auf die Methode.

# Zugriff auf das Textfeld über eine Eigenschaft (Outlets)

Bevor wir die Methode hwButtonTouch mit konkretem Code füllen können, benötigen wir noch eine Zugriffsmöglichkeit auf das Textfeld. Dazu stellen wir eine weitere Verbindung zwischen der App-Ansicht im Storyboard und dem Code her: Wieder mit [ctrl] ziehen wir diesmal das Text-View-Steuerelement in den Codebereich, verwenden diesmal aber andere Einstellungen:

- CONNECTION = OUTLET: Wir wollen auf das Steuerelement zugreifen können. Xcode soll also eine entsprechende Eigenschaft in die ViewController-Klasse einbauen, die auf das Objekt verweist. In der iOS-Nomenklatur wird das als »Outlet« bezeichnet.
- ► NAME: Das ist der Name, unter dem wir das Steuerelement ansprechen möchten. Ich habe textView angegeben.
- ► TYPE = UITEXTVIEW: Das Steuerelement ist eine Instanz der UITextView-Klasse und unter diesem Typ möchten wir auf das Steuerelement auch zugreifen.
- STORAGE = WEAK: Hier geht es darum, ob die Referenz auf das Steuerelement sicherstellt, dass dieses im Speicher bleibt. Das ist nicht notwendig: Das Steuerelement kommt uns sicher nicht abhanden. Insofern können wir die Defaulteinstellung WEAK bedenkenlos übernehmen.

Die resultierende Code-Zeile sieht so aus:

```
@IBOutlet weak var textView: UITextView!
```

Damit ist also eine neue Eigenschaft (Klassenvariable) mit dem Namen textView definiert. Wir können damit auf ein Steuerelement vom Typ UITextView zugreifen, wobei der Typ als Optional angegeben ist. Der Grund dafür besteht darin, dass die Initialisierung des Steuerelements möglicherweise nicht sofort beim App-Start, sondern erst etwas später erfolgt. Indem die Eigenschaft als Optional deklariert ist, wird klar, dass ein Zugriff auf das Steuerelement unter Umständen noch gar nicht möglich ist.

### Die Attribute »@IBAction« und »@IBOutlet«

Sicher ist Ihnen aufgefallen, dass Xcode die Methode mit dem Attribut @IBAction und die Eigenschaft mit dem Attribut @IBOutlet gekennzeichnet hat. Der Interface Builder erkennt anhand dieser Attribute, zu welchen Elementen der Programmierer Verbindungen erlaubt bzw. herstellen darf. Der Interface Builder ist jener Teil von Xcode, der zur Gestaltung grafischer Benutzeroberflächen für iOS-Apps und OS-X-Programme dient.

### **Endlich eigener Code**

Sie können die App zwischenzeitlich noch einmal starten. An ihrem Verhalten hat sich nichts verändert. Die Methode hwButtonTouch ist ja noch leer, und auch die zusätzliche Eigenschaft textView ist noch ungenutzt. Aber das ändert sich jetzt endlich! Jedes Mal, wenn der Button HELLO WORLD berührt wird, soll das Textfeld um eine Zeile mit Datum und Uhrzeit ergänzt werden. In einer ersten Testversion könnte der Code wie folgt aussehen:

```
// Projekt ios-hello-world, Datei ViewController.swift
import UIKit
class ViewController: UIViewController {
    // Zugriff auf das Textfeld
    @IBOutlet weak var textView: UITextView!
    // Methode zur Reaktion auf einen Button-Touch
    @IBAction func hwButtonTouch() {
        let now = NSDate()
        let formatter = NSDateFormatter()
        formatter.dateFormat = "d.M.yyyy H:mm:ss"
        textView.text! +=
        formatter.stringFromDate(now) + "\n"
    }
}
```

Auf den Abdruck der standardmäßig vorhandenen, aber leeren Methoden viewDid-Load und didReceiveMemoryWarnung habe ich verzichtet. In diesem Beispiel können Sie diese Methoden aus dem Code löschen, wir brauchen sie nicht.

Outlets wie die Eigenschaft textView sind Optionals. Es liegt in der Natur von Optionals, dass diese den Zustand nil haben können. Tatsächlich ist dies aber nur vor der Initialisierung der Ansicht der Fall. Sobald die Ansicht auf einem iOS-Gerät sichtbar wird und Action-Methoden ausführen kann, können Sie sich darauf verlassen, dass Outlet-Eigenschaften nicht nil sind. Genau genommen gilt dies ab dem Zeitpunkt, zu dem im View-Controller die Methode viewDidLoad ausgeführt wurde – siehe Abschnitt 11.3, »Die UIViewController-Klasse«.

Hintergrundinformationen zum Umgang mit Datum und Uhrzeit können Sie bei Bedarf in <u>Abschnitt 3.4</u> nochmals nachlesen. Somit bleibt nur noch der Ausdruck textView.text! zu erklären: Ein Blick in die Dokumentation des UITextView-Steuerelements zeigt, dass der Inhalt des Textfelds über die Eigenschaft text gelesen und verändert werden kann. Der Datentyp von text lautet String?, es handelt sich also um ein Optional, das beim Zugriff explizit durch das nachgestellte Ausrufezeichen ausgepackt werden muss.

### Der schnellste Weg zur Dokumentation

Wenn Sie zum ersten Mal ein UITextView-Steuerelement nutzen, kennen Sie die Elemente dieser Klasse noch nicht. Eine kurze Beschreibung der Klasse erhalten Sie, wenn Sie das Schlüsselwort UITextView im Code zusammen mit <u>alt</u> anklicken. Am unteren Rand der eingeblendeten Infobox befindet sich ein Link auf die Klassenreferenz, die in einem eigenen Fenster geöffnet wird (siehe <u>Abbildung 10.5</u>).



Abbildung 10.5 Der Hilfebrowser von Xcode

Jetzt ist es höchste Zeit, die App im Simulator endlich auszuprobieren. Einige Klicks auf den Button beweisen, dass das Programm wie erwartet funktioniert (siehe <u>Abbildung 10.2</u>). Einen Preis für innovatives Layout wird es freilich nicht gewinnen.

# 10.4 Actions und Outlets für Fortgeschrittene

Sie haben nun eine erste Vorstellung davon, wie Actions und Outlets funktionieren. Dieser Abschnitt weist auf einige Besonderheiten im Umgang mit Actions und Outlets hin.

### Eine Action für mehrere Steuerelemente

Es ist zulässig, ein und dieselbe Action-Methode für mehrere Steuerelemente zu verwenden. Dazu richten Sie zuerst für ein Steuerelement die Methode ein. Danach führen Sie eine <u>ctrl</u>-Drag-Operation für das zweite Steuerelement aus, wobei Sie als Ziel die bereits vorhandene Methode verwenden. Achten Sie darauf, dass die gesamte Methode blau unterlegt wird – dann hat Xcode erkannt, dass Sie nicht eine Action *in* der Methode einfügen möchten (das funktioniert nicht), sondern dass Sie das Steuerelement mit der vorhandenen Methode verbinden möchten.

Naturgemäß müssen Sie die Methode nun ein wenig modifizieren: Erst mit einer Auswertung des sender-Parameters können Sie erkennen, welches Steuerelement den Aufruf der Methode ausgelöst hat. Wenn Sie beispielsweise mehrere Buttons mit einer Methode verbunden haben, können Sie wie folgt den Text des Buttons ausgeben:

```
@IBAction func btnAction(sender: UIButton) {
   print(sender.currentTitle!)
}
```

### Ein Outlet für mehrere Steuerelemente (Outlet Collections)

Auch der umgekehrte Fall ist möglich – Sie können mehrere Steuerelemente über ein Outlet ansprechen. Genau genommen handelt es sich dann nicht mehr um ein einfaches Outlet, sondern um eine »Outlet Collection«. Dazu markieren Sie das erste Steuerelement, ziehen es mit <u>ctrl</u>-Drag in den Controller-Code und wählen CONNECTION = OUTLET COLLECTION (siehe <u>Abbildung 10.6</u>). Xcode erzeugt anstelle einer einfachen Outlet-Variablen nun ein Array:

```
@IBOutlet var allButtons: [UIButton]!
```

In der Folge verbinden Sie auch die weiteren Steuerelemente durch <u>ctrl</u>-Drag mit diesem Array. Xcode ist leider nicht in der Lage, mehrere markierte Steuerelemente auf einmal zu verbinden. Im Code können Sie nun unkompliziert Schleifen über alle so verbundenen Steuerelemente bilden.

Connection	Outlet Collection 0
Object	View Controller
Name	allButtons
Туре	UIButton
Cancel	Connect

Abbildung 10.6 Dialog zum Einrichten einer Outlet-Collection

### Actions oder Outlets umbenennen

Wenn Sie im Code Actions oder Outlets einfach umbenennen, funktioniert Ihre App nicht mehr. Xcode merkt sich die Verknüpfung zu Ihren Methoden bzw. Eigenschaften in Form von Zeichenketten. Nach der Umbenennung ist keine korrekte Zuordnung mehr möglich.

Abhilfe: Klicken Sie im Storyboard-Editor das betreffende Steuerelement mit der rechten Maus- oder Trackpad-Taste an. Ein Kontextmenü zeigt nun alle Zuordnungen zu Actions oder Outlets (siehe <u>Abbildung 10.7</u>). Dort löschen Sie die Action- oder Outlet-Verknüpfung, die Sie umbenannt haben. Sollten Sie das vergessen, wird die App abstürzen, wobei die Fehlermeldung *unrecognized selector* lautet.

0	Hello W	orld!		
V	Triggered Segues			
	action		0	
V	Outlet Collections			
	gestureRecognizers		0	
v	Sent Events			
	Did End On Exit		0	
	Editing Changed		ō	
	Editing Did Begin		Õ	
	Editing Did End		Ó	
	Touch Cancel		0	
	Touch Down		0	
	Touch Down Repeat		0	
	Touch Drag Enter		0	
	Touch Drag Exit		0	
	Touch Drag Inside		0	
	Touch Drag Outside		0	
	Touch Up Inside	View Controller hwButtonTouch	0	
	Touch Up Outside		0	
	Value Changed		0	
V	Referencing Outlets			
	New Referencing Outlet		0	
V	<b>Referencing Outlet Collectio</b>	ns		
	New Referencing Outlet Collection		0	
				1

**Abbildung 10.7** Die rechte Maus- oder Trackpad-Taste führt in eine Liste aller Actions und Outlets eines Steuerelements.

Anschließend wiederholen Sie die <u>ctrl</u>-Drag-Operation und ziehen die Verknüpfungslinie direkt zur schon vorhandenen Methode oder Eigenschaft. Damit wird die Verknüpfung wieder neu hergestellt, und Sie müssen nicht alle Einstellungen wiederholen.
#### Steuerelemente kopieren

Wenn Sie Steuerelemente mit 🕷 + 🕻 und 🕷 + 🕅 kopieren, werden dabei alle möglichen unsichtbaren Attribute und Eigenschaften mitkopiert, unter anderem zugeordnete Actions. Oft erspart Ihnen das eine wiederholte Einstellung dieser Merkmale, aber mitunter führt dieses Verhalten zu unerwarteten Nebenwirkungen. Ein Klick auf das Steuerelement mit der rechten Maus- oder Trackpad-Taste offenbart alle zugeordneten Outlets und Actions.

## 10.5 Layout optimieren

Es ist Ihnen sicher aufgefallen, dass das Layout unserer App, also die Anordnung und Größe der Steuerelemente, verbesserungswürdig ist. Die Größe der Steuerelemente ist willkürlich. Wenn Sie die App in verschiedenen iOS-Geräten ausprobieren, werden Sie feststellen, dass teilweise große Teile des Bildschirms ungenutzt bleiben, während das Textfeld bei anderen Geräten sogar abgeschnitten und unvollständig dargestellt wird. Besonders deutlich werden die Layoutdefizite, wenn Sie den iOS-Simulator mit HARDWARE • ROTATE in das Querformat drehen.

Wie würden unsere Layoutwünsche denn aussehen?

- Der Button soll ohne unnötige Abstände links oben im Bildschirm dargestellt werden.
- Das Textfeld soll darunter platziert sein.
- ► Es soll die gesamte verbleibende Größe des Bildschirms nutzen.
- ► Es soll seine Größe bei einer Drehung des Geräts automatisch anpassen.

Momentan wird unser Programm diesen Wünschen deswegen nicht gerecht, weil wir die Position und Größe der Steuerelemente absolut festgelegt haben. Wir haben die Steuerelemente im View weitgehend nach Gutdünken platziert.

#### Layoutregeln

Die Lösung, die Xcode bzw. eigentlich das UIKit, also das Framework zur iOS-Programmierung, hierfür anbietet, heißt Layoutregeln (Constraints). Sie können also für jedes Steuerelement Regeln aufstellen, die dieses einhalten soll. Das UIKit bemüht sich dann, in Abhängigkeit von der gerade vorliegenden Form und Größe des iOS-Geräts, allen Regeln gerecht zu werden. Beispiele für derartige Regeln sind:

 Der horizontale Abstand zwischen dem Steuerelement A und seinem nächstgelegenen linken oder rechten Nachbar soll 8 Punkte betragen. Bei Retina-Geräten mit doppelter Auflösung entspricht das 16 Pixeln.

- ► Der vertikale obere Abstand zwischen dem Steuerelement A und dem Bildschirmrand soll 16 Punkt betragen.
- ► Steuerelement A soll genauso breit sein wie Steuerelement B.
- Steuerelement A soll innerhalb seines Containers vertikal und/oder horizontal zentriert werden.
- ► Die linken Ränder der Steuerelemente A, B und C sollen in einer Linie verlaufen.

Wenn wir also erreichen möchten, dass die Steuerelemente der Hello-World-App wie oben formuliert angeordnet werden, müssen wir nur die entsprechenden Regeln formulieren. Mit etwas Erfahrung gelingt dies rasch, gerade Einsteiger in die iOS-Programmierung scheitern aber oft an der damit verbundenen Komplexität.

## Layoutregeln für den »Hello-World«-Button

Eine detaillierte Erklärung der Layoutregel erhalten Sie in <u>Abschnitt 11.5</u>. An dieser Stelle möchte ich Ihnen nur rezeptartig erklären, wie Sie das Layout der Hello-World-App korrekt einstellen. Dazu klicken Sie im Storyboard zuerst auf den Button HELLO WORLD, dann auf den Button PIN, der sich rechts unten im Editor befindet (siehe Abbildung 10.8).

에 미 의 탱

Stack View

- \rm 🛛 Align
- 머 Pin
- **A** Resolve Auto Layout Issues

**Abbildung 10.8** Die vier winzigen Layout-Button befinden sich rechts unten im Storyboard-Editor.

Damit erscheint der Dialog ADD NEW CONSTRAINTS zur Einstellung diverser Abstände (siehe <u>Abbildung 10.9</u>). Dort klicken Sie zuerst die Verbindungsstege für die Abstände nach oben bzw. zur linken Seite an, so dass diese Stege durchgängig rot angezeigt werden. Anschließend geben Sie für den Abstand nach oben 8 Punkte und für den seitlichen Abstand O Punkte an. Standardmäßig ist die Option CONSTRAIN TO MARGINS aktiv. Sie bewirkt, dass diese Abstände relativ zu einem vom jeweiligen iOS-Gerät vorgegebenen Standardrahmen gerechnet werden. Zuletzt schließen Sie den Dialog mit dem Button ADD 2 CONSTRAINTS. Damit werden zwei neue Regeln zur Positionierung des Buttons festgelegt.



Abbildung 10.9 Layoutregeln für den Hello-World-Button festlegen

Überraschenderweise führen die neuen Regeln nicht zu einer Veränderung der Position des Steuerelementes. Vielmehr zeigen orange strichlierte Linien an, wo der Button bei der Programmausführung platziert wird (siehe <u>Abbildung 10.10</u>). Außerdem visualisieren zwei orange Stege die von Ihnen aufgestellten Regeln. Aus den Zahlenwerten geht hervor, um wie viele Punkte das Element momentan falsch positioniert ist.



Abbildung 10.10 Der Storyboard-Editor zeigt, wo der Button später platziert wird.

Um den Button den neuen Regeln entsprechend zu positionieren, klicken Sie auf den Button RESOLVE AUTO LAYOUT ISSUES und führen dort das Kommando SELECTED VIEWS • UPDATE FRAMES aus.

## Layoutregeln für das Textfeld

Nun ist das Textfeld an der Reihe: Nachdem Sie dieses angeklickt haben, öffnen Sie wieder mit dem PIN-Button den Dialog ADD NEW CONSTRAINTS. Dort stellen Sie die folgenden Abstände ein (siehe Abbildung 10.11):

- ► Links: O Punkte. Dieser Abstand gilt wegen der Option CONSTRAIN TO MARGINS relativ zum Standardrahmen.
- ► Oben: O Punkte. Dieser Abstand wird relativ zum nächstgelegenen Steuerelement gerechnet, in diesem Fall also zum Button.
- ▶ Rechts: O Punkte. Dieser Abstand gilt wieder relativ zum Standardrahmen.
- Unten: 16 Punkte. Auch dieser Abstand gilt relativ zum Standardrahmen. Dieser sieht nach unten aber keinen Rand vor. Damit das Textfeld von allen Rändern gleich weit entfernt ist, muss hier ein etwas größerer Wert angegeben werden.

Add New Constraints	-	
0	•	
	H 0 -	
16	v	
Spacing to near	est neighbor	
Constrain t	to margins	
Width	568	*
🗌 🔟 Height	517	*
🗌 📳 Equal Widths		
🔲 💷 Equal Heights		
🗌 🗧 Aspect Ratio		
🗆 🔠 Align ( Leading Ed	lges	
Update Frames None	lges	0

Abbildung 10.11 Layoutregeln für das Textfeld

ADD 4 CONSTRAINTS beendet die Eingabe und fügt vier Regeln hinzu. Um auch das Textfeld gemäß der neuen Regeln korrekt zu platzieren, klicken Sie nochmals auf den Button RESOLVE AUTO LAYOUT ISSUES und führen dort das Kommando SELECTED VIEWS • UPDATE FRAMES aus.

Damit sollten nun beide Steuerelemente in der quadratischen View korrekt angeordnet sein. Starten Sie die App, um zu testen, ob die Regeln die gewünschte Wirkung zeigen. Testen Sie das Programm im Simulator auch mit anderen iOS-Geräten sowie im Querformat (siehe <u>Abbildung 10.12</u>). Sie werden sehen, dass sich das Programm nun in jeder Situation korrekt verhält.

0.0	Simulator - iPhone 6 Plus - iPhone 6 Plus / iOS 9.0 (13A4305g)
Hello World!	
23.7.2015 10:24:28 23.7.2015 10:24:29 23.7.2015 10:24:29 23.7.2015 10:24:33	

Abbildung 10.12 Die Hello-World-App im iPhone-6-Plus-Simulator im Querformat

## Wenn es Probleme gibt

Der Umgang mit Layoutregeln ist schwierig und führt häufig dazu, dass sich Xcode über fehlende oder über zueinander im Konflikt stehende Regeln beklagt. In <u>Abschnitt 11.5</u>, »Auto Layout«, folgt eine Menge weiterer Details zu diesem Thema. Bis dahin vertröste ich Sie hier mit einigen Tipps:

► Sobald Sie *eine* Regel für ein Steuerelement festlegen, müssen Sie die Größe und Position des Steuerelements *vollständig* durch Regeln bestimmen. Mit anderen Worten: Solange es gar keine Regeln gibt, betrachtet Xcode das Steuerelement als unbestimmt und meckert nicht. Sobald Sie aber beginnen, Regeln festzulegen, müssen Sie dies so tun, dass keine Unklarheiten verbleiben.

Die Anzahl der erforderlichen Regeln ist nicht bei jedem Steuerelement gleich. Manche Steuerelemente können ihre optimale Größe aus dem Inhalt selbst ermitteln. Das trifft z. B. bei einem Button zu. Hier reichen also Regeln, die die Position festlegen. Bei anderen Steuerelementen müssen Sie die Größe selbst einstellen – und das erfordert oft zwei weitere Regeln.

- Sie können vorhandene Regeln nicht ohne Weiteres ändern. Die Dialoge ADD NEW ALIGNMENT CONSTRAINTS bzw. ADD NEW CONSTRAINTS ersetzen bzw. verändern nicht vorhandene Regeln, sondern definieren zusätzliche Regeln. Das führt oft zu Regeln, die sich widersprechen. Einen Überblick über alle Regeln erhalten Sie, wenn Sie die Seitenleiste des Storyboard-Editors einblenden (EDITOR • SHOW DOCUMENT OUTLINE). Dort finden Sie eine Liste aller CONSTRAINTS. Wenn Sie eine der Regeln anklicken, wird die betreffende Regel markiert.
- Bei kleinen Projekten ist es bei Problemen oft am einfachsten, alle Regeln zu löschen und noch einmal von vorne zu beginnen. Dazu klicken Sie auf den Button RESOLVE AUTO LAYOUT ISSUES und führen ALL VIEWS IN VIEW CONTROLLER • CLEAR CONSTRAINTS aus.

# 10.6 Textgröße mit einem Slider einstellen

Als letzte Erweiterung für das Programm fügen wir diesem nun neben dem Button noch einen Slider hinzu, mit dem die Schriftgröße des Textfelds verändert werden kann (siehe <u>Abbildung 10.13</u>).



Abbildung 10.13 Die Hello-World-App mit einem Slider zur Einstellung der Textgröße

## Das Slider-Steuerelement hinzufügen

Sie finden das Steuerelement in der Xcode-Objektbibliothek unter dem Namen *Slider*. Zur Positionierung stellen Sie zuerst mit dem PIN-Button die horizontalen Abstände ein: Der linke Abstand zum HELLO WORLD-Button solle 24 Punkte betragen, der rechte Abstand zum Rand O Punkte. Definieren Sie aber keine Regeln für die vertikale Position!

In einem zweiten Schritt markieren Sie nun mit 💿 sowohl den HELLO WORLD-Button als auch den Slider. Mit dem ALIGN-Button öffnen Sie dann den Dialog NEW ADD ALIGN CONSTRAINTS und wählen dort die Option VERTICAL CENTERS. Diese Regel bewirkt, dass der Button und der Slider vertikal mittig angeordnet werden, was in diesem Fall harmonisch aussieht. Über den Button RESOLVE AUTO LAYOUT ISSUES führen Sie nun das Kommando SELECTED VIEWS • UPDATE FRAMES aus, damit der Slider im Storyboard-Editor an der richtigen Stelle angezeigt wird. Mit dem Slider soll die Textgröße in einem Bereich zwischen 12 und 30 Punkt verändert werden. Dazu wählen Sie den Slider aus und stellen im Attributinspektor die folgenden Eigenschaften ein:

- Minimum: 12
- ► Maximum: 30
- Current: 16

## Den Slider mit einer Methode verbinden

Damit sind die Arbeiten an der Oberfläche abgeschlossen, und wir können uns wieder der Programmierung zuwenden: Öffnen Sie den Assistenz-Editor, und verschieben Sie den Slider mit <u>ctrl</u> in den Code-Bereich. Die Verbindungsparameter stellen Sie wie folgt ein:

- CONNECTION = ACTION: Wir wollen in einer Methode auf das Verschieben des Sliders reagieren.
- ► NAME: Die Methode muss einen Namen bekommen. Ich habe mich für sliderMove entschieden.
- ► TYPE = UISLIDER: In der Methode müssen wir die aktuelle Position des Sliders herausfinden. Deswegen soll die Instanz des Sliders an die Methode übergeben werden.
- EVENT = VALUE CHANGED: Die Defaulteinstellung passt hier gut andere Ereignisse interessieren uns nicht.
- ARGUMENTS = SENDER: Damit wird die Instanz des Sliders als Parameter an die Methode übergeben. Den Datentyp des Parameters haben wir ja bereits mit UISlider festgelegt.

Zu Testzwecken bauen wir in die Methode vorerst nur die print-Funktion ein, um eine Veränderung des Sliders in Xcode verfolgen zu können. Uns interessiert die value-Eigenschaft, die die Slider-Position im eingestellten Wertebereich als Fließkommazahl liefert.

```
// wird bei jeder Slider-Bewegung ausgeführt
@IBAction func sliderMove(sender: UISlider) {
    // Testausgabe
    print(sender.value)
}
```

Jetzt geht es nur noch darum, die Schrift des Textfelds entsprechend zu verändern. Dazu lesen wir mit textView.font? die aktuelle Font-Instanz aus, bilden daraus mit der Methode fontWithSize eine neue Instanz in der gewünschten Größe und weisen diese der font-Eigenschaft des Textfelds wieder zu. Da fontWithSize einen CGFloatParameter erwartet, muss die Fließkommazahl von sender.value in den CGFloat-Typ umgewandelt werden. CGFloat ist auf 32-Bit-Architekturen ein Float, auf 64-Bit-Architekturen aber ein Double.

```
@IBAction func sliderMove(sender: UISlider) {
  if textView != nil {
    // neue Font-Instanz in der gewünschten
    // Größe erzeugen
    textView.font =
       textView.font?.fontWithSize(CGFloat(sender.value))
  }
}
```

# 10.7 Apps auf dem eigenen iPhone/iPad ausführen

Den iOS-Simulator in Ehren, aber natürlich wollen Sie Ihre Programme auch auf »richtiger« Hardware testen. Seit Mitte 2015 steht diese Testmöglichkeit erfreulicherweise kostenlos zur Verfügung. Dazu verbinden Sie im Dialog PREFERENCES • ACCOUNTS Xcode mit Ihrer Apple ID. Außerdem muss Ihr iOS-Gerät durch ein USB-Kabel mit dem Computer verbunden sein.

Nach diesen Vorbereitungsarbeiten können Sie das iOS-Gerät in der Symbolleiste von Xcode auswählen. Xcode beklagt sich anfänglich darüber, dass ein »Provisioning Profile« fehlt. Das ist eine Sammlung von Schlüsseln, die das Gerät mit Ihrer Apple-ID verbindet. Xcode kann dieses Problem zum Glück selbstständig lösen – Sie müssen nur den Button FIX ISSUE anklicken.

Wenn Sie 💌 + R drücken bzw. den RUN-Button anklicken, überträgt Xcode nun die Hello-World-App auf das iPhone oder iPad und startet sie dort. Das gelingt nur, wenn Ihr Smartphone oder Tablet entsperrt ist. Wenn also eine Ziffern- oder Fingerabdruck-Sperre aktiv ist, müssen Sie das Gerät zuerst einschalten, bevor Sie Ihre App in Xcode starten. Bemerkenswert ist, dass trotz der externen Programmausführung die Debugging-Funktionen von Xcode aktiv bleiben. Wenn Sie also z. B. einen Breakpoint setzen, wird die App an dieser Stelle angehalten. Sie können in Xcode den Zustand der Variablen ergründen und das Programm dann wieder fortsetzen.

Die App bleibt jetzt auf dem iPhone oder iPad. Sie kann dort losgelöst von Xcode ausgeführt werden – dann aber ohne Debugging-Möglichkeiten. Sie können Ihre App wie jede andere installierte App problemlos wieder löschen, indem Sie sie zuerst länger anklicken und dann auf das x-Symbol drücken.

#### Einschränkungen des Free Provisioning

Apple bezeichnet das Verfahren zum Ausführen von Apps auf iOS-Geräten ohne Apple-Developer-Account als »Free Provisioning«. Dabei gibt es aber Einschränkungen: Der Test von einigen Zusatzfunktionen erfordert weiterhin einen kostenpflichtigen Apple-Developer-Account. Das gilt z. B. für In-App-Käufe, die Teilnahme am iAD-Netzwerk oder die Verwendung der Apple-Pay-Funktionen.

## Apple Developer Program

Bevor Sie eine App in den App Store hochladen können, müssen Sie Ihre App speziell vorbereiten und signieren (siehe <u>Abschnitt 16.9</u>, »App im App Store einreichen«). Das ist nur mit dem Schlüsselsystem des Apple Developer Program möglich. Die Mitgliedschaft hat auch andere Vorteile – etwa den unkomplizierten Zugang zu Beta-Versionen von iOS, OS X und Xcode, den Zugang zu Entwicklerforen etc.:

#### https://developer.apple.com/programs

Dieses Service-Paket lässt sich Apple mit zurzeit 100 EUR pro Jahr bezahlen. Im Gegensatz zu früher, als es verschiedene Entwicklerprogramme für iOS, OS X und Safari gab, hat Apple diese Programme nun zu einem einzigen verbunden.

#### Eine vorhandene oder eine neue Apple-ID verwenden?

Bevor Sie sich dem Entwicklerprogramm anschließen, müssen Sie sich überlegen, welche Apple-ID Sie hierfür verwenden. Normalerweise spricht nichts gegen Ihre gewöhnliche Apple-ID. Sollten Sie diese ID aber schon im Rahmen von *iTunes Connect* zum Verkauf von Musik oder Büchern nutzen, dann benötigen Sie eine zweite Apple-ID für das Entwicklerprogramm.

Sie können dem Entwicklerprogramm wahlweise als Einzelperson oder als Team beitreten. Als Einzelperson benötigen Sie dazu lediglich eine Kreditkarte. Nach Abschluss des Bezahlprozesses kann es ein paar Minuten dauern, bis Ihr Entwicklerzugang freigeschaltet wird und Sie die entsprechende *Welcome*-E-Mail erhalten.

Nun können Sie in den Xcode-Einstellungen im Dialogblatt Accounts mit Add APPLE ID Ihre Apple-ID mit Xcode verbinden. Von dort gelangen Sie mit VIEW DETAILS in einen weiteren Dialog, in dem Sie Schlüssel generieren können (siehe <u>Abbildung 10.14</u>). Vorerst benötigen Sie lediglich einen Schlüssel zur iOS-Entwicklung (also den Eintrag IOS DEVELOPMENT).

Nach diesen Vorbereitungsarbeiten können Sie nun ein mit einem USB-Kabel angeschlossenes iOS-Gerät in der Symbolleiste von Xcode auswählen. Wie beim Free Provisioning beklagt sich Xcode anfänglich darüber, dass das »Provisioning Profile« fehlt; FIX ISSUE behebt dieses Problem. Das Gerät wird damit in das Entwicklungsprogramm aufgenommen. Insgesamt dürfen Sie pro Jahr maximal 500 Geräte mit Ihrem Konto verbinden: 100 iPhones, 100 iPods, 100 iPads sowie je 100 Apple-TV- und Apple-Watch-Geräte. Einen Überblick über alle mit Ihrem Konto verbundenen iOS-Geräte finden Sie auf der Webseite des Entwicklerprogramms:

Accounts A Xé nts & Colors Text Editing Key Bindings Michael Kofler Apple II Signing Identities Action iOS Development iOS Distribution Mac Development Create Create Mac App Distribution Mac Installer Distribution Create Developer ID Application Create Developer ID Installer Create nt Provisioning Profiles Action XC: \* distr-prov-currency-deleteme distr-prov-currency2 iOSTeam Provisioning Profile: info.kofler.ios-currency-converter iOSTeam Provisioning Profile: info.kofler.ios-currency-converter2 iOSTeam Provisioning Profile: ios-currency-converter Download all Done View Details... + - 183

https://developer.apple.com/account/ios/device

Abbildung 10.14 Verwaltung der Schlüssel des iOS-Entwicklerprogramms in Xcode

# 10.8 Komponenten und Dateien eines Xcode-Projekts

Wenn Sie in Xcode ein neues iOS-Projekt starten, besteht dieses standardmäßig schon aus einer Menge Dateien (siehe <u>Abbildung 10.15</u>) – und dabei bleibt es nicht. Dieser Abschnitt gibt Ihnen einen kurzen Überblick darüber, welche Datei welchen Zweck hat. Detaillierte Erläuterungen zu vielen Dateien folgen dann in den weiteren Kapiteln.

- ► AppDelegate.swift enthält Code zur Verarbeitung von Ereignissen des App-Lebenszyklus (siehe Abschnitt 11.4, »Phasen einer iOS-App«).
- ► LaunchScreen.xib enthält eine spezielle Ansicht der App, die während des Starts als eine Art Willkommensdialog angezeigt wird (siehe <u>Abschnitt 16.6</u>, »Startansicht (Launch Screen)«).

- Images.xcassets dient als Container für die Bilddateien der App. Dazu zählen neben dem Icon der App auch alle anderen Bitmaps, die Sie irgendwann anzeigen möchten. Die Besonderheit von Xcassets-Dateien besteht darin, dass Bitmaps in mehreren Auflösungen gespeichert werden können. Bei der Ausführung verwendet iOS dann automatisch die Datei, die am besten zum Display des iOS-Geräts passt (siehe <u>Abschnitt 12.7</u>, »Bild-Management in Images.xcasset« und Abschnitt 16.7, »App-Icon«).
- Info.plist enthält diverse Projekteinstellungen in Form einer sogenannten Property List (Key-Value-Datei).
- Main.storyboard beschreibt das Aussehen und die Eigenschaften der Ansichten (View-Controller) einer App.
- ViewController.swift enthält den Controller-Code der ersten Ansicht des Storyboards. Für jede weitere Ansicht im Storyboard müssen Sie in der Regel eine weitere Swift-Datei hinzufügen, die eine von UIViewController abgeleitete Klasse definiert (siehe Kapitel 12, »Apps mit mehreren Ansichten«).



Abbildung 10.15 Überblick über die Code-Dateien im Projektnavigator von Xcode

#### Dateien im Navigator verschieben

Sie können die Dateien des Projekts im Navigator verschieben und in Gruppen gliedern. Zwei Dinge sind in diesem Zusammenhang bemerkenswert: Zum einen spielt es für Xcode keine Rolle, in welcher Gruppe die Dateien sich befinden. Xcode findet in jedem Fall alle zum Kompilieren erforderlichen Dateien. Und zum anderen sind Gruppen *keine* Unterverzeichnisse im Projektverzeichnis. Gruppen helfen bei der Organisation der Dateien, haben aber keinen Einfluss darauf, wo Dateien tatsächlich gespeichert werden. Der Projektnavigator ist also kein Abbild des Dateisystems!

#### Weitere Dateien

Bei »richtigen« Apps, die also nicht nur Test- oder Beispielcharakter haben, kommen zu den anfänglich vorhandenen Dateien zumeist viele Dateien hinzu:

- Weitere Code-Dateien bilden die innere Logik Ihres Programms ab, also das Datenmodell gemäß des MVC-Musters (siehe <u>Abschnitt 11.1</u>, »Model-View-Controller (MVC)«).
- Zusätzliche Lokalisierungsdateien enthalten Zeichenketten für die alle Sprachen, in denen die App später ausgeführt werden kann (siehe <u>Abschnitt 11.8</u>, »Mehrsprachige Apps«).
- ► Ja nach Zielsetzung der App sind außerdem Text-, XML-, HTML-, Datenbank- sowie Audio- und Video-Dateien erforderlich. Diese Dateien werden zusammen mit der App ausgeliefert (»Bundle-Dateien«).

## Test- und Produktgruppe

Neben der eigentlichen Projektgruppe, deren Name mit dem Projektnamen übereinstimmt, kann ein Projekt bis zu drei weitere Gruppen aufweisen:

- ▶ projektnameTests und projektnameUITests enthält Code und Einstellungen zum automatisierten Test Ihres Projekts. Das zugrunde liegende XCTest-Framework hat eine ähnliche Zielsetzung wie Unit Tests in anderen Programmiersprachen. In diesem Buch gehe ich darauf allerdings nicht weiter ein.
- Products enthält das kompilierte Programm. Bei der iOS-App-Entwicklung werden die hier enthaltenen Dateien aber selten benötigt, weil die Ausführung von Apps durch Xcode automatisiert ist und eine Weitergabe von Apps an andere Benutzer nur über den App Store möglich ist.

# Kapitel 13 GPS- und Kompassfunktionen

In diesem Kapitel geht es um die Nutzung der GPS- und Kompassfunktionen Ihres Smartphones durch ein Swift-Programm. Ich präsentiere Ihnen den Umgang mit diesen Funktionen stark beispielorientiert, wobei sich die Komplexität der Apps allmählich steigert:

- ▶ In *Hello MapView* lernen Sie das MapView-Steuerelement kennen.
- Das zweite Programm zeigt, wie Sie auf einer Karte die gerade zurückgelegte Wegstrecke grafisch aufzeichnen.
- ► Eine Kompass-App versucht der Apple-eigenen Kompass-App Konkurrenz zu machen. Gleichzeitig lernen Sie hier grundlegende Grafikfunktionen von iOS kennen und erfahren, wie Sie diese in eigene Steuerelemente integrieren.

## 13.1 Hello MapView!

Das MapView-Steuerelement mit dem Klassennamen MKMapView (MK = Map Kit) ermöglicht es Ihnen, mit geringem Aufwand eine App mit Navigationsfunktionen zu erstellen. Grundsätzlich dient dieses Steuerelement dazu, eine Karte darzustellen, optional auch mit Satelittenbildern und – wo verfügbar – in 3D-Ansicht.

## MapKit-Framework

Beim ersten Test des Steuerelements wird Ihre App vermutlich mit der folgenden nichtssagenden Fehlermeldung abbrechen: *Could not instantiate class named MKMapView*. Schuld daran ist, dass Xcode die erforderliche MapKit-Bibliothek nicht in die App einbindet. Dieses Problem lösen Sie, indem Sie zuerst im Projektnavigator Ihr Projekt und anschließend das zugehörige App-Target auswählen. Im Dialogblatt CAPABILITIES aktivieren Sie nun die Funktion MAPS (siehe Abbildung 13.1).

Damit weiß Xcode nun, dass Sie Maps-Funktionen nutzen möchten. Das entsprechende MapKit-Framework wird nun automatisch in die App integriert. Davon überzeugen Sie sich durch einen Blick in das Dialogblatt GENERAL, an dessen Ende Sie den Eintrag LINKED FRAMEWORKS AND LIBRARIES finden (siehe Abbildung 13.2).



Abbildung 13.1 Aktivierung der Maps-Capabilities

••• • • • •	ndexing		
🗖 🖬 Q 🛆 \ominus 🔲 🖻 🗐	🔡 🛛 < 💛 칠 ios-test-gp	s	
<ul> <li>ios-test-gps</li> <li>image: map (i.e., framework)</li> <li>ios-test-gps</li> </ul>	General Capabi	lities Resource Tags Info Build Sett	ings Build Phases Build Rules
<ul> <li>AppDelegate.swift</li> <li>ViewController.swift</li> <li>Main.storyboard</li> </ul>	TARGETS	Add emb	edded binaries here
Images.xcassets           LaunchScreen.xib           Info.plist		+ -	
Products		Linked Frameworks and Libraries Name	Status
		💼 MapKit.framework	Required 🛟
+ 💿 🛛 🖾	+ - 💿	+ -	

Abbildung 13.2 Zusammenstellung der Zusatz-Frameworks der App

#### **Um Erlaubnis fragen**

Wenn Sie Ihre App mit dem MapView-Steuerelement jetzt ausführen, zeigt sie zwar eine Europakarte in großem Maßstab, nicht aber Ihren aktuellen Ort an. Na gut, werden Sie sich denken, dann aktiviere ich eben im Attributinspektor die Option SHOWS USER LOCATION! An sich ist die Idee natürlich richtig, aber nun tritt eine neue Fehlermeldung auf: *Trying to start MapKit location updates without prompting for location authorization. Must call requestWhenInUseAuthorization or requestAlwaysAuthorization first.* 

Was heißt das nun wieder? Sie müssen in Ihrem Programm explizit um die Erlaubnis fragen, auf Ortsdaten des iOS-Geräts zugreifen zu dürfen. Das ist aus zweierlei Gründen notwendig. Zum einen schätzen es viele Smartphone-Anwender nicht, wenn jedes Programm ständig weiß, wo sich das Telefon und in der Regel auch sein Besitzer gerade aufhält, und zum anderen kosten die GPS-Funktionen relativ viel Strom und verkürzen somit die Akkulaufzeit. Deswegen muss der Benutzer bei jeder App dem Zugriff auf Standortdaten zuerst zustimmen.

Dieses »Um-Erlaubnis-fragen« erledigen Sie am einfachsten in der viewDidLoad-Methode des View-Controllers. Dort erzeugen Sie einen CLLocationManager (CL steht hier für »Core Location«) und führen dann die Methode requestWhenInUseAuthorization aus. Falls Ihre App die Position auch im Hintergrund abfragen soll, verwenden Sie stattdessen die Methode requestAlwaysAuthorization.

```
import CoreLocation
class ViewController: UIViewController {
  var locmgr = CLLocationManager()
  override func viewDidLoad() {
    super.viewDidLoad()
    // um Erlaubnis fragen, ob die Ortungsdienste
    // verwendet werden dürfen
    locmgr.requestWhenInUseAuthorization()
  }
}
```



Abbildung 13.3 Darf die App auf Standortdaten zugreifen?

## Info.plist-Einstellungen

In den von Apple gestalteten Erlaubnisdialog wird eine Begründung eingebaut, warum die App diesen Dienst nutzen will (siehe <u>Abbildung 13.3</u>). Einen entsprechenden Eigenschaftseintrag müssen Sie in der Datei Supporting Files/Info.plist mit ADD Row hinzufügen (siehe <u>Abbildung 13.4</u>). Sie dürfen die Zeichenkette mit der Begründung leer lassen, aber Sie müssen den entsprechenden Eintrag in die Property List einfügen, sonst scheitert die Programmausführung mit einer Fehlermeldung!

Die beiden Eigenschaften haben ausufernd lange Namen, wobei es ärgerlicherweise in der Property List keine Vervollständigung gibt. Achten Sie darauf, dass Ihnen kein Tippfehler passiert!

	) 📷 iP6 ios-test-gps   Build ios-test-g	ps: Succeed	ed   Today at 10:07 📃 🖉 ↔ 🔲 🗖	
□ ╦ Q ∆ ⊘ ▥ ⊃ ♬	$\mathbb{R} \mid \langle \rangle \rangle \mid \mathbb{R}$ ios-test-gps $\rangle$ ios-test-	-gps $ angle$ 📄 Inf	fo.plist $ angle$ No Selection	
🔻 💁 ios-test-gps	Key	Туре	Value	
MapKit, framework	Bundle identifier 🛟	String	\$(PRODUCT_BUNDLE_IDENTIFIER)	
The ios-test-ons	InfoDictionary version 🛟	String	6.0	
AnnDelegate swift	Bundle name 🛟	String	\$(PRODUCT_NAME)	
AppDelegate.swift	Bundle OS Type code	String	APPL	
ViewController.swift	Bundle versions string, short	String	1.0	
Main.storyboard	Bundle creator OS Type code	String	7777	
Images.xcassets	Bundle version 🛟	String	1	
📐 LaunchScreen.xib	Application Category	String		\$
Info.plist	Application requires iPhone envir 🛊	Boolean	YES	\$
Products	Maps routing app supported modes	Array	(3 items)	
	NSLocationAlwaysUsageDesc 🛊 🔘 🖨	String	Dieses Programm kann die aktuelle Position nur aufzeichnen, wenn Sie dies erlauben.	
	Required background modes \$\$	Array	(1 item)	
	Launch screen interface file base 💲	String	LaunchScreen	
	Main storyboard file base name	String	Main	
	▶ Required device capabilities ‡ ○ ○	Array	(1 item)	
	Supported interface orientations	Array	(3 items)	
+0	Supported interface orientations ( \$	Array	(4 items)	

Abbildung 13.4 Begründung für die Nutzung der Ortungsdienste

- ► NSLocationWhenInUseUsageDescription für requestWhenInUseAuthorization()
- NSLocationAlwaysUsageDescription für requestAlwaysAuthorization()

## **Erste Tests**

Die Beschreibung der Voraussetzungen für eine erste funktionierende App mit einer MapView waren zwar recht lang, tatsächlich dauert es mit etwas Xcode-Übung aber kaum länger als eine Minute, eine Hello-MapView-App zusammenzuschustern und die zwei erforderlichen Swift-Code-Zeilen einzufügen. Das Ergebnis sieht auf den ersten Blick beinahe wie die Apple-App »Karten« aus. Die App zeigt also eine Landkarte mit der gerade aktuellen Position (siehe <u>Abbildung 13.5</u>).



Abbildung 13.5 Eine Mini-App mit einem MapView-Steuerelement

Den sichtbaren Kartenausschnitt können Sie wie üblich durch Schieben, Zoomen und Drehen verändern. Ansonsten kann unsere Mini-App aber natürlich nicht mit »Karten« mithalten. Such- und Navigationsfunktionen fehlen ebenso wie die Möglichkeit, die Darstellungsform umzuschalten.

Die entsprechenden Einstellungen können Sie vorweg im Attributinspektor durchführen (siehe <u>Abbildung 13.6</u>), oder Sie müssen in der laufenden App Eigenschaften des MKMapView-Steuerelements durch Swift-Code verändern.



Abbildung 13.6 MapView-Eigenschaften im Attributinspektor einstellen

## Kartenfunktionen im iOS-Simulator

Auf den ersten Blick erweckt der iOS-Simulator den Eindruck, als könnten Sie den sichtbaren Kartenausschnitt dort nur verschieben, nicht aber verdrehen oder zoomen.

Das täuscht aber: Sobald Sie <u>alt</u> drücken, erscheinen im Simulator zwei graue Punkte, die zwei Fingern entsprechen. Diese Punkte bewegen sich rund um den Mittelpunkt der Karte. Mit gedrückter Maus- oder Trackpad-Taste können Sie nun eine Zoom- oder Drehbewegung durchführen. Das erfordert anfänglich etwas Übung, funktioniert aber bald schon ganz zufriedenstellend.

Darüber hinaus bietet das Programm die Möglichkeit, verschiedene Bewegungsabläufe zu simulieren. Die entsprechenden Kommandos sind im Menü DEBUG • LOCA-TION versteckt. Trotzdem ist der Test von Programmen mit geografischen Funktionen im Simulator natürlich nur eingeschränkt möglich.

# 13.2 Wegstrecke aufzeichnen

Im folgenden Beispielprogramm geht es um eine App, die nach dem Start die gerade aktuelle Position des Benutzers verfolgt und in der Karte einzeichnet. Wenn Sie die App starten und dann einen Spaziergang unternehmen, wird Ihr Weg also in Form einer roten Linie auf der Karte nachgezeichnet. Gleichzeitig wird im unteren Bildschirmbereich Ihre aktuelle Position, Geschwindigkeit etc. angezeigt (siehe Abbildung 13.7).



Abbildung 13.7 App zur grafischen Aufzeichnung einer Wegstrecke

## Programmaufbau und Auto Layout

Die App besteht aus zwei Steuerelementen, einer MapView und einem Label. Das mehrzeilige Label ist am linken, unteren und rechten Rand fixiert, außerdem ist seine Höhe fix mit 61 Punkten vorgegeben. Das darüber befindliche Map-View-Steuerelement ist am linken, oberen und rechten Rand sowie an der Oberkante des Labels fixiert. Damit füllt es den gesamten freien Bildschirm aus, der nicht vom Label beansprucht wird.

Im Attributinspektor ist die MapView-Darstellung auf STANDARD gestellt. Jede Benutzerinteraktion ist deaktivert (also alle Allows-Optionen), dafür ist die Option SHOWS USER LOCATION aktiv. Im Dialogblatt CAPABILITIES ist neben MAPS diesmal auch der Punkt BACKGROUND MODES für LOCATION UPDATES aktiviert (siehe <u>Abbildung 13.8</u>). Damit läuft das Programm, so es einmal gestartet ist, auch im Hintergrund weiter und protokolliert Positionsdaten. Vergessen Sie diese Einstellung, wird die App, wenn sie nicht mehr aktiv ist, nach einiger Zeit gestoppt und stürzt beim »Wiederaufwachen« ab. Details zu den verschiedenen App-Zuständen (Vordergrund, Hintergrund, Suspended) lesen Sie bitte in Abschnitt 11.4, »Phasen einer iOS-App«, nach.



Abbildung 13.8 Die Test-App nutzt die abgebildeten iOS-Funktionen.

#### Achtung, verminderte Akkulaufzeit

Diese Test-App läuft unbegrenzt im Hintergrund, was nicht nur die Akku-Laufzeit mindert, sondern auch die Chancen, dass das Programm im App Store akzeptiert wird. Wenn Sie aus dem Testprogramm eine »richtige« App entwickeln möchten, sollten Sie startMonitoringSignificantLocationChanges aufrufen, damit die Position nicht ständig, sondern nur bei nennenswerten Veränderungen aktualisiert wird. Dadurch wird die weiter unten beschriebene Methode locationManager (\_:didUpdateLocations:) viel seltener aufgerufen. Auch eine zeitliche Limitierung der Aufzeichnung kann zweckmäßig sein.

#### Die ViewController-Klasse

Wesentlich interessanter als die optische App-Gestaltung ist bei diesem Beispiel der Code: Die eigene ViewController-Klasse ist wie üblich von UIViewController abgeleitet. Die Klasse implementiert aber außerdem noch die beiden Protokolle CLLocationManagerDelegate und MKMapViewDelegate. Sie sind erforderlich, damit die App Location- bzw. MapView-Ereignisse in Methoden verarbeiten kann. Zwei IBOutlets ermöglichen wie üblich den Zugriff auf die Steuerelemente, zwei Eigenschaften speichern den Location Manager sowie die aufgezeichneten Positionen. Die eigentliche Arbeit erledigen drei Methoden, die im folgenden Listing nur angedeutet sind; deren detaillierte Beschreibung folgt gleich.

```
// Projekt ios-test-gps, Datei ViewController.swift
// Aufbau der ViewController-Klasse
import UIKit
import CoreLocation
import MapKit
class ViewController: UIViewController,
                      CLLocationManagerDelegate,
                      MKMapViewDelegate
{
  @IBOutlet weak var map: MKMapView!
                                           // Zugriff auf die
 @IBOutlet weak var label: UILabel!
                                         // Steuerelemente
 var locmgr:CLLocationManager!
                                          // Location Manager
 var coords:[CLLocationCoordinate2D] = [] // Positions - Array
 override func viewDidLoad() { ... }
                                          // Initialisierung
                                          // neue Position
 func locationManager(...) { ... }
 func mapView(...) -> ... { ... }
                                           // Route zeichnen
}
```

## Initialisierung in viewDidLoad

Die Methode viewDidLoad wird aufgerufen, sobald iOS mit der Low-Level-Initialisierung der App fertig ist. Jetzt ist der Zeitpunkt gekommen, um die eigenen Initialisierungsarbeiten zu erledigen. In unserem Fall geht es dabei um zwei Dinge:

- Der Location Manager soll uns in Zukunft regelmäßig mit Informationen darüber versorgen, wo sich das iPhone gerade befindet. Dazu erzeugen wir wie im vorigen Abschnitt eine Instanz der CLLocationManager-Klasse. Neu sind die weiteren Einstellungen:
  - Mit locmgr.delegate = self geben wir an, dass unsere ViewController-Instanz Location-Ereignisse verarbeiten soll. Das ist nur zulässig, weil die ViewController-Klasse das Protokoll CLLocationManagerDelegate implementiert. Somit können wir unsere Klasse mit im Protokoll definierten Methoden ausstatten, die dann beim Auftreten eines Ereignisses aufgerufen werden. In diesem Beispiel gibt es nur eine derartige Methode, nämlich die im nächsten Abschnitt beschriebene Methode locationManager.

- Mit desiredAccuracy = kCLLocationAccuracyBest geben wir an, dass wir die Position des iOS-Geräts in größtmöglicher Genauigkeit wissen möchten.
- requestAlwaysAuthorization sorgt beim erstmaligen Start der App für den schon bekannten Dialog mit der Frage, ob die App Standortdaten verarbeiten darf. Vergessen Sie nicht, in der Datei Info.plist die Eigenschaft NSLocation-AlwaysUsageDescription hinzuzufügen und ihr einen Erklärungstext zuzuweisen.
- startUpdatingLocation startet schließlich die Ereignisverarbeitung und führt dazu, dass wenig später erstmals die locationManager-Methode aufgerufen wird, wenn neue Positionsdaten zur Verfügung stehen.
- Anders als im ersten Beispiel wollen wir diesmal auch auf die Darstellung des Map-View-Steuerelements Einfluss nehmen. Deswegen hält unser View-Controller auch das MKMapViewDelegate-Protokoll ein. map.delegate = self bewirkt auch hier, dass wir die resultierenden Methodenaufrufe verarbeiten möchten. Das betrifft in diesem Beispiel die Methode mapView, die etwas weiter unten beschrieben wird.

```
override func viewDidLoad() {
  super.viewDidLoad()

  // Location Manager initialisieren
  locmgr = CLLocationManager()
  locmgr.delegate = self
  locmgr.desiredAccuracy = kCLLocationAccuracyBest
  locmgr.requestAlwaysAuthorization()
  locmgr.startUpdatingLocation()

  // Map-Methoden verarbeiten
  map.delegate = self
}
```

#### locationManager-Delegate

Die im Folgenden abgedruckte Methode locationManager (\_:didUpdateLocations:) wird von nun an circa einmal pro Sekunde aufgerufen. Im ersten Parameter wird der für das Ereignis verantwortliche Location Manager übergeben. Interessanter ist der zweite Parameter, der ein Array mit den aktuellen Positionsangaben übergibt, wobei das letzte Element die aktuellsten Daten enthält. Ein erfreulicher Unterschied zwischen Swift 1 und Swift 2 besteht darin, dass die API besser an Swift angepasst wurde: Positionen werden nicht mehr als AnyObject-Array, sondern direkt als CLLocation-Array übergeben.





#### Eingabe von Delegation-Methoden in Xcode

Vielleicht fragen Sie sich, woher Sie wissen, welche Parameter diese Methode erwartet, und wie Sie die Methode am besten in Xcode eingeben. Ganz einfach: Tippen Sie die Anfangsbuchstaben der Methode ein. Xcode zeigt nun eine Liste der zur Auswahl stehenden Methoden, die in unserem Fall zwar alle locationManager heißen, sich aber durch den zweiten Parameter unterscheiden (siehe <u>Abbildung 13.9</u>).

Wir benötigen die Variante der Methode, bei der der zweite Parameter didUpdate-Locations heißt – was auch aus der im vorigen Absatz angegebenen Signatur locationManager (\_:didUpdateLocations:) hervorgeht. Wählen Sie die richtige Methode aus, und drücken Sie  $\leftarrow$ ].

Die folgenden Zeilen extrahieren aus dem aktuellsten CLLocation-Objekt den Längenund Breitengrad, die Seehöhe (leider nicht besonders genau), die Geschwindigkeit und die Bewegungsrichtung in Grad. All diese Informationen werden in mehreren Zeilen im Label angezeigt.

setRegion stellt ein, welchen Kartenausschnitt die MapView zeigen soll. Mit den gewählten Daten wird die gerade aktuelle Position immer mittig im Steuerelement dargestellt. Der sichtbare Ausschnitt soll circa eine Breite von einem Kilometer haben.

Zuletzt geht es nun noch darum, die aktuelle Position im Array coords zu speichern. Da die ersten Positionen oft sehr ungenau sind, ignorieren wir sie. Aber sobald zumindest fünf Positionen vorliegen, erzeugen wir aus den letzten beiden Positionen ein MKPolyline-Objekt. Dieses Objekt fügen wir mit mapOverlay in einen Speicher ein, der sich alle über der Karte darzustellenden Daten merkt. Beachten Sie, dass an dieser Stelle noch nichts gezeichnet wird! Darum kümmert sich das MapKit-Steuerelement selbst, wobei wir diesen Prozess in der mapView-Methode unterstützen.

```
// Projekt ios-test-gps, Datei ViewController.swift
func locationManager(manager:CLLocationManager,
                     didUpdateLocations locations:[CLLocation])
{
 for loc in locations { // alle Positionen übergeben
   // Position anzeigen und dem coords-Array hinzufügen
    let long =
      String(format: "%.4f", loc.coordinate.longitude)
    let lat =
      String(format: "%.4f", loc.coordinate.latitude)
    let alt =
                String(format: "%.1f", loc.altitude)
    let speed = String(format: "%.1f", loc.speed)
    let course = String(format: "%.1f", loc.course)
    label.text = "long= \(long)° / lat = \(lat)° \n" +
                 "alt = (alt) m \langle n'' +
                 "speed = (speed) m/s n'' +
                 "course = \(course)° \n" +
                 "time = \(loc.timestamp) \n"
    // sichtbaren Bereich der Karte (inkl. Zoom) einstellen
    // aktuelle Position immer zentriert
    let span = 0.01 // in Grad; 1° entspricht 111 km,
                     // 0.01° entspricht 1100 m
    let reg = MKCoordinateRegion(
      center: map.userLocation.coordinate,
      span: MKCoordinateSpanMake(span, span))
    map.setRegion(reg, animated: false)
    // aktuelle Position im Array speichern
    coords.append(loc.coordinate)
    // fügt dem Map-Overlay eine Linie vom letzten
    // zum vorletzten Punkt hinzu
    let n = coords.count
    if n > 4 { // die ersten Punkte ignorieren, oft ungenau
     var pts = [coords[n-1], coords[n-2]]
      let polyline =
        MKPolyline(coordinates: &pts, count: pts.count)
     map.addOverlay(polyline)
    }
 } // for - Ende
} // func-Ende
```

#### Location-Benachrichtigungen ohne MapView

Es bietet sich oft an, Ereignisse des Location Managers in Kombination mit einer Map-View zu verarbeiten, aber das ist keineswegs zwingend. Auch ohne MapView kann Ihr Programm einen Location Manager einrichten und dessen Daten verarbeiten. Unbedingt erforderlich ist aber die Abfrage, ob Ihre App Standarddaten empfangen darf (also requestXxxAuthorization in viewDidLoad).

## Die mapView-Methode

Die folgende, im CLLocationManagerDelegate-Protokoll definierte mapView-Methode wird immer dann aufgerufen, wenn das MapView-Steuerelement neu gezeichnet wird. Unsere Aufgabe ist es, darin für unsere Daten ein Overlay-Renderer-Objekt zu erzeugen und zurückzugeben. Dieses Objekt bestimmt, wie die über der Karte darzustellenden Liniensegmente zu zeichnen sind – in unserem Beispiel als roter, 3 Punkte breiter Linienzug.

```
func mapView(mapView: MKMapView,
             rendererForOverlay overlay: MKOverlay)
          -> MKOverlayRenderer
{
  if overlay is MKPolyline {
    // falls Polyline - Overlay: passenden
    // MKPolylineRenderer erzeugen
    let polylineRenderer = MKPolylineRenderer(overlay: overlay)
    polylineRenderer.strokeColor = UIColor.redColor()
    polylineRenderer.lineWidth = 3
    return polylineRenderer
  } else {
    // sonst: leere MKOverlayRenderer-Instanz zurückgeben
    return MKOverlayRenderer()
  }
}
```

## Erweiterungsmöglichkeiten

Wenn Sie Spaß an dem kleinen Programm haben, gibt es eine Menge Erweiterungsmöglichkeiten:

Das Programm bietet im laufenden Betrieb keine Möglichkeit, den sichtbaren Ausschnitt einzustellen. Die aktuelle Position wird immer im Bildschirmmittelpunkt angezeigt, der Zoom-Faktor ist unveränderlich. Schuld daran ist der Aufruf von map.setRegion in der Methode locationManager. Ein anderer Ansatz könnte darin bestehen, setRegion nur einmal nach dem Start der App aufzurufen und dem Benutzer die Kontrolle über die MapView ansonsten zu überlassen. Dazu müssen die Optionen ALLOWS ZOOMING, SCROLLING und ROTATING im Attributinspektor aktiviert werden.

- Ebenso fehlt der App die Möglichkeit, zwischen den verschiedenen Darstellungsvarianten (also Karte, Satellit, Hybrid) umzustellen. Ein Button und eine Code-Zeile wie map.mapType = MKMapType.Hybrid könnten da rasch Abhilfe schaffen.
- Das Programm speichert die aufgezeichnete Route nicht. Sobald das Programm von iOS aus dem Speicher entfernt wird, hat es alles vergessen. Wenn Sie Routen aufzeichnen möchten, müssen Sie eine Speichermöglichkeit für das Array coords anbieten.
- Als logische Ergänzung würden sich nun ein paar Buttons oder ein eleganterer Steuerungsmechanismus anbieten, um die Positionsaufzeichnung zu starten, zu stoppen bzw. zurückzusetzen.

## 13.3 Kompassfunktionen

Zwar gibt es in Xcode kein eigenes Kompasssteuerelement, ansonsten ist die Nutzung der Kompassfunktionen aber denkbar einfach: Wir benötigen wie in den vorangegangenen Beispielen einen Location Manager und müssen im View-Controller das Protokoll CLLocationManagerDelegate implementieren. In viewDidLoad erzeugen wir wie gehabt den Location Manager. Anstelle von startUpdatingLocation führen wir diesmal aber startUpdatingHeading aus: Wir sind nicht an Positionsinformationen, sondern nur an Richtungsangaben interessiert. Dafür müssen wir nicht einmal um Erlaubnis bitten.

Die für uns relevante Delegate-Methode locationManager ist am Namen des zweiten Parameters zu erkennen. Dieser muss didUpdateHeading lauten, d. h. die Signatur der Methode lautet locationManager (\_:didUpdateHeading:). Der Parameter stellt uns ein CLHeading-Objekt zur Verfügung. Von dessen vielen Eigenschaften interessiert uns nur eine: trueHeading gibt an, in welche Richtung das obere Ende der Benutzeroberfläche des iPhones oder iPads zeigt. Die Angabe erfolgt in Grad. O° bedeutet, dass das Gerät aus der Sicht des Benutzers nach Norden zeigt, 90° gelten für Osten etc.

#### iPad-Kompass

Die Kompassfunktionen sind auch bei iPads ohne Mobilfunk- und GPS-Funktionen verwendbar. Beachten Sie aber, dass der Kompass durch Gehäuse mit Magnetverschluss massiv aus dem Gleichgewicht kommt! Eine minimale Auswertung der Kompassdaten mit Debugging-Anzeige in Xcode erfordert somit nur wenige Zeilen Code:

```
import UIKit
import CoreLocation
class ViewController: UIViewController,
                      CLLocationManagerDelegate
{
  var locmgr:CLLocationManager!
  // Location Manager initialisieren
  override func viewDidLoad() {
    super.viewDidLoad()
    locmgr = CLLocationManager()
    locmgr.delegate = self
    locmgr.desiredAccuracy = kCLLocationAccuracyBest
    locmgr.startUpdatingHeading()
  }
  // wird aufgerufen, wenn das iPhone/iPad in eine
  // andere Richtung zeigt
  func locationManager(manager: CLLocationManager,
                       didUpdateHeading newHeading: CLHeading) {
    print(newHeading.trueHeading)
  }
}
```

#### Kompasskalibrierung

Mitunter erkennen iOS-Geräte die Notwendigkeit, die Kompassfunktion neu zu kalibieren. Das ist vor allem dann der Fall, wenn die Funktion zum ersten Mal nach langer Zeit verwendet wird oder wenn das Gerät ein störendes Magnetfeld in der Nähe feststellt.

Es ist Ihrer App überlassen, ob bzw. wie sie auf diese Kalibrierungsaufforderung reagiert. Wenn Sie keinen entsprechenden Code vorsehen, dann verzichtet iOS auf die Kalibrierung; es kann dann aber sein, dass die Richtungsangaben ungenau sind. Besser ist es daher, in den View-Controller den folgenden Code einzubauen:

```
func locationManagerShouldDisplayHeadingCalibration(
   manager: CLLocationManager) -> Bool
{
   return true
}
```

Das führt dazu, dass Ihre App, wann immer sie es für notwendig hält, einen Kompasskalibrierdialog einblendet. Ihre App-Benutzer müssen nun die Kalibrierung durchführen, ob sie wollen oder nicht. In einer »echten« App ist es vermutlich zweckmäßiger, die Benutzer vorher zu informieren und ihnen die Möglichkeit zu geben, diesen Prozess abzubrechen.

## **Grafische Darstellung eines Kompasses**

Es ist zwar kein Problem, den trueHeading-Wert in einem Label anzuzeigen, besonders hilfreich ist das aber selten. Wer einen Kompass braucht, ist in der Regel an einem grafischen Zeiger nach Norden interessiert. Also müssen wir versuchen, den vom Location Manager gelieferten Winkel grafisch darzustellen. Das führt uns zu einem neuen Aspekt der iOS-Programmierung: zur Nutzung grafischer Funktionen und zur Gestaltung eigener Steuerelemente.

# 13.4 Eigene Steuerelemente mit Grafikfunktionen

Hinter den Kulissen sind alle iOS-Steuerelemente Klassen. Was tun Sie, wenn Sie eine vorhandene Klasse um neue Funktionen erweitern möchten? Sie nutzen den Mechanismus der Vererbung (siehe <u>Abschnitt 8.1</u>). Wie dies konkret funktioniert, ist Thema dieses Abschnitts. Nebenbei lernen Sie auch gleich einige Grundkonzepte der Grafikprogrammierung kennen, wobei ich mich hier aber auf recht simple 2D-Funktionen beschränke.

## Eine Klasse für ein neues Steuerelement

Um ein neues Steuerelement zu erzeugen, brauchen Sie als Erstes eine neue Klasse. Dazu führen Sie im aktuellen iOS-Projekt FILE • NEW • FILE aus und wählen die Vorlage COCOA TOUCH CLASS aus (zu finden unter der Rubrik IOS • SOURCE). Im zweiten Schritt benennen Sie die Klasse, z. B. mit CompassView, und geben im Listenfeld SUB-CLASS OF an, von welcher vorhandenen Klasse Sie Ihre Kreation ableiten möchten (siehe <u>Abbildung 13.10</u>).

Für unsere Zwecke eignet sich UIView am besten. Dabei handelt es sich um einen rechteckigen Bereich, dessen Inhalt Sie selbst grafisch gestalten können. Dazu müssen Sie lediglich die drawRect-Methode überschreiben und mit eigenem Code ausstatten.

Zuletzt müssen Sie noch angeben, wo Sie die neue \*.swift-Datei speichern möchten. Sofern Ihr iOS-Projekt nicht schon aus sehr vielen Dateien besteht, spricht nichts dagegen, dies einfach im Grundverzeichnis Ihres Projekts zu tun. In diesem Fall bestätigen Sie den Vorschlag von Xcode einfach mit CREATE. Xcode zeigt die neue Datei an, die anfänglich folgenden Inhalt hat:

Choose options for your n	ew file:	
Class:	CompassView	
Subclass of:	UIView	V
	Also create XIB file	
	iPhone	\$
Language:	Swift	٥
Cancel		Previous Next

Abbildung 13.10 Eine neue Klasse für das Steuerelement anlegen

```
// Projekt ios-test-compass
// Datei CompassView.swift
class CompassView: UIView {
    // Only override drawRect: if you perform custom drawing.
    // An empty implementation adversely affects performance during
    // animation.
    override func drawRect(rect: CGRect) {
        // Drawing code
    }
}
```

#### Grafikprogrammierung

In der neuen Klasse wird die Methode drawRect immer dann aufgerufen, wenn das ganze Steuerelement oder auch nur Teile davon neu zu zeichnen sind. Der neu zu zeichnende Bereich geht aus dem Parameter rect hervor. Im Regelfall werden Sie diesen Parameter ignorieren und einfach alles neu zeichnen.

Beginnen wir mit einem Beispiel: Um innerhalb des neuen Steuerelements eine rote, drei Punkte breite Linie zu zeichnen, ist der folgende Code erforderlich, den ich gleich erläutern werde:

```
// im Steuerelement eine rote, schräge Linie zeichnen
override func drawRect(rect: CGRect) {
    let context = UIGraphicsGetCurrentContext()
    let red = UIColor.redColor().CGColor
    CGContextSetLineWidth(context, 3.0)
    CGContextSetStrokeColorWithColor(context, red)
    CGContextMoveToPoint(context, 10, 10)
    CGContextAddLineToPoint(context, 200, 500)
    CGContextStrokePath(context)
}
```

Alle Zeichenoperationen müssen auf einen Grafikkontext angewendet werden. Diesen Kontext ermittelt die Methode UIGraphicsGetCurrentContext, die uns die UIView-Basisklasse zur Verfügung stellt. Im Grafikkontext werden die Parameter der nachfolgenden Grafikoperationen gespeichert. Dazu zählen die Zeichenfarbe und die Linienstärke, die mit CGContextSetXxx-Methoden eingestellt werden. (CG steht dabei für »Core Graphics«.)

Anschließend legen die Methoden CGContextMoveToPoint und CGContextStrokePath den Start- und Endpunkt der Linie fest. Beachten Sie, dass Sie die Koordinaten in Form von CGFloat-Werten angeben müssen. Dabei handelt es sich je nach Architektur um 32- oder 64-Bit-Fließkommazahlen. Ergebnisse von Berechnungen müssen Sie zumeist explizit mit CGFloat (ausdruck) in diesen Datentyp umwandeln. Die so festgelegte Linie wird zuletzt mit CGContextStrokePath gezeichnet.

#### Zeichnen im UIView versus Zeichnen auf einem MapView

An dieser Stelle greifen wir das Thema Grafik schon zum zweiten Mal auf. Zu Beginn des Kapitels haben wir ja auch im MapView-Steuerelement den zuletzt zurückgelegten Weg markiert. Die dabei eingesetzten Overlay-Methoden sind aber ein MapView-spezifischer Sonderweg, der nur wenige Ähnlichkeiten mit den hier präsentierten und an vielen Stellen in iOS üblichen Zeichenmethoden hat.

#### Das Steuerelement verwenden

Unser CompassView-Steuerelement zeichnet in dieser Form zugegebenermaßen noch keinen Kompass, aber es ist bereits ein syntaktisch korrektes und funktionierendes Steuerelement. Probieren wir es also aus!

Nun stellt sich die Frage, wie das neue Steuerelement in eine App eingefügt werden kann: In der Objektbibliothek erscheint die CompassView nämlich nicht. Tatsächlich ist die Vorgehensweise – zumindest beim ersten Mal – ein wenig merkwürdig: Sie fügen in die App nämlich nicht die CompassView ein, sondern das zugrunde liegende Basissteuerelement – in diesem Beispiel also eine UIView.

Dann klicken Sie das Steuerelement an, öffnen in der rechten Seitenleiste den IDEN-TITY INSPECTOR und stellen im Feld CUSTOM CLASS die Steuerelementklasse ein, die Sie *tatsächlich* nutzen möchten – also CompassView (siehe <u>Abbildung 13.11</u>). Im Auswahlfeld stehen nur passende Steuerelemente zur Auswahl, also solche, die von dem Steuerelement abgeleitet sind, das Sie ursprünglich in den View-Controller eingefügt haben.

Diese Vorgehensweise ist dieselbe, die Ihnen aus dem Umgang mit mehreren View-Controllern schon vertraut ist: Auch dort wird der View-Controller, also genau genommen ein Objekt der UIViewController-Klasse, nachträglich mit einer eigenen Klasse verbunden – z.B. mit MyViewController oder wie immer Sie Ihre Klasse genannt haben.



Abbildung 13.11 »Custom Class« bestimmt die tatsächlich genutzte Steuerelementklasse.

Ein Test im Simulator beweist, dass die Linie innerhalb des CompassView tatsächlich wie geplant gezeichnet wird (siehe <u>Abbildung 13.12</u>). Dass die Linie unten abgeschnitten ist, liegt daran, dass das Steuerelement im View-Controller zu klein dimensioniert wurde.



Abbildung 13.12 Die erste erfolgreiche Grafikausgabe

#### Eine richtige CompassView

Unsere ersten Tests in Ehren, aber mit einem Kompass hat das Steuerelement noch nicht viel zu tun. Um das zu ändern, ergänzen wir in die CompassView-Klasse zuerst um die Eigenschaft heading. Diese Double-Zahl entspricht der trueHeading-Eigenschaft der CLHeading-Klasse. Liefert diese den Wert 90 (d. h., das iOS-Gerät zeigt nach Osten), dann wird die Kompassnadel entsprechend nach Westen gestellt (weil Norden, relativ zum Standpunkt des Nutzers, nun links ist).

Damit der Kompass bei jeder Änderung dieser Eigenschaft neu gezeichnet wird, versehen wir die Eigenschaft mit einem Property Observer. Jede Änderung führt nun dazu, dass die Methode setNeedsDisplay ausgeführt wird. Diese von der UIView-Klasse vererbte Methode löst einen Aufruf von drawRect auf.

```
class CompassView: UIView {
  var heading = 0.0 { // Kompassrichtung in Grad, 0 = Norden
    didSet { // bei Änderung neu zeichnen
    setNeedsDisplay()
  }
  }
  override func drawRect(rect: CGRect) { ... }
}
```

Nun müssen wir noch drawRect um einige Anweisungen ergänzen, die mittig im Steuerelement einen Kreis und darin eine symbolisierte Kompassnadel zeichnen. Der erforderliche Code ist ziemlich lang, aber nicht allzuschwer zu verstehen, wenn Sie die Sinus- und Cosinus-Funktionen kennen. In side wird die Seitenlänge des Quadrats ausgerechnet, das innerhalb des CompassView zur Darstellung des Kompasses vorgesehen ist. Die Seitenlänge ergibt sich aus dem kleineren Wert der Steuerelementlänge bzw. -breite. Die diversen x- und y-Variablen geben die Eckpunkte der Kompassnadel an.

Im Code kommen außerdem zwei neue Zeichenmethoden vor: CGContextAddArc zeichnet den Kreis, CGContextDrawPath zeichnet ein gefülltes umrandetes Polygon.

```
// Projekt ios-test-compass
// Datei CompassView.swift
override func drawRect(rect: CGRect) {
  let context = UIGraphicsGetCurrentContext()
  let rad = heading / 180.0 * M_PI // Winkel von 0 bis 2*Pi
  // Kompassgröße
  let side = min(frame.size.width, frame.size.height)
  let side2 = side/2 // halbe Seitenlänge
  let radout = side2 * 0.95 // Radius zur Spitze
  let radin = side2 * 0.20 // Radius für Ost/West-Punkte
```

```
// Kompassspitze Nord
let xnorth = side2 - radout * CGFloat(sin(rad))
let ynorth = side2 - radout * CGFloat(cos(rad))
// Kompassspitze Süd
let xsouth = side2 - radout * CGFloat(sin(rad + M PI))
let ysouth = side2 - radout * CGFloat(cos(rad + M PI))
// Kompasseite Ost/West
let xeast = side2 - radin * CGFloat(sin(rad + M PI 2))
let yeast = side2 - radin * CGFloat(cos(rad + M PI 2))
let xwest = side2 - radin * CGFloat(sin(rad + 3 * M PI 2))
let ywest = side2 - radin * CGFloat(cos(rad + 3 * M PI 2))
// Kompasseite Ost/West für farblich abgesetzte Spitze
let xeast2 = (2 * xnorth + xeast) / 3
let yeast2 = (2 * ynorth + yeast) / 3
let xwest2 = (2 * xnorth + xwest) / 3
let ywest2 = (2 * ynorth + ywest) / 3
// Farben
let black = UIColor.blackColor().CGColor
let red = UIColor.redColor().CGColor
// Kreis
CGContextSetLineWidth(context, 2.0)
CGContextSetStrokeColorWithColor(context, black)
UIColor.blackColor().set()
CGContextAddArc(context,
                side2, side2,
                                     // x, y
                side2 - 2,
                                      // Radius
                0.0, CGFloat(M PI * 2.0), 1)
CGContextStrokePath (context)
// Kompassnadelspitze (rot)
CGContextSetLineWidth(context, 1.0)
CGContextSetFillColorWithColor(context, red)
CGContextSetStrokeColorWithColor(context, red)
CGContextMoveToPoint(context, xnorth, ynorth)
CGContextAddLineToPoint(context, xeast, yeast)
CGContextAddLineToPoint(context, xwest, ywest)
CGContextAddLineToPoint(context, xnorth, ynorth)
// zeichnet Umrandung und Inhalt
CGContextDrawPath(context, CGPathDrawingMode.FillStroke)
```

```
// Kompassnadel schwarz (Norden)
CGContextSetFillColorWithColor(context, black)
CGContextSetStrokeColorWithColor(context, black)
CGContextMoveToPoint(context,
                                xeast2, yeast2)
CGContextAddLineToPoint(context, xeast, yeast)
CGContextAddLineToPoint(context, xwest,
                                        ywest)
CGContextAddLineToPoint(context, xwest2, ywest2)
CGContextAddLineToPoint(context, xeast2, yeast2)
CGContextDrawPath (context, CGPathDrawingMode.FillStroke)
// Kompassnadel weiß (Süden)
CGContextSetStrokeColorWithColor(context, black)
CGContextMoveToPoint(context, xsouth, ysouth)
CGContextAddLineToPoint(context, xeast, yeast)
CGContextAddLineToPoint(context, xwest, ywest)
CGContextAddLineToPoint(context, xsouth, ysouth)
CGContextStrokePath(context) // nur Stroke
```

Naturgemäß können Sie die optische Gestaltung des Steuerelements noch optimieren, z. B. indem Sie die vier Himmelsrichtungen auf der Rose markieren, Gradangaben wie auf einem Zifferblatt einfügen etc. Auch 3D-Effekte wären denkbar – aber Sie wissen ja: Spätestens seit iOS 7 gilt bei der grafischen Gestaltung von Apps die Devise »Weniger ist mehr«.

#### Automatischer Redraw bei Größenänderung

}

Wenn sich die Größe des Steuerelements ändert, z.B. bei einer Drehung eines iOS-Geräts, dann erwarten wir natürlich, dass das Steuerelement in der neuen Größe neu gezeichnet wird. iOS kümmert sich darum – aber tut dies nicht automatisch. Vielmehr müssen Sie dazu das Steuerelement im Storyboard zuerst anklicken und dann im Attributinspektor die View-Eigenschaft MODE auf REDRAW stellen (siehe <u>Abbildung 13.13</u>). Hinter den Xcode-Kulissen heißt die betreffende Eigenschaft der UIView-Klasse contentMode.

	0	I 💎	3	Ð	
Compass View					
Heading				90	\$
Color		Default			٢
North Color		Default			٢
View					
Mode	Redraw			٢	
Tag				0	10

**Abbildung 13.13** Die Einstellung »Mode = Redraw« stellt sicher, dass das Steuerelement bei Größenänderungen korrekt neu gezeichnet wird.

## Kompassnadel einstellen

Nun geht es nur noch darum, die Kompassnadel entsprechend der Daten der locationManager-Methode auszurichten und die Abweichung gegenüber der Nordrichtung in einem Label anzuzeigen:

Im Simulator lässt sich dieses Programm nicht sinnvoll testen, aber die probeweise Ausführung auf einem iPhone zeigt, dass das Programm wie erwünscht funktioniert (siehe <u>Abbildung 13.14</u>).



Abbildung 13.14 Eine Mini-App mit dem neuen »CompassView«-Steuerelement

## Den Kompass an die Ausrichtung des Geräts anpassen

Allerdings fördert der Test ein neues Problem zutage: Wird das iPhone ins Querformat gedreht, zeigt die Kompassnadel plötzlich nicht mehr nach Süden! Schuld daran ist, dass der Location Manager nichts von der Drehung der iOS-Benutzeroberfläche weiß. Abhilfe ist zum Glück nicht schwierig: Sie müssen bei jeder Änderung der Ausrichtung des Geräts auch die Ausrichtung des Kompasses anpassen. Dazu richten Sie in viewDidLoad die Methode rotated ein, die immer dann automatisch ausgeführt werden soll, wenn sich die Geräteausrichtung ändert. add0bserver meldet rotated beim Notification Center an. Dabei handelt es sich um eine zentrale Kommunikationseinheit für iOS-Apps. Hintergrundinformationen zum Notification Center folgen in Abschnitt 15.3, »Location Manager selbst gemacht«.

In rotated werten Sie die orientation-Eigenschaft von currentDevice aus und stellen entsprechend die headingOrientation des Location Managers ein. Die naheliegende Zuweisung

```
locmgr.headingOrientation = UIDevice.currentDevice().orientation
```

ist übrigens nicht möglich, weil die Eigenschaften headingOrientation und orientation unterschiedliche Datentypen aufweisen.

```
// Projekt ios-test-compass
// Datei ViewController.swift
override func viewDidLoad() {
  super.viewDidLoad()
 locmgr = CLLocationManager()
 // ... wie bisher
 // bei einer Änderung der Geräteausrichtung
 // die Methode rotated() ausführen
 NSNotificationCenter.defaultCenter().addObserver(
    self,
    selector: "rotated",
    name: UIDeviceOrientationDidChangeNotification,
    object: nil)
}
// Kompassausrichtung an die Geräteausrichtung anpassen
func rotated() {
  switch UIDevice.currentDevice().orientation {
 case .Portrait:
    locmgr.headingOrientation = .Portrait
 case .LandscapeLeft:
    locmgr.headingOrientation = .LandscapeLeft
  case .LandscapeRight:
    locmgr.headingOrientation = .LandscapeRight
  case .PortraitUpsideDown:
    locmgr.headingOrientation = .PortraitUpsideDown
  default: break
 }
}
```

#### Xcode-Integration mit IBDesignable und IBInspectable

Prinzipiell erfüllen das Kompasssteuerelement und seine Integration in die Kompass-App nun alle Aufgaben. Der Umgang mit dem Steuerelement ist aber wenig elegant: In Xcode wird das Steuerelement nur als rechteckiger Rahmen dargestellt; seine Eigenschaft heading kann nur per Code, aber nicht wie bei anderen Steuerelementen im Attributinspektor eingestellt werden.

Mit ein wenig Mühe können wir aus dem CompassView-Steuerelement ein vollwertiges Steuerelement machen. Dazu müssen wir die CompassView-Klasse mit dem Attribut @IBDesignable versehen und die für Xcode zugänglichen Eigenschaften der Klasse mit dem Attribut @IBInspectable kennzeichnen. Vorweg ein paar Hintergrundinformationen:

- @IBInspectable macht eigene Eigenschaften für den Attributinspektor von Xcode zugänglich. Wenn Sie also mit Swift ein neues Steuerelement entwickeln und eine Eigenschaft mit dem Attribut @IBInspectable auszeichnen, können Sie diese Eigenschaft im Attributinspektor einstellen. Dabei werden unter anderem die folgenden Datentypen unterstützt: Bool CGFloat, CGPoint, CGRect, CGSize, Double, Int, String, UIColor und UIImage.
- @IBDesignable ist für eigene Steuerelemente gedacht, die von der UIView-Klasse abgeleitet sind. Xcode kann mit diesem Attribut ausgestattete Steuerelemente im Storyboard-Editor direkt darstellen.

Damit das Steuerelement in Xcode eine »Live View« bietet, also bereits in der Vorschau korrekt angezeigt wird, stellen Sie der Klasse @IBDesignable voran. Außerdem machen Sie Eigenschaften des Steuerelements mit @IBInspectable für den Attributinspektor zugänglich. Dabei hat sich herausgestellt, dass der Datentyp der Eigenschaft explizit angegeben werden muss – hier also mit Double. Der Compiler erkennt zwar aufgrund der Zuweisung des Defaultwerts 0.0, dass es sich um eine Double-Variable handeln muss, aber für @IBInspectable ist das anscheinend zu wenig.

```
// Projekt ios-test-compass
// Datei CompassView.swift
import UIKit
@IBDesignable class CompassView: UIView {
    // Kompassrichtung in Grad, 0 = Norden
    @IBInspectable var heading:Double = 0.0 {
    didSet { // bei Änderung neu zeichnen
        setNeedsDisplay()
    }
    }
    // weiterer Code wie bisher
}
```


Abbildung 13.15 Vorschau des »CompassView«-Steuerelements in einem Storyboard, Einstellung seiner Eigenschaften im Attributinspektor

Da sich Eigenschaften nun derart bequem im Attributinspektor einstellen lassen, liegt es nahe, auch die Farben des Kompasses als Eigenschaften zu definieren:

```
// Farbe für die Kompassrose und die Nadel
@IBInspectable var color:UIColor = UIColor.blackColor()
```

```
// Farbe für die nach Norden zeigende Kompassspitze
@IBInspectable var northColor:UIColor = UIColor.redColor()
```

In der drawRect-Methode müssen die Farbeinstellungen entsprechend angepasst werden, z. B. so:

```
CGContextSetStrokeColorWithColor(context, color.CGColor)
```

Der Lohn unserer Arbeit zeigt sich in Xcode: Das Kompasssteuerelement wird nun bereits in der Vorschau korrekt angezeigt, seine Eigenschaften können direkt eingestellt werden (siehe <u>Abbildung 13.15</u>). Beachten Sie übrigens, dass Xcode die Eigenschaftsnamen nicht einfach unverändert übernimmt. Vielmehr wird der erste Buchstabe zu einem Großbuchstaben, außerdem wird die Bezeichnung bei Klein/Groß-Wechsel abgetrennt. Aus northColor wird also NORTH COLOR. Schade, dass Xcode das Steuerelement nicht auch in der Objektbibliothek anzeigt!

# Kapitel 19 OS-X-Grundlagen

Dieses Kapitel gibt einen systematischen Einstieg in einige grundlegende Themen der OS-X-Programmierung. In den folgenden Abschnitten lernen Sie:

- ► wie Sie Storyboard-Projekte mit mehreren Fenstern, View-Controllern und Segues organisieren
- ► wie Sie mit dem Tab-View-Controller einen Einstellungsdialog zusammensetzen und Optionen in den User-Defaults speichern
- ▶ wie Sie Standarddialoge zur Auswahl von Dateien, Schriften und Farben aufrufen
- ▶ wie Sie Maus- und Tastaturereignisse verarbeiten
- ► wie Sie das Hauptmenü und Kontextmenüs gestalten und auf die Menüauswahl reagieren
- wie Sie Programme ohne Menüs gestalten (sogenannte »Menubar-Apps«)
- ► wie Sie Eigenschaften von Steuerelementen und Eigenschaften Ihrer Klassen durch sogenanntes »Binding« verknüpfen

Schon an dieser Stelle sei betont, dass dieses Kapitel keinen Anspruch auf Vollständigkeit erhebt. Es gibt genug Themen rund um die OS-X-Programmierung, um gleich ein ganzes Dutzend derartiger Kapitel zu füllen. Das Ziel dieses Kapitel besteht also darin, ein Fundament zu schaffen und wichtige Arbeitstechniken vorzustellen. Mit diesem Wissen und etwas Internet-Recherche sollte es Ihnen möglich sein, sich selbst in weitere Themen einzuarbeiten.

# 19.1 Programme mit mehreren Fenstern

Der Lottozahlengenerator aus dem vorigen Kapitel bestand aus nur einem Fenster. In diesem Abschnitt gebe ich Ihnen anhand eines Beispiels Tipps zur Verwaltung von mehreren Fenstern. Dabei setze ich wie bei allen weiteren Programmen voraus, dass Sie mit Storyboards arbeiten.

Das Beispielprogramm zeigt anfänglich ein Startfenster an. Mit dessen Buttons erzeugen dann beliebig viele Exemplare eines weiteren Fensters. Dabei können Sie eine Nachricht übergeben. Die Buttons der neuen Fenster demonstrieren verschiedene Möglichkeiten, eine an das Fenster gekoppelte Ansicht als *Sheet*, als Popup-Fenster oder als modalen Dialog zu öffnen (siehe Abbildung 19.1).

	Noch ein Fenster		
Ansicht	OK ter		
Modaler Dia	alog Schließen		
Startfenster	Noch	ein Fenster	
Startfenster  Nachricht an Fenster 2	Noch	ein Fenster	
Startfenster  Nachricht an Fenster 2  Noch ein Fenster (Storyboard-Segue)	Noch Nein	ein Fenster	
Startfenster  Nachricht an Fenster 2  Noch ein Fenster (Storyboard-Segue)  Noch ein Fenster (Segue-Code)	Noch Nein Ansicht zeigen (Sheet)	ein Fenster Popup-Fenster	
Startfenster  Nachricht an Fenster 2  Noch ein Fenster (Storyboard-Segue)  Noch ein Fenster (Segue-Code)  Noch ein Fenster (Code)	Nein Ansicht zeigen (Sheet) Modaler Dialog	ein Fenster Popup-Fenster Schließen	

Abbildung 19.1 Beispielprogramm zur Fensterverwaltung

Das Storyboard für das Beispiel besteht aus zwei Window- und vier View-Controllern (siehe <u>Abbildung 19.2</u>). Neue Window- oder View-Controller fügen Sie aus der Objektbibliothek in den Storyboard-Editor ein. Die View-Controller sind anfänglich mit NSViewController-Klassen verbunden.



Abbildung 19.2 Storyboard des Beispielprogramms zur Fensterverwaltung

Um die View-Controller mit eigenem Code zu verbinden, fügen Sie dem Projekt eine neue Datei vom Typ COCOA CLASS hinzu, wählen NSViewController als Basisklasse und deaktivieren die Option ALSO CREATE XIB FILE. Anschließend wählen Sie den View-Controller im Storyboard aus und stellen im Identity Inspector die gerade erzeugte Klasse als CUSTOM CLASS ein.

### Segues

Segues dienen in iOS-Programmen dazu, Übergänge zwischen Ansichten zu gestalten. Unter OS X können Segues aber auch einfach Verbindungen zwischen Controllern herstellen – dann spricht man von »Containment Segues« oder »Presenting Segues«.

Gewöhnliche Segues erzeugen Sie, indem Sie mit <u>[ctrl]</u>-Drag eine Verbindung von einem Steuerelement zu einem View-Controller herstellen. Dabei haben Sie die Wahl zwischen verschiedenen Typen:

- ► SHEET: den View-Controller als an die Fensterleiste fixierten Subdialog anzeigen
- ▶ POPOVER: den View-Controller in einem Popup-Fenster anzeigen
- MODAL: den View-Controller als modalen Dialog anzeigen; damit wird das Hauptprogramm bis zum Schließen dieses Dialogs blockiert.
- ► SHOW: den View-Controller als neues, eigenes Fenster anzeigen
- ► CUSTOM: ermöglicht eine individuelle Gestaltung.

Containment Segues erstellen Sie, falls notwendig, indem Sie eine Verbindung vom Window-Controller zum View-Controller herstellen. Sie starten den <u>ctrl</u>-Drag-Vorgang im blauen Icon WINDOW CONTROLLER in der Titelleiste. Als einziger Typ steht dann WINDOW CONTENT zur Auswahl.

#### Segues von einem Steuerelement zu einem Windows-Controller

Xcode erlaubt es Ihnen, auch Segues zwischen einem Steuerelement und einem Windows-Controller herzustellen (also nicht wie üblich zu einem View-Controller). Sie haben dann die Wahl zwischen den drei Segue-Туреп зноw, сиsтом und модаL.

Grundsätzlich funktionieren diese Segues einwandfrei, allerdings wird dabei die Methode prepareToSegue nicht aufgerufen. Der praktische Nutzen solcher Segues ist damit gering, weil Sie keine Daten vom Quell-Controller zum Ziel-Controller übergeben können. Diese Funktion wirkt momentan (Xcode 6.3) noch unausgereift.

Im Beispielprogramm gibt es die folgenden Segues:

- vom Start-Window-Controller zum View-Controller (Containment Segue, wurde von Xcode bei der Projekterstellung eingerichtet)
- vom zweiten Window-Controller zum View-Controller Window2VC (Containment Segue, wurde von Xcode erstellt, als ein zweiter Window-Controller in das Storyboard eingefügt wurde)

- ► vom Button Noch ein Fenster (Storyboard-Segue) zum View-Controller Window2VC
- ► vom Button ANSICHT ZEIGEN (SHEET) zum View-Controller des Popup-Fensters (Typ sheet)
- ► vom Button MODALER DIALOG zum View-Controller ModalVC (Typ MODAL)
- vom Button POPUP-FENSTER zum View-Controller mit dem Text ANSICHT 2 (Typ POPOVER)

### Datenübergabe mit der Methode prepareForSegue

Grundsätzlich erfolgt die Anzeige der neuen Ansicht automatisch, also ohne Code. Oft wollen Sie aber vom Quell-Controller Daten an den Ziel-Controller übergeben. Dabei hilft Ihnen die aus der iOS-Programmierung schon vertraute Methode prepareForSegue. Sie wird vor dem Segue aufgerufen und ermöglicht es, auf den Ziel-Controller zuzugreifen. Dabei dürfen Sie aber noch nicht auf dessen Steuerelemente (Outlets) zugreifen, weil diese noch nicht initialisiert sind. Stattdessen übergeben Sie die Daten an Eigenschaften der Klasse des Ziel-Controllers.

In der viewDidLoad-Methode des Ziel-Controllers können Sie die Eigenschaften dann auslesen und gegebenenfalls in Steuerelemente übertragen. Im Beispielprogramm wird dieser Mechanismus verwendet, um einen Text aus dem Startfenster in ein neues Fenster zu übergeben. Der erforderliche Code im Quell-Controller sieht so aus:

```
// Projekt osx-multi
// Datei ViewController.swift
class ViewController: NSViewController {
   @IBOutlet weak var txtfield: NSTextField!

   // Datenübergabe an den Ziel-Controller
   override func prepareForSegue(segue: NSStoryboardSegue,
      sender: AnyObject?)
   {
      if let dest = segue.destinationController as? Window2VC {
         dest.data = txtfield.stringValue
      }
   }
}
```

Im Ziel-Controller wertet viewDidLoad die Eigenschaft data aus:

```
// Projekt osx-multi
// Datei Window2VC.swift
class Window2VC: NSViewController {
 @IBOutlet weak var label: NSTextField!
 var data:String?
```

```
// Initialisierung eines Labels
override func viewDidLoad() {
   super.viewDidLoad()
   label.stringValue = data ?? "keine Nachricht"
}
```

Oft wollen Sie beim Schließen des Ziel-Controllers Daten zurück in den Quell-Controller übergeben. Das gelingt am einfachsten, wenn Sie im Ziel-Controller eine weak-Variable einrichten, die zurück auf den Quell-Controller zeigt. Die Variable muss weak sein, damit kein zyklischer Verweis zwischen Quell- und Ziel-Controller entsteht. Ein zyklischer Verweis würde verhindern, dass die Speicherverwaltung den Ziel-Controller wieder aus dem Speicher entfernt, wenn dieser nicht mehr benötigt wird.

Eleganter, aber aufwendiger zu programmieren als der Rückverweis wäre ein eigenes Delegation-Protokoll wie ich es in <u>Abschnitt 15.7</u>, »Detailansicht mit Richtungspfeil«, demonstriert habe.

Die Variable initialisieren Sie in prepareForSegue. Der Ziel-Controller kann damit nun auf Daten und Methoden des Quell-Controllers zugreifen. Im Beispielprogramm wird dieser Mechanismus anhand des modalen Ja/Nein-Dialogs demonstriert. Der Aufruf dieses Dialogs erfolgt durch einen Button des Window2VC-Controllers und führt zu einem Aufruf von prepareForSegue:

```
// Projekt osx-multi, Datei Window2VC.swift
class Window2VC: NSViewController {
   override func prepareForSegue(
     segue: NSStoryboardSegue, sender: AnyObject?)
   {
     if let dest = segue.destinationController as? ModalVC {
        dest.srcVC = self
     }
   }
}
```

Der Ziel-Controller, also in der Klasse ModalVC, kann nun über die Eigenschaft srcVC auf den Quell-Controller zugreifen:

```
// Projekt osx-multi, Datei Window2VC.swift
class ModalVC: NSViewController {
  weak var srcVC: Window2VC!
  @IBAction func btnNo(sender: NSButton) {
    srcVC.label.stringValue = "Nein"
    dismissController(sender)
}
```

```
@IBAction func btnYes(sender: NSButton) {
    srcVC.label.stringValue = "Ja"
    dismissController(sender)
  }
}
```

### Fenstergröße fixieren

Neue Fenster/Dialoge/Popups übernehmen grundsätzlich die Größe des View-Controllers. Die Größe von Fenstern, die durch einen SHOW- oder MODAL-Segue erzeugt werden, ist aber in der Regel veränderlich. Wenn Sie das nicht möchten, bestehen zwei Möglichkeiten:

- ► Sie können die Größe durch Layoutregeln fixieren.
- Sie können nach dem Erzeugen des Fensters dessen Eigenschaften als unveränderlich festlegen.

Dieser Abschnitt behandelt die erste Variante, Tipps zum zweiten Lösungsweg folgen im nächsten Abschnitt. Das Problem mit den Layoutregeln besteht darin, dass Sie die Größe der ersten View in dem View-Controller nicht durch Regeln fixieren können. Sie müssen stattdessen den umgekehrten Weg beschreiten und für zumindest ein Steuerelement im View eine fixe Größe einstellen. Anschließend legen Sie auch den Abstand dieses Steuerelements zu den Rändern des Containers fix fest.

Diese Vorgehensweise habe ich beim View-Controller Window2VC gewählt (siehe <u>Abbildung 19.3</u>): Das Textfeld hat eine Größe von 322 × 17 Punkten, die Abständen zum Container (SUPERVIEW) sind in allen vier Richtungen fixiert.



Abbildung 19.3 Layoutregeln für das Textfeld fixieren die Größe der Ansicht.

### Window-Eigenschaften des Ziel-Controllers einstellen

Bei Übergängen vom Typ MODAL oder VIEW erscheint der View-Controller in einem neuen Fenster, das unabhängig vom Ausgangsfenster ist. Selbst wenn der Ziel-Controller mit einem Window-Controller verbunden ist, werden die dort eingestellten Fenstereigenschaften ignoriert. Wenn Sie individuelle Einstellungen für das Fenster wünschen, müssen Sie diese in der Methode viewDidAppear des Ziel-Controllers einstellen. Beachten Sie, dass eine Einstellung in viewDidLoad nicht möglich ist! Zu diesem Zeitpunkt existiert noch kein Fenster, die Eigenschaft window enthält daher nil.

Im Beispielprogramm stellt viewDidAppear der Window2VC-Klasse den Fenstertitel ein.

```
// Projekt osx-multi, Datei Window2VC.swift
class Window2VC: NSViewController {
   override func viewDidAppear() {
     super.viewDidAppear()
     let win = view.window!
     win.title = "Noch ein Fenster"
   }
}
```

Auch beim modalen Dialog (Klasse ModalVC) wird der Fenstertitel auf diese Weise eingestellt. Außerdem soll die Fenstergröße unveränderlich sein. Das Fenster darf nicht minimiert oder geschlossen werden. Um das zu erreichen, wird die Eigenschaft styleMask ohne die sonst üblichen Attribute NSResizableWindowMask, NSClosable-WindowMask und NSMiniaturizableWindowMask eingestellt. Das Fenster soll also lediglich den Titel anzeigen.

```
// Projekt osx-multi, Datei ModalVC.swift
override func viewDidAppear() {
   super.viewDidAppear()
   view.window?.title = "Ein modaler Dialog"
   view.window?.styleMask = NSTitledWindowMask
}
```

### Ansichten/Fenster schließen

Ansichten, die mit einem Segue des Typs POPOVER, MODAL oder SHEET erzeugt wurden, können mit der Methode dismissController geschlossen werden. Selbst diese eine Zeile Code können Sie sich oft sparen, in dem Sie mit <u>ctrl</u>-Drag eine Verbindung vom Button zum blauen View-Controller-Icon zeichnen. Im nun erscheinenden Menü wählen Sie unter RECEIVED ACTIONS den Eintrag DISMISSCONTROLLER aus. Beachten Sie aber, dass diese Auswahl eine eventuell andere für den Button eingerichtete Action ersetzt. Wenn es also eine Action-Methode gibt, dann müssen Sie die Methode dismissController dort ausführen.

Ein Sonderfall sind Ansichten, die mit einem SHOW-Segue in einem eigenen, unabhängigen Fenster angezeigt werden. Hier bleibt dismissController wirkungslos. Sie müssen explizit das Fenster mit close schließen.

```
// Projekt osx-multi, Datei Window2VC.swift
class Window2VC: NSViewController {
    // Fenster schließen
    @IBAction func btnClose(sender: NSButton) {
      view.window?.close()
    }
}
```

### Segues per Code ausführen

Um einen Segue per Code auszuführen, müssen Sie ihm zuerst im Attributinspektor einen Namen (IDENTIFIER) geben. Anschließend können Sie ihn mit performSegue-WithIdentifier ausführen:

```
@IBAction func btnOpen(sender: NSButton) {
    performSegueWithIdentifier("SegueToAnother", sender: self)
}
```

### Fenster per Code erzeugen

Wenn Sie ein neues Fenster ohne Segue per Code erzeugen möchten, müssen Sie im Storyboard sowohl einen Window- als auch einen View-Controller einrichten. Dem Window-Controller müssen Sie im Identity Inspector im Feld Storyboard ID einen Namen geben.

In jedem View-Controller können Sie über die Eigenschaft storyboard auf das Storyboard des Programms zugreifen. Die Methode instantiateControllerWithIdentifier erzeugt eine Instanz des namentlich genannten Window-Controllers. showWindow zeigt das Fenster an.

Vergessen Sie nicht, eine Referenz auf den neuen Controller in einer Variablen oder in einem Array zu speichern! Gibt es keine Referenz, löscht die Speicherverwaltung das neu erzeugte Objekt, und das Fenster verschwindet sofort wieder vom Bildschirm.

Im folgenden Beispielcode sollen zusätzlich Daten an den View-Controller im neuen Fenster übergeben werden. Das ist prinzipiell kein Problem, die Vorgehensweise ist aber anders als in prepareForSegue. Der View-Controller ist nämlich bereits initialisiert, und der Code in viewDidLoad wurde schon ausgeführt. Dafür ist es nun möglich, Eigenschaften des Ziel-View-Controllers einzustellen, und ein Property Observer überträgt die Daten dann auf die bereits zur Verfügung stehenden Steuerelemente.

```
// Projekt osx-multi, Datei ViewController.swift
class ViewController: NSViewController {
  var win2array:[NSWindowController] = []
  // anderes Fenster ohne Segue erzeugen
  @IBAction func btnOpen2(sender: NSButton) {
    if let winctrl = storyboard!
      .instantiateControllerWithIdentifier("w2vc")
      as? NSWindowController
    {
      if let w2vc = winctrl.contentViewController as? Window2VC {
        // Daten übertragen
        w2vc.data = txtfield.stringValue
      }
      winctrl.showWindow(self)
      win2array.append(winctrl) // sonst sofort wieder weg!
    }
  }
}
// Projekt osx-multi, Datei Window2VC.swift
class Window2VC: NSViewController {
  @IBOutlet weak var label: NSTextField!
  // Property Observer für 'data', wenn das
  // Textfeld zur Verfügung steht, werden
  // Änderungen sofort dort angezeigt
  var data:String? {
    didSet {
      label?.stringValue = data!
    }
  }
  // ... weiterer Code
}
```

# 19.2 Tab-View-Controller

Der in OS X 10.10 eingeführte Tab-View-Controller ergänzt das schon lange verfügbare Tab-View-Steuerelement um eine Variante, die besonders gut für das Storyboard geeignet ist: Mit dem Tab-View-Controller können Sie mehrblättrige Dialoge gestalten, bei denen für jede Dialogseite ein eigener View-Controller zuständig ist. Das Beispielprogramm in diesem Abschnitt zeigt, wie Sie mit dem Tab-View-Controller einen Einstellungsdialog gestalten (siehe <u>Abbildung 19.4</u>). Der Einstellungsdialog kann wahlweise über den Menüeintrag EINSTELLUNGEN oder einen gleichnamigen Button im Hauptfenster geöffnet werden.

Tab-Bar-Controller-Beispiel	Allgemeines
Einstellungen sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit	Schriftgröße 15 © Radio © Radio © Radio © Radio © Radio

Abbildung 19.4 Veränderung der Textgröße in einem Einstellungsdialog

Dieser Abschnitt ist nicht nur eine logische Fortsetzung zum View-Controller-Beispiel aus dem vorigen Abschnitt, er greift auch nochmals das Thema der User-Defaults auf (siehe auch <u>Abschnitt 11.7</u>, »Daten persistent speichern«):

- Zum einen sollen die im Einstellungsdialog veränderten Optionen natürlich bleibend gespeichert werden.
- Zum anderen zeigt das Beispiel einen praktikablen Weg auf, wie Sie Defaulteinstellungen für die User-Defaults-Datenbank definieren.

# Defaults für User-Defaults?

Die vielen »Defaults« können einen hier schwindlig werden lassen. Eine kurze Erklärung: Die »User-Defaults« sind Benutzereinstellungen, die in einer \*.plist-Datei gespeichert werden. Sobald ein Programm einmal einen Wert dort gespeichert hat, kann es später wieder auf ihn zurückgreifen.

Was geschieht aber, wenn das Programm zum ersten Mal läuft und die User Defaults noch leer sind? Dann müssen Standardwerte (Defaults) zum Einsatz kommen. Hier geht es also um diese Default-Werte. Die hier vorgestellten Techniken gelten auch für iOS.

Die einzige Funktion des Beispielprogramms ist die bleibende Einstellung der Textgröße. Alle anderen Radio- und Checkbox-Buttons im Einstellungsdialog dienen nur als Platzhalter.

# Storyboard und Tab-View-Controller-Einstellungen

Das Beispielprogramm besteht im Wesentlichen aus einem gewöhnlichen Fenster und einem Einstellungsdialog. In Kombination mit den dazugehörenden View-Controllern und Segues ergibt das ein eindrucksvolles Storyboard (siehe <u>Abbildung 19.5</u>).



Abbildung 19.5 Storyboard des Tab-View-Controller-Beispiels

Wenn Sie einen Tab-View-Controller aus der Objektbibliothek in das Storyboard einfügen, bildet Xcode eine Kombination aus dem Tab-View-Controller und zwei View-Controllern, die für zwei Seiten des mehrblättrigen Dialogs zuständig sind. Benötigen Sie mehr Seiten, fügen Sie einfach weitere View-Controller ein und stellen mit <u>ctrl</u>-Drag eine Verbindung vom Tab-View-Controller zum neuen View-Controller her.

Für den Tab-View-Controller selbst führen Sie im Attributinspektor normalerweise nur eine einzige Einstellung durch: Die STYLE-Eigenschaft bestimmt das Erscheinungsbild des Dialogs, wobei TABS ON TOP, TABS ON BOTTOM oder TOOLBAR zur Auswahl stehen. Letzere Option kommt bei diesem Beispiel zur Anwendung und gibt dem Dialog das aus anderen Programmen vertraute Erscheinungsbild von Einstellungsdialogen.

### Tab-Reihenfolge ändern

Mit der Einstellung STYLE = TOOLBAR geht dem Tab-View-Controller leider die Vorschaufunktion verloren. Wenn Sie die Reihenfolge der Tabs ändern möchten, schalten Sie vorübergehend auf TABS ON TOP um und verschieben die Dialogblätter in der Tab-Leiste nach Bedarf.

Der Tab-View-Controller übernimmt die Beschriftung der Reiter automatisch aus den Titeln der enthaltenen View-Controller. Damit jedes Dialogblatt die für Einstellungsdialoge typischen Icons zeigt, fügen Sie zuerst passende Bilder in Images.xcassets ein. Anschließend wählen Sie in der Document Outline oder direkt im Storyboard die Reiter aus (Tool-Bar-Items) und stellen im Attributinspektor die IMAGE-Eigenschaft ein.

# Dialogblattgröße

Die Größe des mehrblättrigen Dialogs ergibt sich aus der Größe des gerade aktiven View-Controllers. Wenn Sie vermeiden möchten, dass sich die Fenstergröße mit jedem Seitenwechsel ändert, achten Sie darauf, alle Dialogblätter gleich groß zu gestalten.

Ob die Größe des Dialogs verändert werden kann, hängt davon ab, wie die Größe des View-Controllers bestimmt ist. Gibt es im View-Controller (zumindest) ein Steuerelement, das eine fixe Größe und fixe Abstände zu den vier Rändern des Controllers aufweist, übernimmt der Dialog die Größe unveränderlich. Gibt es hingegen keine derartigen Layoutregeln, kann die Größe des Fensters mit dem mehrblättrigen Dialog im laufenden Betrieb verändert werden.

Im Beispielprogramm können Sie beide Effekte ausprobieren. Im View-Controller für das Dialogblatt ALLGEMEINES gibt es keine Layoutregeln. Solange dieses Dialogblatt angezeigt wird, können Sie die Fenstergröße ändern. Wechseln Sie aber in das Dialogblatt DATENSCHUTZ, vergrößert sich der Einstellungsdialog so weit, dass der gesamte View-Controller Platz findet. Eine Größenänderung ist nun nicht mehr möglich. Die Größe des View-Controllers ergibt sich im Beispielprogramm aus der Checkbox CHECK MIT GANZ LANGEM TEXT. Für dieses Steuerelement wurden sechs Layoutregeln definiert, um Länge, Breite sowie Randabstände zu fixieren.

### Segues

Im Beispielprogramm gibt es die folgenden Verbindungen und Segues:

- ► vom Window-Controller zum View-Controller mit dem Lorem-Ipsum-Text (Typ: WINDOW CONTENT)
- ► vom Tab-View-Controller zu den View-Controllern mit den Dialogseiten (Typ: TAB ITEM)

- ▶ vom Button Einstellungen zum Tab-View-Controller (Typ: MODAL)
- ▶ vom Menüeintrag PREFERENCES zum Tab-View-Controller (Typ: MODAL)

Mehr Details zum Umgang mit Menüeinträgen folgen in <u>Abschnitt 19.6</u>, »Menüs«, aber auf ein Detail möchte ich an dieser Stelle schon hinweisen: Da das Menü ein eigenes, vom View-Controller unabhängiges Objekt ist, kommt es im View-Controller des Fensters zu keinem prepareForSegue-Aufruf!

# Splitter-Steuerelement

Der View-Controller ALLGEMEINES enthält ein Textfeld und einen sogenannten »Splitter«. Dieses durch die NSSplitter-Klasse abgebildete Steuerelement enthält zwei winzige Buttons mit Pfeilspitzen, die nach oben und unten zeigen. Für den Splitter ist im Attributinspektor ein zulässiger Wertebereich von 6 bis 127 eingestellt. Größere oder kleinere Werte können im Textfeld direkt eingegeben werden, aber nicht durch einen Klick auf den Splitter.

### Splitter und Textfeld durch Bindings verbinden

Häufig bietet es sich an, Splitter und Textfeld durch Bindungs zu verbinden (siehe <u>Abschnitt 19.8</u>, "Bindings"). In diesem Beispiel müssen aber *vier* Einstellungen synchronisiert werden: die Textgröße im Hauptfenster, der in den User-Defaults gespeicherte Wert sowie die Inhalte des Textfelds und des Splitters aus dem Einstellungsdialog. Das lässt sich am einfachsten durch einige Zeilen Code bewerkstelligen.

# Klassen

Der Code zu dem Beispiel verteilt sich über drei Klassen:

- AppDelegate enthält eine eigene Init-Funktion, die die Datei appdefaults.plist mit den User-Defaults verbindet.
- ► ViewController mit dem View-Controller zum Hauptfenster lädt beim Erscheinen des Fensters die zuletzt eingestellte Schriftgröße aus den User Defaults.
- SettingsGeneralVC mit dem View-Controller zur Dialogseite ALLGEMEINES synchronisiert das Textfeld zur Veränderungen der Schriftgröße mit dem Splitter, den Einstellungen in den User-Defaults und der Schriftgröße im Hauptfenster.

# Application Defaults mit den User-Defaults verbinden (AppDelegate.swift)

Die NSUserDefaults-Klasse bietet die Möglichkeit, mit der Methode registerDefaults ein Dictionary mit Defaulteinstellungen festzulegen. Die dort enthaltenen Einstellungen kommen zum Einsatz, wenn noch nie Benutzereinstellungen gespeichert wurden oder es für den betreffenden Schlüssel keinen Eintrag gibt. Im Beispielprogramm wird dieses Dictionary aus der Datei appdefaults.plist extrahiert. Damit können die Defaulteinstellungen sehr komfortabel direkt in Xcode eingetragen und verwaltet werden (siehe Abbildung 19.6).

● ● ● ▶ ■ ♠)➡ Finished running osx-tabviewcontroller : osx-tabviewcontroller				
	🔡 🛛 < 🔰 🤷 osx-tabviewcontroller	osx-tabviewcontroller ) 🚞 Sup	porting Files $ angle$ in appdefaults.plist $ angle$ No Selection	
🔻 🔄 osx-tabviewcontroller	Key	Type Value		
osx-tabviewcontroller	▼ Root	Dictionary (1 item)		
AppDelegate.swift	fontsize	Number 12		
ViewController.swift				
SettingsGeneralVC.swift				
images.xcassets				
💽 Main.storyboard				
🔻 🚞 Supporting Files				
📄 appdefaults.plist				
info.plist				
Products				
+ 🖲 🕓 🕬				

Abbildung 19.6 Die Datei appdefaults.plist mit Defaulteinstellungen für das Programm

Die Verbindung zwischen den User-Defaults und den dazugehörigen Defaultwerten muss ganz früh unmittelbar nach dem Programmstart durchgeführt werden, also noch bevor Window- oder View-Controller geladen werden. Aus diesem Grund wurde in AppDelegate.swift die Init-Funktion der AppDelegate-Klasse überschrieben. Würde der selbe Code in den Methoden applicationWillFinishLaunching oder applicationDidFinishLaunching eingebaut, wäre es zu spät, die Defaultwerte blieben unberücksichtigt.

Der eigentliche Code ist unspektakulär: standardUserDefaults liefert eine Referenz auf die immer zur Verfügung stehende NSUserDefaults-Instanz des Programmes. URLForResource ermittelt den Pfad zu der mit dem Programm mitgelieferten Bundle-Datei appdefaults.plist. Die Init-Funktion der NSDirectory-Klasse bildet aus dieser Datei ein NSDirectory. Dieses wird dann mit der schon erwähnten Methode registerDefaults verarbeitet.

Die AppDelegate-Klasse enthält darüber hinaus die Eigenschaft mainVC. Sie wird später in viewDidLoad der View-Controller-Klasse initialisiert und gibt dann allen Klassen des Programms die Möglichkeit, über das AppDelegate-Objekt auf die Steuerelemente des Hauptfensters zuzugreifen.

```
// Projekt osx-tabviewcontroller
// Datei AppDelegate.swift
@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {
    // Verweis auf den View-Controller des Hauptfensters
    var mainVC:ViewController!
    // Defaultwerte für die Benutzereinstellungen laden
    override init() {
      super.init()
```

```
let userDefaults = NSUserDefaults.standardUserDefaults()
if let url = NSBundle.mainBundle().URLForResource(
    "appdefaults", withExtension: "plist"),
    appDefaults = NSDictionary(contentsOfURL: url)
    {
        userDefaults.registerDefaults(
            appDefaults as! [String : AnyObject])
    }
}
```

### Textgröße aus den User-Defaults lesen (ViewController.swift)

Die Klasse ViewController enthält die beiden Eigenschaften userDefaults und app, die auf Instanzen von AppDelegate und der NSUserDefaults zeigen. Die beiden Ausrufezeichen bei der Initialisierung der app-Eigenschaft sind normalerweise etwas, was den erfahrenen Swift-Programmierer nervös macht. In diesem Fall besteht aber kein Grund zur Sorge: Die delegate-Eigenschaft hat zwar den Typ NSApplicationDelegate?, aber so wie das Programm aufgebaut ist, *muss* diese Eigenschaft eine Instanz der gerade oben beschriebnen AppDelegate-Klasse verweisen.

In viewDidLoad wird zuerst die Eigenschaft mainVC der AppDelegate-Instanz initialisiert. Danach wird aus den User-Defaults die zuletzt gespeicherte Schriftgröße gelesen. Ein Aufruf von setFontSize verändert die Schrift. Dabei wird ein neues NSFont-Objekt erzeugt, das bis auf die Größe die anderen Eigenschaften der bisherigen Schrift übernimmt.

```
// Projekt osx-tabviewcontroller, Datei ViewController.swift
class ViewController: NSViewController {
    @IBOutlet var txtfield: NSTextView!
    // Zugriff auf die User-Defaults und die AppDelegate-Instanz
    var userDefaults =
        NSUserDefaults.standardUserDefaults()
    let app =
        NSApplication.sharedApplication().delegate! as! AppDelegate
    override func viewDidLoad() {
        app.mainVC = self
        // Schriftgröße aus Defaults laden und setzen
        let size = userDefaults.integerForKey("fontsize")
        setFontSize(size)
    }
```

```
// Schriftgröße von txtfield ändern
func setFontSize(size:Int) {
   txtfield.font = NSFont(
     descriptor: txtfield.font!.fontDescriptor,
     size: CGFloat(size))
}
```

### Andere Schriftattribute ändern

Um aus einer vorhandenen Schrift ein neues NSFont-Objekt mit einer anderen Schriftgröße zu machen, gibt es den die im obigen Listing eingesetzte Init-Funktion. Wenn Sie andere Schriftattribute ändern möchten, setzen Sie am besten einen NSFont-Manager ein:

```
let fontmanager = NSFontManager.shareFontManager()
let newfont = fontmanager.convertFont(oldfont,
    toHaveTrait: NSFontTraitMask.BoldFontMask)
```

### Einstellungen ändern (SettingsGeneralVC.swift)

Auch die SettingsGeneralVC-Klasse beginnt mit der schon bekannten Definition der Variablen app und userDefaults. In viewDidLoad wird die delegate-Eigenschaft des Textfeldes auf self gesetzt. Damit kommt es bei Änderungen im Textfeld zum Aufruf der Methode controlTextDidChange.

Außerdem lädt userForKey den zuletzt gespeicherten Wert für die Textgröße und ruft setAndSaveFontSize auf. Diese Methode synchronisiert das Textfeld, den Splitter, die Textgröße im Hauptfenster und den Wert in den User-Defaults. Auch die Methoden sizestepper und controlTextDidChange rufen diese Methode auf, sobald ein Benutzer einen neuen Wert im Textfeld eingibt oder diesen durch die Pfeil-Buttons vergrößert oder verkleinert.

```
// Projekt osx-tabviewcontroller, Datei SettingsGeneralVC.swift
class SettingsGeneralVC: NSViewController, NSTextFieldDelegate {
  var app = NSApplication.sharedApplication().delegate!
    as! AppDelegate
  var userDefaults =
    NSUserDefaults.standardUserDefaults()
@IBOutlet weak var sizestepper: NSStepper!
@IBOutlet weak var fntsize: NSTextField!

  override func viewDidLoad() {
    super.viewDidLoad()
    fntsize.delegate = self
```

```
// aktuelle Schriftgröße
    let size = userDefaults.integerForKey("fontsize")
    setAndSaveFontSize(size)
 }
 // Änderungen an den Steuerelementen
 @IBAction func sizestepper(sender: NSStepper) {
    setAndSaveFontSize(Int(sizestepper.stringValue)!)
 }
 override func controlTextDidChange(obj: NSNotification) {
    if let newsize = Int(fntsize.stringValue.) {
      setAndSaveFontSize(newsize)
    }
  }
 // Schriftgröße in den Steuerelementen des Settings-Dialogs
 // und im Hauptfenster einstellen, außerdem in den
 // User-Defaults speichern
  private func setAndSaveFontSize(size:Int) {
    sizestepper.stringValue = "\(size)"
    fntsize.stringValue = "\(size)"
    userDefaults.setInteger(size, forKey: "fontsize")
    app.mainVC.setFontSize(size)
 }
}
```

# User-Defaults-Interna

Die User-Defaults eines OS-X-Programms werden in einer binären \*.plist-Datei im Library-Verzeichnis gespeichert. Der Dateiname dieser Datei ergibt sich aus dem Bundle Identifier des Programms. Beim Beispielprogramm ergibt sich der folgende Dateiname:

```
~/Library/Preferences/info.kofler.osx-tabviewcontroller.plist
```

Im Terminal können Sie mit defaults read die ganze Datei in Textform anzeigen. Dabei ist es nicht notwendig, den Pfad und die Kennung .plist anzugeben.

```
defaults read info.kofler.osx-tabviewcontroller
{
   fontsize = 12;
}
```

Mit dem defaults-Kommando können Sie auch einzelne Werte auslesen oder verändern. Details über den Umgang mit diesem Kommando können Sie im Terminal mit man defaults ergründen.

# 19.3 Standarddialoge

Nicht jeden Dialog müssen Sie selbst programmieren. Für einige, häufig vorkommende Aufgaben stellt das AppKit-Framework fertige Dialoge zur Verfügung. Dazu zählen:

- die Anzeige von Nachrichten und einfachen Auswahldialogen (JA, NEIN, ABBRECHEN etc.)
- ▶ die Auswahl einer vorhandenen Datei oder eines Verzeichnisses zum Öffnen
- ▶ die Auswahl einer neuen Datei zum Speichern
- die Auswahl einer Schrift
- die Auswahl einer Farbe

Das Beispielprogramm osx-dialogs zeigt die Anwendung dieser Standarddialoge (siehe <u>Abbildung 19.7</u>). Ein wenig befremdlich ist dabei, dass die dazu erforderlichen Techniken je nach Dialog stark variieren. Der gesamte Code befindet sich in ViewController.swift. Der Code beginnt mit Outlets für die fünf Ergebnis-Label und die sogenannte »Color Well«:

```
// Projekt osx-dialogs, Datei ViewController.swift
class ViewController: NSViewController {
    @IBOutlet weak var txtMessage: NSTextField!
    @IBOutlet weak var txtFilename: NSTextField!
    @IBOutlet weak var txtFont: NSTextField!
    @IBOutlet weak var txtColor: NSTextField!
    @IBOutlet weak var colorwell: NSColorWell!
```

```
// ... diverse Methoden, Details folgen gleich
}
```



Abbildung 19.7 Demonstration verschiedener Standarddialoge

### Nachrichten anzeigen und Ja/Nein-Entscheidungen treffen

Zur Anzeige von Nachrichten, die üblicherweise mit einem OK-Button zu bestätigen sind, und einfachen Dialogen zur Entscheidung zwischen JA/NEIN oder anderen Optionen verwenden Sie eine Instanz der NSAlert-Klasse. Bevor Sie den Dialog mit runModal anzeigen, stellen Sie mit der Eigenschaft messageText die Nachricht ein. addButtonWithTitle fügt beliebig viele Buttons hinzu, wobei der erste Button ganz rechts angezeigt wird.

Die Nachrichtenbox zeigt normalerweise links das Programm-Icon an. Über die icon-Eigenschaft wählen Sie bei Bedarf ein anderes Bild aus Images.xcassets. Mit der zusätzlichen Einstellung alertStyle = CriticalAlertStyle erreichen Sie, dass Ihr Icon um ein Warndreick erweitert wird.

runModal blockiert das restliche Programm, bis der Dialog geschlossen ist. Die Methode gibt für die ersten drei Button NSAlertFirst-, NSAlertSecond- bzw. NSAlert-ThirdButtonReturn zurück. Gibt es mehr Buttons, müssen Sie deren Rückgabewerte selbst mit NSAlertThirdButtonReturn+n errechnen.

```
// Projekt osx-dialogs, Datei ViewController.swift
// Fortsetzung
class ViewController: NSViewController {
  @IBAction func btnMessage(sender: NSButton) {
    let alert = NSAlert()
    alert.messageText = "Wollen Sie das Objekt löschen?"
    alert.addButtonWithTitle("Abbrechen")
    alert.addButtonWithTitle("Löschen")
    alert.icon = NSImage(named: "warning") // aus Images.xcassets
    switch alert.runModal() {
    case NSAlertFirstButtonReturn:
      txtMessage.stringValue = "Abbrechen"
    case NSAlertSecondButtonReturn:
      txtMessage.stringValue = "Löschen"
    default:
      break
    }
 }
}
```

### Datei- und Verzeichnisauswahl

Zur Datei- und Verzeichnisauswahl stehen die Klassen NSOpenPanel bzw. NSSavePanel zur Auswahl. Obwohl die mit runModal angezeigten Dialoge ganz ähnlich aussehen, gibt es doch einige grundlegende Unterschiede. So unterstützt nur NSOpenPanel eine Mehrfachauswahl und die Auswahl von Verzeichnissen. Dafür können Sie mit NSFilePanel einen Dateinamen angeben, den es noch gar nicht gibt. Tun Sie dies nicht, erscheint eine Rückfrage, ob Sie die vorhandene Datei überschreiben möchten.

Der Umgang mit den beiden Panel-Klassen ist ganz ähnlich wie beim NSAlert: Sie stellen zuerst einige Eigenschaften ein, öffnen den Dialog dann mit runModal und werten danach die Eigenschaft URL oder URLs aus. Die folgenden Zeilen zeigen den Umgang mit dem NSSavePanel:

```
// Projekt osx-dialogs, Datei ViewController.swift
// Fortsetzung
class ViewController: NSViewController {
  @IBAction func btnFileSave(sender: NSButton) {
    let saveFile = NSSavePanel()
    saveFile.title = "Datei speichern"
    saveFile.prompt = "Speichern"
    saveFile.worksWhenModal = true
    saveFile.canCreateDirectories = true
    saveFile.runModal()
    if let url = saveFile.URL, fname = url.path {
        txtFilename2.stringValue = fname
    }
    }
}
```

Auf die Wiedergabe des ganz ähnlichen Codes für das NSOpenPanel habe ich hier verzichtet – werfen Sie gegebenenfalls einen Blick in die Beispieldateien zum Buch. Wie Sie die Dateiauswahl durch die richtige Einstellung der allowedFileTypes-Eigenschaft auf bestimmte Dateitypen einschränken, zeigt ein Beispiel in <u>Abschnitt 20.7</u>, »Drag & Drop-Empfänger für Icons«.

# Schrift einstellen

Jedem OS-X-Programm steht eine Instanz des NSFontManager zur Verfügung, der bei allen erdenklichen Aufgaben im Zusammenhang mit Schriften hilft. Der Zugriff auf die Instanz erfolgt über die statische Methode sharedFontManager. Sie können mit convertFont aus einer vorhandenen Schrift eine neue mit einem veränderten Attribut erstellen, mit traitsOfFont die Attribute eines gegebenen NSFont-Objekts herausfinden, mit availableFonts alle verfügbaren Fonts auflisten etc.

Für uns ist hier aber die Methode orderFrontFontPanel am interessantesten. Sie zeigt den aus allen OS-X-Programmen bekannten Dialog zur Schriftauswahl an, der intern durch die NSFontPanel-Klasse realisiert ist. Beachten Sie, dass es sich dabei nicht um einen modalen Dialog handelt. Das Hauptprogramm bleibt weiter benutzbar, der Schriftendialog kann parallel dazu offen bleiben, solange der Benutzer dies wünscht.

Vor dem Aufruf von orderFrontFontPanel müssen Sie die Eigenschaften delegate und target auf self stellen, damit in Ihrer View-Controller-Klasse die Methode changeFont aufgerufen wird. Zumeist ist es außerdem zweckmäßig, mit setSelectedFont die gerade aktuelle Schrift voreinzustellen.

Ein wenig merkwürdig ist die richtige Vorgehensweise in der Methode changeFont: Der Font-Manager stellt Ihnen die gerade ausgewählte Schrift nicht als fertiges NSFont-Objekt zur Verfügung. Stattdessen müssen Sie ausgehend von einem beliebigen vorhandenen NSFont-Objekt mit convertFont ein neues Objekt erzeugen, das der durchgeführten Auswahl entspricht.

```
// Projekt osx-dialogs, Datei ViewController.swift
// Fortsetzung
class ViewController: NSViewController {
  // Font-Dialog anzeigen
  @IBAction func btnChooseFont(sender: NSButton) {
    let fontmanager = NSFontManager.sharedFontManager()
    fontmanager.delegate = self
    fontmanager.target = self
    fontmanager.setSelectedFont(txtFont.font!, isMultiple: false)
    fontmanager.orderFrontFontPanel(self)
  }
  // Reaktion auf Auswahl einer neuen Schriftart
  override func changeFont(sender: AnyObject?) {
    let oldfont = txtFont.font!
    let newfont = NSFontManager.sharedFontManager()
      .convertFont(oldfont)
    txtFont.font = newfont
  }
}
```

Beachten Sie, dass changeFont von nun an immer wieder aufgerufen wird, wenn der Benutzer Änderungen im Schriftendialog durchführt. Wenn Sie in Ihrem Programm verschiedene Schriften einstellen können, müssen Sie in changeFont darauf Rücksicht nehmen.

### Farbe einstellen

Das Beispielprogramm zeigt gleich zwei Varianten, um die Farbe eines Textfelds einzustellen:

Der Button FARBE AUSWÄHLEN führt zum Aufruf der Action-Methode btnChoose-Color. Dort greift der Code über sharedColorPanel auf die in jedem OS-X-Programm vorhandene Instanz des NSColorPanel zu. Die Methoden setTarget und setAction geben an, dass bei einer Farbauswahl die Methode colorSelect der eigenen Klasse aufgerufen werden soll. makeKeyAnd-OrderFront zeigt das Color Panel an. Bei der Farbauswahl können Sie in der durch setAction definierten Methode über die color-Eigenschaft des NSColorPanel die ausgewählte Farbe auslesen.

Noch deutlich einfacher ist es, zur Farbauswahl das Color-Well-Steuerelement zu verwenden (Klasse NSColorWell). Das Steuerelement zeigt einen farbigen Button an. Beim Anklicken erscheint automatisch der Farbauswahldialog. Jedes Mal, wenn der Benutzer eine Farbe auswählt, kommt es zum Aufruf der zugeordneten Action-Methode, in der das als sender übergebene NSColorWell-Objekt die ausgewählte Farbe verrät.

```
// Projekt osx-dialogs, Datei ViewController.swift
// Fortsetzung
class ViewController: NSViewController {
 // Farbauswahl mit Button
  @IBAction func btnChooseColor(sender: NSButton) {
    let colorpanel = NSColorPanel.sharedColorPanel()
    colorpanel.setTarget(self)
    colorpanel.setAction("newColor:")
    colorpanel.makeKeyAndOrderFront(self)
  }
  func newColor(sender:AnyObject?) {
    let colorpanel = NSColorPanel.sharedColorPanel()
    txtColor.textColor = colorpanel.color
    // mit Color Well synchronisieren
    colorwell.color = colorpanel.color
 }
 // Farbauswahl mit Color Well
  @IBAction func colorSelect(sender: NSColorWell) {
    txtColor.textColor = sender.color
  }
}
```

# 19.4 Maus

In vielen Fällen müssen Sie sich um Maus- oder Trackpad-Ereignisse gar nicht kümmern: Das gerade angeklickte Steuerelement erhält den Fokus, es werden Button-Ereignisse ausgelöst, Scrollbalken lassen sich verschieben, Text markieren etc. – alles ohne eine Zeile Code. Aber wie immer gibt es Ausnahmen, und zwar insbesondere dann, wenn Sie mit View- oder Image-View-Elementen arbeiten: Diese Steuerelemente sehen nämlich keine Actions zur Verarbeitung von Mausklicks vor, und es gibt auch kein Delegate-Protokoll mit entsprechenden Methoden. Vielmehr müssen Sie eine eigene, von NSView oder NSImageView abgeleitete Klasse erzeugen und diese dann im Identity Inspector als CUSTOM CLASS angeben.

### Mausereignisse

In der von NSView abgeleiteten Klasse können Sie nun die folgenden Methoden überschreiben:

- ▶ mouseEntered: Der Mauszeiger wurde in das Steuerelement hineinbewegt.
- ▶ mouseDown: Die linke Maus- oder Trackpad-Taste wurde gedrückt.
- ▶ mouseDragged: Die Maus wurde bei gedrückter Maustaste bewegt.
- ▶ mouseUp: Die Maustaste wurde losgelassen.
- mouseExited: Der Mauscursor hat das Steuerelement verlassen oder befindet sich momentan über einem anderen Steuerelement innerhalb der View.

Die Methoden mouseDown, mouseDragged und mouseUp gelten nur für die linke Maustaste. Für die rechte gibt es drei weitere Methoden:

- rightMouseDown: Die rechte Maus- oder Trackpad-Taste wurde gedrückt.
- ▶ rightMouseDragged: Die Maus wurde bei gedrückter Maustaste bewegt.
- ▶ rightMouseUp: Die Maustaste wurde losgelassen.

An all diese Methoden wird ein NSEvent-Objekt übergeben, aus dem die Mausposition sowie der Status der Zustandstasten (), (alt) und () ermittelt werden kann. Bevor wir uns mit der Mausposition beschäftigen können, sind aber noch einige Grundlagen zum Koordinatensystem erforderlich, das innerhalb eines NSView-Steuerelements gilt.

### Koordinatensysteme, Bounds und Frames

Grundsätzlich gilt sowohl für das Fenster als Ganzes als auch für jedes einzelne darin enthaltene Steuerelement ein Koordinatensystem, dessen Ursprung in der Ecke links unten ist (siehe <u>Abbildung 19.8</u>). Als Einheit werden Punkte verwendet, wobei aber bei Retina-Bildschirmen in jede Richtung zwei Pixel gezeichnet werden. Sehr hilfreich ist in diesem Zusammenhang, dass Koordinaten und Größen schon seit jeher mit CGFloat-Zahlen ausgedrückt werden, also mit Fließkommazahlen. Intern handelt es sich bei CGFloat-Zahlen unter OS X um Double-Zahlen, weil es alle aktuelle Versionen von OS X nur noch in 64-Bit-Versionen gibt.

Jedes Steuerelement gibt anhand von zwei Eigenschaften Auskunft über seine Position und Größe:



Abbildung 19.8 Koordinatensystem des Fensters und einer darin enthaltenen View

- ▶ bounds enthält NSRect-Daten im Koordinatensystem des Steuerelement.
- ▶ frame enthält ein weiteres NSRect-Element, das aber das Koordinatensystem des Containers verwendet, also der View, in der das Steuerelement enthalten ist.

Grundsätzlich gilt, dass keine der beiden Eigenschaften das Koordinatensystem des Fensters verwendet! Die frame-Koordinaten stimmen nur dann mit den Fensterkoordinaten überein, wenn sich Steuerelemente direkt in der ersten View des Fensters befinden, die dieses vollständig ausfüllt.

#### NSPoint, NSSize, NSRect versus CGPoint, CGSize, CGRect

Beim Umgang mit bounds und frame werden Sie mit drei Datenstrukturen konfrontiert, die Sie auch sonst häufig benötigen. NSPoint enthält einen Koordinatenpunkt, dessen Position aus den Eigenschaften x und y hervorgeht. NSSize enthält eine Größenangabe, die durch die Eigenschaften width und height definiert ist. Ein NSRect -Element beschreibt Position und Größe eines Rechtecks. Intern setzt es sich aus einem NSPoint- und einem NSSize-Element zusammen. Die beiden Strukturen sprechen Sie über die Eigenschaften origin und size an.

NSPoint, NSSize und NSRect sind als typealias der aus Core Graphics stammenden Strukturen CGPoint, CGSize und CGRect definiert. Deswegen können Sie zwischen diesen Strukturen beliebig wechseln.

### Mausposition ergründen

Das an die diversen mouse-Methoden übergebene NSEvent-Objekt verrät über die Eigenschaft locationInWindow die Position des Mauscursors. Wie der Name der Eigenschaft schon vermuten lässt, wird die Position des Mauscursors im Koordinatensystem des Fensters angegeben.

Häufig werden Sie die Koordinatenposition aber im Koordinatensystem der NSView-Klassen benötigen, die Sie implementieren. Zur Umrechnung stellt die NSView-Klasse die Methode convertPoint zur Verfügung. An diese Methode übergeben Sie im ersten Parameter die Position. Im zweiten Parameter können Sie ein anderes NSView-Objekt angeben, wenn die ursprüngliche Position relativ zu diesem Objekt ist. Bei Positionen im Fensterkoordinatensystem übergeben Sie hier einfach nil.

```
class MyView: NSView {
  override func mouseDown(theEvent: NSEvent) {
    let locationInView =
        convertPoint(theEvent.locationInWindow, fromView:nil)
    ...
  }
}
```

### Statustasten

In vielen Programmen müssen in Mausereignissen auch die Zustandstasten  $\textcircledline (tri), (alt) und (tri) berücksichtigt werden. Das NSEvent-Objekt stellt Ihnen ent$ sprechende Informationen in der modifierFlags-Eigenschaft als OptionSet (siehe<u>Abschnitt 4.4</u>, »Option-Sets (OptionSetType)«) zur Verfügung. Um zu testen, ob einbestimmtes Flag gesetzt ist, verwenden Sie am besten die contains-Methode:

if mod.contains(.ShiftKeyMask) { ... } // Shift ist gedrückt

### Beispielprogramm



Abbildung 19.9 Kreise zeichnen mit der Maus

Das Beispielprogramm definiert außer den vom Xcode vorgesehenen Klassen die eigene Klasse MyView sowie die Datenstruktur Circle, um die Daten eines Kreises zu speichern. Der Swift-Compiler erzeugt für die Circle-Struktur eine Init-Funktion, an die die Parameter in der Reihenfolge übergeben werden, in der sie definiert sind.

```
// Projekt osx-mouse, Datei Circle.swift
struct Circle {
   var x:CGFloat // x-Koordinate
   var y:CGFloat // y-Koordinate
   var radius:CGFloat // Radius
   var lineWidth: CGFloat // Linienstärke
   var color:NSColor // Farbe
}
```

Die MyView-Klasse ist von NSView abgeleitet. Im Identity Inspector ist diese Klasse als CUSTOM CLASS für das View-Steuerelement im Zentrum des View-Controllers eingestellt (siehe Abbildung 19.10).



Abbildung 19.10 Das Storyboard des Beispielprogramms

### **Die MyView-Klasse**

Die MyView-Klasse hat zwei Aufgaben:

- ► Einerseits reagiert sie auf Mausklicks. Sie ermittelt die Position und fügt dann dem Array circles ein neues Circle-Element mit den Eckdaten des Kreises hinzu.
- Andererseits zeichnet sie den Inhalt der View bei Bedarf neu also immer dann, wenn ein neuer Kreis hinzukommt, wenn das Fenster vorübergehend verdeckt war oder wenn sich die Fenstergröße ändert.

Bei Änderungen der Fenstergröße bleibt die Position der bisher gezeichneten Kreise unverändert. Das ist insofern nicht selbstverständlich, als eine direkte Interpretation der Koordinaten dazu führen würde, dass die Kreise bei einer Vergrößerung des Fensters nach unten wandern würden. Deswegen speichert das Programm die Y-Koordinate so, als würde sich der Koordinatenursprung oben befinden. Beim Zeichnen wird die Y-Koordinate dann wieder in das OS-X-Koordinatensystem gerechnet.

Auf noch ein Detail möchte ich hinweisen: Die vielleicht naheliegendere Lösung der Aufgabenstellung bestünde darin, einfach in der mouseDown-Methode einen Kreis an der Mausposition zu zeichnen. Das ist aber zum einen gar nicht vorgesehen; gelänge es doch, ergibt sich der Nachteil, dass die Zeichenoperation nicht nachhaltig ist. Das Programm und somit auch die selbst implementierte View muss ja zu einem späteren Zeitpunkt in der Lage sein, seinen bzw. ihren Inhalt wieder neu zu zeichnen. Aus diesem Grund speichert das Programm Position, Größe, Farbe etc. jedes Kreises in einem Array von Circle-Strukturen.

Die Klasse MyView ist mit dem Attribut @IBDesignable gekennzeichnet. Das bedeutet, dass Xcode die Klasse sofort kompiliert und das Steuerelement im Storyboard-Editor so anzeigt, wie es später aussehen wird. In unserem Fall bewirkt dies, dass das Steuerelement mit weißem und nicht wie sonst üblich mit grauem Hintergrund dargestellt wird.

```
// Projekt osx-mouse, Datei MyView.swift
@IBDesignable
class MyView: NSView {
  var circles = [Circle]()
  // Code folgt gleich
  override func drawRect(dirtyRect: NSRect) { ... }
  override func mouseDown(theEvent: NSEvent) { ... }
}
```

### Die drawRect-Methode

In eigenen Steuerelementen muss die drawRect-Methode überschrieben werden, um den Inhalt des Steuerelements zu zeichnen. In diesem Buch gab es dafür bei der iOS-Programmierung ja schon mehrere Beispiel, z. B. beim Steuerelement zur Richtungsanzeige in der »Schatzsuche«-App (siehe Abschnitt 15.4).

Bei diesem Beispiel ist die Aufgabenstellung einfach: Zuerst wird mit NSRectFill ein weißer Hintergrund gezeichnet; danach wird für jeden Kreis aus dem circles-Array ein NSBezierPath-Objekt erzeugt und gezeichnet. Der Code sollte auch ohne Hintergrundwissen in der Grafikprogrammierung plausibel sein. Beachten Sie die Berechnung der Y-Koordinate des Kreismittelpunkts: Diese ergibt sich aus der Innengröße der View minus dem im Circle-Element gespeicherten Wert.

Eine denkbare Optimierung bestünde darin, vorweg bei jedem Kreis zu überprüfen, ob dieser überhaupt innerhalb des als dirtyRect-Parameter vorgegebenen Zeichenbereichs liegt (Bounding-Box-Test). Wenn sehr viele komplexe Grafikelemente effizient dargestellt werden sollen, könnte das zu einem flüssigeren Bildaufbau führen. Bei diesem Beispiel lohnt sich die Mühe aber nicht.

```
// Projekt osx-mouse, Datei MyView.swift
@IBDesignable
class MyView: NSView {
  var circles = [Circle]()
  override func drawRect(dirtyRect: NSRect) {
    // weißer Hintergrund
    NSColor.whiteColor().setFill()
    NSRectFill(dirtyRect)
    // falls nicht NSView als Basisklasse:
    // super.drawRect(dirtyRect)
    // Vordergrund
    for c in circles {
      let path = NSBezierPath()
      path.appendBezierPathWithArcWithCenter(
        NSPoint( x: c.x,
                 y: bounds.size.height - c.y),
            radius: c.radius,
        startAngle: 0,
          endAngle: 360)
      c.color.set()
      path.lineWidth = c.lineWidth
      path.stroke()
    }
  }
}
```

### Aufruf von super.drawRect

Bei Steuerelementen, die direkt von NSView abgeleitet sind, enthält die drawRect-Methode der Basisklasse keinen Code. Ein Aufruf ist daher überflüssig.

Bei anderen Steuerelementen müssen Sie aber in Ihrer eigenen drawRect-Methode super.drawRect aufrufen, wenn das Steuerelement zuerst seinen eigenen Inhalt zeichnen soll. Der Aufruf von super.drawRect sollte nach dem Zeichnen eines eigenen Hintergrunds erfolgen, aber vor weiteren Grafikausgaben, die zuletzt über dem Originalsteuerelement sichtbar sein sollen.

### Die mouseDown-Methode

In mouseDown wird die Position des Mauscursors in das View-Koordinatensystem umgerechnet. Die Y-Koordinate wird außerdem so umgerechnet, dass sie den Abstand von oben und nicht von unten bestimmt. Je nachdem, ob die 🔄-Taste gedrückt wird oder nicht, verwendet die Methode eine unterschiedliche Linienstärke. Das so erzeugte Kreiselement wird nun dem circles-Array hinzugefügt.

Jetzt könnten wir es uns einfach machen und mit der Anweisung

```
setNeedsDisplayInRect(bounds)
```

einfach ein Neuzeichnen des gesamten Steuerelements auslösen. Aber stellen Sie sich vor, das Programm läuft im Vollbildmodus auf einem Retina-iMac mit rund 15 Millionen Pixel: Der gesamte Fensterinhalt würde neu gezeichnet, obwohl sich nur ein vergleichsweise kleiner Bereich geändert hat. Deswegen ist es hier wirklich zweckmäßig, ein NSRect-Element zusammenzusetzen, das den Kreis umhüllt. Dabei dürfen Sie die Linienstärke nicht vergessen, sonst wird der Kreis beim Zeichnen an den Rändern beschnitten! Wichtig ist auch, dass die Methode setNeedsDisplayInRect die Y-Koordinate natürlich im Koordinatensystem der View erwartet.

```
// Projekt osx-mouse, Datei MyView.swift, Fortsetzung
class MyView: NSView {
 override func mouseDown(theEvent: NSEvent) {
   let locationInView =
      convertPoint(theEvent.locationInWindow, fromView:nil)
   let x = locationInView.x
   let y1 = locationInView.y
                                 // OS-X-Koordinatensystem
   let y2 = bounds.size.height - y1 // eigenes Koordinatensys.
   let r = CGFloat(20)
   let lw:CGFloat
   if theEvent.modifierFlags.contains(.ShiftKeyMask) {
     lw = CGFloat(4)
   } else {
     lw = CGFloat(2)
   }
   let color = NSColor.redColor()
   circles.append(Circle(x: x, y: y2, radius: r,
                          lineWidth: lw, color: color))
```

Wenn Sie durch eine Mausbewegung mit gedrückter Maustaste viele Kreise hintereinander zeichnen möchten, rufen Sie mouseDown einfach auch aus der mouseDragged-Methode auf. Auf die mouseDragged-Methode werden wir dann wieder in <u>Abschnitt 20.2</u>, »Drag & Drop«, stoßen, wo sie verwendet wird, um eine Drag & Drop-Operation zu initiieren.

```
// bei gedrückter Maustaste viele Kreise zeichnen
override func mouseDragged(theEvent: NSEvent) {
   mouseDown(theEvent)
}
```

# 19.5 Tastatur

Mit Tastaturereignissen ist es ähnlich wie mit Mausereignissen: In vielen Fällen kümmert sich OS X bzw. das AppKit-Framework um deren Verarbeitung, z. B. in Textfeldern. Es gibt aber zwei Ausnahmen:

In Textfeldern wollen Sie mitunter sofort bei jeder Eingabe auf diese reagieren, z. B. um bestimmte Eingaben zu unterbinden oder um andere Objekte zu synchronisieren. In solchen Fällen implementieren Sie in der View-Controller-Klasse das NSTextFieldDelegate-Protokoll, setzen die delegate-Eigenschaft des Textfelds auf self und können dann in diversen controlText-Methoden auf Textereignisse reagieren.

Ein Beispiel für diese Vorgehensweise gibt die Klasse SettingsGeneralVC.swift im Projekt osx-tabviewcontroller, das ich in <u>Abschnitt 19.2</u>, »Tab-View-Controller«, beschrieben habe.

Bei Steuerelementen, die von sich aus keine Tastatureingaben verarbeiten, müssen Sie eine eigene Klasse mit diversen Responder-Methoden programmieren. Der Umgang mit der NSResponder-Klasse steht im Mittelpunkt dieses Abschnitts.

### Die NSResponder-Klasse

Die NSView-, die NSWindow- und die NSApplication-Klasse haben eine Gemeinsamkeit: Sie sind alle von der NSResponder-Klasse abgeleitet. Das gibt ihnen die Möglichkeit, zu einem sogenannten »First Responder« zu werden. Mit diesem Begriff wird dasjenige Objekt bezeichnet, das als erstes Tastatureingaben, Menükommandos etc. verarbeitet. In jedem Fenster kann immer nur ein Objekt bzw. Steuerelement der First Responder sein. Es ist aber erlaubt, dass mehrere Fenster jeweils ihren eigenen First Responder aufweisen. In diesem Fall verarbeitet das gerade aktive Fenster die Eingaben.

Wenn der Benutzer in einem Fenster mit mehreren Textfeldern eines davon anklickt, dann muss zuerst das gerade aktive Steuerelement seinen First-Responder-Status abgeben. Anschließend muss das angeklickte Steuerelement den First-Responder-Status akzeptieren. Erst danach kann das Textfeld Eingaben empfangen. In anderen GUI-Frameworks würde man sagen: Es hat den Eingabefokus erhalten.

Damit Ihre eigene View-Klasse als First Responder agieren kann, sind nur wenige Zeilen Code erforderlich:

```
class MyView: NSView {
  override var acceptsFirstResponder:Bool { return true }
  override func becomeFirstResponder() -> Bool {
    return true
  }
  override func resignFirstResponder() -> Bool {
    return true
  }
}
```

Eine kurze Erklärung:

- Die Read-only-Eigenschaft acceptsFirstResponder muss den Wert true zurückgeben. Damit bringt das NSView-Objekt zum Ausdruck, dass es bereit ist, als First Responder Tastaturereignisse zu verarbeiten.
- ► Die Methode becomeFirstResponder wird aufgerufen, wenn die View zum First Responder wird. Diese Methode gibt üblicherweise true zurück, d.h., die View akzeptiert den First-Responder-Status.

In Fenstern mit mehreren Steuerelementen ist es zumeist zweckmäßig, das Steuerelement jetzt zu kennzeichnen. Beispielsweise können Sie in Ihrer Klasse eine highlight-Eigenschaft definieren, diese in drawRect auswerten und dort bei Bedarf einen Rahmen rund um das Steuerelement zeichnen. Ein Beispiel für diese Vorgehensweise finden Sie in <u>Abschnitt 20.2</u>, »Drag & Drop«. Die Methode resignFirstResponder wird aufgerufen, wenn die View den First-Responder-Status wieder verliert. Wenn das aus irgendeinem Grund gerade nicht möglich ist, können Sie das durch die Rückgabe von false verhindern.

## Tastaturereignisse

Sobald Ihre View der First Responder ist, führt jede Tastatureingabe zum Aufruf der folgenden Methoden:

- ► keyDown: Eine Taste wurde gedrückt. Diese Methode wird mehrfach aufgerufen, wenn die Taste länger gedrückt bleibt (Auto-Repeat).
- ▶ keyUp: Eine Taste wurde losgelassen.
- ▶ flagsChanged: Die Zustandstasten 🄄, [ctrl], [alt] oder 🕷 haben sich geändert.

An diese Methoden wird jeweils ein NSEvent-Objekt übergeben, das Ihnen von den Mausereignissen ja schon vertraut ist. Sie können nun einige tastaturspezifische Eigenschaften dieses Objekts auswerten:

- ► characters enthält eine Zeichenkette mit dem eingegebenen Zeichen.
- ► keyCode enthält den Code der gedrückten Taste.
- modifierFlags enthält eine Kombination von Werten, die den Zustand von 

   <u>ctrl</u>, <u>alt</u> oder <u>s</u> widerspiegeln.

Es ist grundsätzlich möglich, in keyDown diese Eigenschaften auszuwerten und im Programm dann entsprechend darauf zu reagieren. Allerdings bereitet speziell die Interpretation des keyCode-Werts Probleme. Das Cocoa-Framework enthält nämlich keine Enumeration bzw. keine Konstanten mit den zulässigen Werten.

Einfacher ist es in der Regel, das NSEvent-Objekt an die Methode interpretKeyEvents zu übergeben. Dann kümmert sich diese Methode um die Auswertung der Eingabe. In der Folge kommt es für jede denkbare Aktion zur Ausführung einer entsprechenden Methode. Die Dokumentation der NSResponder-Klasse zählt über 80 derartige Methoden auf. Zu ihnen zählen:

- cancelOperation: Der Vorgang soll abgebrochen werden ([esc]).
- ► deleteXxx: Es sollen Daten gelöscht werden.
- ▶ insertText: Es wurde ein Zeichen eingegeben.
- ▶ insertXxx: Es sollen Sonderzeichen eingefügt werden, z. B. ein Tabulatorzeichen.
- ▶ moveXxx, pageXxx: Die Cursorposition soll verändert werden.
- ► scrollXxx: Der sichtbare Ausschnitt der Daten soll verändert werden.
- ► selectXxx: Der ausgewählte Bereich der Daten soll verändert werden.

### Beispielprogramm

Das Beispielprogramm zu diesem Abschnitt zeigt das Icon eines Hockey-Spielers. Dessen Position können Sie mit den Cursortasten oder mit den Tasten I, J, K und M steuern. Bei einer Bewegung nach links ändert das Icon-Symbol sogar seine Richtung – der Hockey-Schläger zeigt also immer nach vorne.



Abbildung 19.11 Beispielprogramm zur Verarbeitung von Tastaturereignissen

### Verwenden Sie Sprite Kit zur Spielprogrammierung!

Der spielerische Charakter dieses Beispiels soll Sie keineswegs dazu animieren, selbst auf dieser Basis Spiele zu programmieren! Dazu gibt es eigene Bibliotheken, unter anderem das sogenannte Sprite Kit.

Das Beispielprogramm besteht lediglich aus einem Fenster mit der standardmäßig bereits enthaltenen View. Dessen CUSTOM CLASS wurde auf MyView umgestellt. Bis auf wenige Zeilen in viewDidLoad des View-Controllers befindet sich der gesamte Code dieses Beispiels in MyView.swift.

Bei der Initialisierung eines MyView-Objekts werden aus Images.xcassets zwei Icons mit den nach vorne bzw. nach hinten laufenden Hockey-Spieler geladen. Die Klasseneigenschaften x und y bestimmen die aktuelle Position des Spielers, img verweist auf das gerade gültige Icon. x, y und img werden in viewDidLoad der View-Controller-Klasse mit Startwerten belegt.

```
// Projekt osx-keyboard, Datei MyView.swift
class MyView: NSView {
    // Bitmaps sind im Images.xcassets
    let forward = NSImage(named: "hockey")!
    let backward = NSImage(named: "hockey-back")!
    let size:CGFloat = 64 // Größe des Icons
```

```
// Eigenschaften
                   // x-Koordinate
 var x:CGFloat!
 var v:CGFloat!
                   // y-Koordinate, von oben gerechnet!
 var img:NSImage!
 // als First Responder auftreten
 override var acceptsFirstResponder:Bool { return true }
 override func becomeFirstResponder() -> Bool {
   return true
 }
 override func resignFirstResponder() -> Bool {
   return true
 }
 // ... weitere Methoden, Details folgen
}
```

drawRect zeichnet das Hockey-Icon mit der Methode drawInRect an die aktuelle Position. Vorher stellen zwei verschachtelte min/max-Funktionen sicher, dass sich der Hockey-Spieler innerhalb der View befindet. Die Kontrolle an dieser Stelle vereinfacht nicht nur den restlichen Code, sondern ist auch bei einer Veränderung der Fenstergröße wirksam. Dabei kommt es nämlich immer zu einem drawRect-Aufruf.

Beim Drücken einer Taste wird die Methode keyDown aufgerufen. Sie übergibt das NSEvent-Objekt direkt an interpretKeyEvents. Wenn diese Methode das Drücken der Cursortasten erkennt, kommt es zum Aufruf der vier move-Methoden. Eingaben von Buchstaben führen zum Aufruf von insertText, wo die Tasten I, J, K und M wie Cursortasten behandelt werden. In den move-Methoden wird der Einfachheit halber ein Neuzeichnen der gesamten View ausgelöst.
```
// Projekt osx-keyboard, Datei MyView.swift, Fortsetzung
class MyView: NSView {
  // Eingabe auswerten
  override func keyDown(theEvent: NSEvent) {
    interpretKeyEvents([theEvent])
  }
  override func insertText(insertString: AnyObject) {
    if let input = insertString as? String {
      switch input.lowercaseString {
      case "j":
        moveLeft(self)
      case "k":
        moveLeft(self)
      case "i":
        moveUp(self)
      case "m":
        moveDown(self)
      default:
        break
      }
    }
  }
  // Figur bewegen
  override func moveLeft(sender: AnyObject?) {
    x = x - 10
    img = backward
    setNeedsDisplayInRect(bounds)
  }
  override func moveRight(sender: AnyObject?) {
    x = x + 10
    img = forward
    setNeedsDisplayInRect(bounds)
  }
  override func moveDown(sender: AnyObject?) {
    y = y + 10
    setNeedsDisplayInRect(bounds)
  }
  override func moveUp(sender: AnyObject?) {
    y = y - 10
    setNeedsDisplayInRect(bounds)
  }
}
```

# 19.6 Menüs

Menüs sind ein zentrales Steuerungselement vieler OS-X-Programme, wenngleich viele Programme für eine möglichst menülose Bedienung optimiert sind. Dieser Abschnitt gibt einen Überblick über die verschiedenen Erscheinungsformen von Menüs und über deren Anwendung in eigenen Programmen. Das dazugehörende Beispielprogramm osx-menu besteht aus zwei Fenstern. Das Menü EIGENES MENÜ besteht aus den folgenden Einträgen:

- ► TEST 1 ist immer verwendbar, die Action-Methode befindet sich in AppDelegate.
- ► TEST 2 ist nur verwendbar, wenn Fenster 1 aktiv ist. Die Action-Methode befindet sich in der ViewController-Klasse.
- ► TEST 3 und TEST 4 sind verwendbar, wenn eine Instanz von Fenster 2 aktiv ist. TEST 4 zeigt ein Auswahlhäkchen an, das durch die Menüauswahl gesetzt bzw. wieder entfernt wird. Die Action-Methoden befinden sich im ViewController2.

Innerhalb des zweiten Fensters steht außerdem ein Kontextmenü mit drei Einträgen zur Verfügung (siehe Abbildung 19.12).



Abbildung 19.12 Menübeispielprogramm

#### **Die Responder-Kette**

Wie die Verbeitung von Menüaktionen vor sich geht, lässt sich nur mit etwas Grundwissen über den sogenannten First Responder und die nachfolgende Responder-Kette begreifen. Auf den First Responder sind Sie ja im vorigen Abschnitt schon gestoßen: So wird das Objekt bezeichnet, das Tastaturereignisse empfangen und verarbeiten kann. Oft handelt es sich dabei um ein Textfeld, aber prinzipiell sind alle Objekte von Klassen dazu in der Lage, die von NSResponder abgeleitet sind. Zu diesen Klassen zählen:

- ► NSView und somit alle Steuerelemente
- NSViewController
- NSWindow und NSWindowController
- NSDocumentController (hilft bei der Konzeption von Programmen, die in ihren Fenstern Dokumente verwalten – also Editoren jeder Art, Office-Programme etc.)
- NSApplication

Der Nachrichtenfluss beginnt beim First Responder. Ist dieser an der Nachricht nicht interessiert, leitet er die Nachricht an das nächste Objekt in der Responder-Kette weiter (siehe <u>Abbildung 19.13</u>). Dieser Nachrichtenfluss endet, wenn ein Objekt das Ereignis verarbeitet. »Eine Nachricht empfangen« bedeutet, dass eine entsprechende Methode aufgerufen wird.

Wie die Responder-Kette ausgeprägt ist, hängt von der Gestaltung des Programmes ab. Gibt es einen Window-Controller? Gibt es View-Controller? Gibt es einen Document-Controller? Sind Window-Delegates implementiert? Wie verschachtelt sind die Steuerelemente im Fenster?

Zudem ändert sich die Responder-Kette während der Programmausführung ständig: Mal ist das eine Fenster mit einem Textfeld aktiv, dann wieder ein Dialog mit anderen Steuerelementen. Sicher ist nur: Am Ende der Responder-Kette steht immer das NSApplication-Objekt sowie dessen Delegates, in einem typischen Storyboard-Projekt also die AppDelegate-Klasse.



Abbildung 19.13 Eine mögliche Responder-Kette für Ereignisse

Und warum ist dies für Menüs relevant? Weil auch Menüereignisse, also die Auswahl eines Menükommandos, über die Responder Chain zuerst an den First Responder geleitet werden und anschließend über alle anderen Objekte der Responder-Kette.

#### Gestaltung der Menüleiste

Neue Xcode-Projekte enthalten eine umfangreiche vordefinierte Menüleiste. Nicht benötigte Menüs oder Menüeinträge klicken Sie einfach an und löschen sie mit —.

Wenn Sie neue Menüs hinzufügen möchten, suchen Sie in der Objektbibliothek nach *menu*. Lassen Sie sich von den vielen Ergebnissen nicht irritieren – die meisten Einträge sind einfach vordefinierte Menüs, z. B. das Font-Menü zur Veränderung der Schrift. Vorerst reicht es aus, wenn Sie zwischen zwei Menüklassen differenzieren:

- ► NSMenu (MENU in der Objektbibliothek) ist ein Menü, also eine aus mehreren Einträgen bestehende Box.
- ▶ NSMenuItem (MENU ITEM) ist ein *Eintrag* eines Menüs, also z. B. SPEICHERN.

Um also einen zusätzlichen Eintrag in ein schon vorhandenes Menü einzubauen, verschieben Sie per Drag & Drop ein MENU ITEM an die gewünschte Stelle im Menü. Während des Verschiebevorgangs klappen die Menüs nach einigen Sekunden bei Bedarf auf. Nach dem Einfügen verändern Sie den Menütext per Doppelklick oder im Attributinspektor. Auch das zugeordnete Tastenkürzel kann direkt im Menü verändert werden. Ein Doppelklick im rechten Bereich eines Menüeintrags zeigt eine kleine, rechteckige Box. Sie können nun eine beliebige Tastenkombination drücken, die dann als Kürzel eingetragen wird. Unter OS X übliche Tastenkürzel sollten Sie möglichst vermeiden.

#### Programmname-Menü

Der Name des ersten Menüs in der Menüleiste ist durch den Programmnamen vorgegeben. Sie können zwar im Storyboard-Editor einen anderen Namen einstellen, im laufenden Programm wird dann aber doch der Programmname angezeigt.

Um ein ganzes Menü einzufügen, verschieben Sie in SUBMENU MENU ITEM aus der Objektbibliothek in die Menüleiste. Dabei handelt es sich in Wirklichkeit um drei Objekte: um ein NSMenuItem mit dem Eintrag für die Menüleiste (Beschriftung MENU), um ein NSMenu mit dem neuen Menü und um ein darin befindliches NSMenuItem (Beschriftung (MENUITEM).

Wenn Sie Ihr Programm nun starten, werden Sie feststellen, dass Ihre eigenen Menüeinträge in grauer Schrift angezeigt werden und nicht verwendbar sind. Das liegt daran, dass die Einträge im aktuellen Kontext, also für das geöffnete Fenster, nicht mit einer Aktion verbunden sind. (Was bedeutet hier »im aktuellen Kontext«? Es ist möglich, dass ein Menü mit einer Methode eines View-Controllers verbunden ist. Dadurch ist der Menüeintrag dann verwendbar, wenn ein Fenster mit diesem View-Controller das aktive Fenster ist und der View-Controller oder ein darin enthaltenes Steuerelement der First Responder ist, also Eingaben verarbeitet. Klicken Sie ein anderes Fenster an, wird der Menüeintrag wieder grau.)

# **Responder-Aktionen**

Natürlich soll nach der Auswahl eines Menüeintrags auch etwas passieren. Einige Menüeinträge des Default-Menüs sind standardmäßig schon mit Aktionen verbunden.

Um eigene Menüeinträge mit vordefinierten Responder-Aktionen zu verbinden, zeichnen Sie mit <u>ctrl</u>-Drag eine Verbindungslinie vom Menüeintrag zum orangenen Icon FIRST RESPONDER in der Titelleiste des Menüs. In einem Popup-Dialog stehen weit über 100 Aktionen zur Auswahl (siehe <u>Abbildung 19.14</u>). Es handelt sich dabei um alle Action-Methoden der Systemklassen und des Projekts. Um eine Aktion auszuwählen, tippen Sie am besten deren Anfangsbuchstaben ein.



Abbildung 19.14 Menüeintrag mit Responder-Aktion verbinden

Bei der Aktionsliste handelt es sich um eine Zusammenstellung der Methoden der NSResponder-, NSDocumentController- und NSApplication-Klassen sowie um @IBAction-Methoden Ihrer eigenen Klassen. Die Zusammenstellung ist insofern irreführend, als die aufgelisteten Methoden nur dann tatsächlich ausgeführt werden können, wenn das betreffende Objekt auch Teil der Responder-Kette ist. Bei vielen Methoden ist dies aber normalerweise nicht der Fall – z. B. wenn es in Ihrem Programm gar keinen NSDocumentController gibt.

# Menüaktionen in der AppDelegate-Klasse

In der Regel wollen Sie eigene Menüeinträge mit eigenen Methoden verbinden. Naheliegend wäre es, wie bei Steuerelementen mit <u>ctrl</u>-Drag eine Action-Methode in der View-Controller-Klasse einzufügen. Leider unterstützt Xcode das nicht. Wie Sie dennoch Menüaktionen auf View-Controller-Ebene verarbeiten können, erkläre ich Ihnen gleich. Vorerst konzentrieren wir uns aber auf den Weg, den Xcode vorsieht: Sie öffnen im Haupteditor das Storyboard und im Assistenzeditor AppDelegate.swift und stellen dann mit <u>ctrl</u>-Drag eine Verbindung her. Auf diese Weise können Sie sowohl Actions als auch Outlets einfügen. Da sich die AppDelegate-Klasse am Ende der Responder-Kette befindet (siehe <u>Abbildung 19.13</u>), ist sichergestellt, dass die Aktion auf jeden Fall ausgeführt wird – ganz egal, welches Fenster bzw. ob überhaupt ein Fenster geöffnet ist bzw. welches Steuerelement gerade als First Responder agiert.

Leider ist die AppDelegate-Klasse für viele Aktionen nicht der am besten geeignete Ort – vor allem, wenn das Menükommando eigentlich ein Fenster oder einen Bereich eines Fensters betrifft. Wenn es in Ihrem Programm ohnedies nur ein einziges Fenster gibt, macht die Verwendung der AppDelegate-Klasse für Menükommandos den Code auch nicht wesentlich unübersichtlicher. Für Mehr-Fenster-Anwendungen ist die hier skizzierte Vorgehensweise aber weder empfehlenswert noch sinnvoll durchführbar.

Die Kompromisslösung sieht so aus, dass Sie in der AppDelegate-Klasse eine Eigenschaft zu definieren, die auf den View-Controller Ihres einzigen Fensters zeigt. In viewDidLoad des View-Controllers initialisieren Sie diese Eigenschaft. Damit können Sie nun von der AppDelegate-Klasse auf alle Eigenschaften und Methoden des View-Controllers zugreifen. Das folgende Listing mit Code aus beiden Klassen veranschaulicht den Lösungsweg.

```
// Projekt osx-menu, Datei AppDelegate.swift
@NSApplicationMain
class AppDelegate: NSObject, NSApplicationDelegate {
  // Zugriff auf den View-Controller des Fensters
  var mainVC:ViewController?
  // Action-Methode für einen Menüeintrag
  @IBAction func menuTest1(sender: NSMenuItem) {
    mainVC?.mylabel.stringValue = "Test 1"
  }
}
// Projekt osx-menu, Datei ViewController.swift
class ViewController: NSViewController {
  @IBOutlet weak var mylabel: NSTextField!
  override func viewDidLoad() {
    super.viewDidLoad()
    let app =
      NSApplication.sharedApplication().delegate as! AppDelegate
    app.mainVC = self
  }
}
```

#### Segues

Menüs können per <u>ctrl</u>-Drag direkt mit einem View-Controller verbunden werden. In diesem Fall wird der betreffende Controller bei einer Menüauswahl in einem Fenster angezeigt. Ein Beispiel für einen derartigen Segue finden Sie in <u>Abschnitt 19.2</u>, »Tab-View-Controller«.

#### Menüaktionen in eigenen View-Klassen

Selbstverständlich ist es möglich, Menüeinträge mit Action-Methoden zu verbinden, die in einem View-Controller definiert sind. Dazu müssen Sie die Methode aber selbst eintippen, ein <u>ctrl</u>-Drag vom Menüeintrag in das Code-Fenster funktioniert nicht. Vergessen Sie nicht, dass an die Action-Methode ein Parameter vom Typ NSMenuItem übergeben wird.

```
// Projekt osx-menu, Datei ViewController.swift
// Fortsetzung
class ViewController: NSViewController {
  @IBOutlet weak var mylabel: NSTextField!
  // Action-Methode für 'Test 2'
  @IBAction func menuTest2(sender: NSMenuItem) {
    mylabel.stringValue = "Test 2"
  }
}
```

Die Verbindung vom Menüeintrag zur gerade verfassten Action-Methode stellen Sie her, indem Sie einen <u>ctrl</u>-Drag vom Menüeintrag zum orangen Icon FIRST RESPON-DER des Menüfensters durchführen. In der endlosen Liste der Responder-Aktionen ist nun auch menuTest2 enthalten – diese Methode wählen Sie aus.

Es gibt eine wichtige systembedingte Einschränkung: Der so verbundene Menüeintrag kann nur ausgewählt werden, wenn ein Steuerelement im View-Controller der First Responder ist – denn nur dann ist der View-Controller ein Glied der Responder-Kette! In der Praxis heißt das:

- Das Menükommando ist nur dann aktiv, wenn es ein Objekt in der Responder-Kette mit einer passenden Action-Methode gibt. Ist das nicht der Fall, wird der Menüeintrag in grauer Schrift angezeigt und kann nicht ausgewählt werden.
- ► Das Menükommando ist auch dann nicht benutzbar, wenn Ihr View-Controller gar kein Steuerelement enthält, das ein First Responder ist.

Letzteres Problem tritt am ehsten in Testprogrammen auf, wenn ein Fenster z. B. nur einen Label enthält. Aber natürlich gibt es auch »reale« Anwendungen, bei denen ein Fenster ohne Buttons, Textfelder etc. auskommt. Zum Glück gibt es eine einfache Lösung: Sie implementieren einfach eine von NSView abgeleitete Klasse mit First-Responder-Funktionen (siehe <u>Abschnitt 19.5</u>, »Tastatur«) und ordnen diese Klasse der ersten View im View-Controller zu.

Das Beispielprogramm zu diesem Abschnitt demonstriert diese Vorgehensweise beim zweiten Fenster. Der Code der Klasse MyView für die View im View-Controller 2 sieht so aus:

```
// Projekt osx-menu, Datei MyView.swift
class MyView: NSView {
   override var acceptsFirstResponder:Bool { return true }
   override func becomeFirstResponder() -> Bool {
     return true
   }
   override func resignFirstResponder() -> Bool {
     return true
   }
}
```

#### Veränderung von Menüeinträgen per Code

An die Action-Methode wird das NSMenuItem-Objekt im sender-Parameter übergeben. Das gibt Ihnen die Möglichkeit, den Menüeintrag in der Methode zu verändern. Ein einfaches Beispiel dafür ist das Anzeigen bzw. Entfernen eines Auswahlhäkchen vor dem Menüeintrag.

```
// Projekt osx-menu, Datei ViewController.swift
class ViewController2: NSViewController {
   @IBOutlet weak var label: NSTextField!
   // Action-Methode für den Menüeintrag 'Test 4'
   @IBAction func menuTest4(sender: NSMenuItem) {
    label.stringValue = "Test 4"
    // Auswahlhäkchen setzen/entfernen
    if sender.state == NSOffState {
      sender.state = NSOffState
    }
   else {
      sender.state = NSOffState
    }
}
```

Auf die gesamte Menüleiste können Sie über die menu-Eigenschaft des NSApplication-Objekts zugreifen. Mit addItem können Sie nun bei Bedarf NSMenu und NSMenuItemObjekte hinzufügen und auf diese Weise dynamische Menüs realisieren. Ein Beispiel für ein per Code erzeugtes Menü finden Sie in <u>Abschnitt 19.7</u>, »Programme ohne Menü«. Dort geht es darum, eine sogenannte »Menubar-App« zu programmieren, also ein Programm ohne ein reguläres Menü, aber dafür mit einem Icon im rechten Bereich der Menüleiste.

#### Menüeintrage je nach Kontext aktivieren oder deaktivieren

Wenn Sie Menüeinträge per Code aktivieren bzw. deaktivieren möchten, implementieren Sie hierfür das NSMenuValidation-Protokoll. Dessen Methode validateMenu-Item wird aufgerufen, bevor ein Menüeintrag sichtbar wird. Je nachdem, in welchem Zustand sich Ihr Programm gerade befindet, können Sie nun entscheiden, ob der Menüeintrag aktiv sein soll oder nicht. Dementsprechend muss Ihre Methode true oder false zurückgeben.

# Kontextmenüs

Um ein Kontextmenü zu definieren, ziehen Sie ein MENU aus der Objektbibliothek in den Titelbereich des View-Controllers oder an die entsprechende Stelle in der Document Outline. Auch die Bearbeitung der Einträge muss dort erfolgen. Eine Menüvorschau wie beim Hauptmenü gibt es also nicht. Dafür können Sie für die Menüeinträge unkompliziert per <u>ctrl</u>-Drag Outlets und Actions in die View-Controller-Klasse einfügen (siehe <u>Abbildung 19.15</u>).



Abbildung 19.15 Action-Methode für einen Kontextmenüeintrag einrichten

Damit das Kontextmenü automatisch angezeigt wird, sobald der Anwender die rechte Maustaste bzw. die linke Taste zusammen mit <u>ctrl</u> drückt, müssen Sie nur die menu-Eigenschaft der View einstellen, beispielsweise in viewDidLoad. Im Beispielprogramm gibt es im zweiten Fenster ein Kontextmenü. Alle drei Einträge dieses Menüs sind mit derselben Methode menuContext verbunden:

```
// Projekt osx-menu, Datei ViewController.swift
// Fortsetzung
class ViewController2: NSViewController {
   @IBOutlet weak var label: NSTextField!
   @IBOutlet var contextMenu: NSMenu!
   override func viewDidAppear() {
     // ...
     view.menu = contextMenu
   }
   // Action-Methode für alle drei Kontextmenüeinträge
   @IBAction func menuContext(sender: NSMenuItem) {
     label.stringValue = sender.title
   }
}
```

# 19.7 Programme ohne Menü

Für Programme, die überwiegend im Hintergrund laufen, besteht die Möglichkeit, auf ein Menü ganz zu verzichten. Dazu reicht es aus, in der Projektdatei Info.plist den Eintrag APPLICATION IS AGENT (UIELEMENT) = YES einzufügen (siehe <u>Abbildung 19.16</u>). Das Programm lässt sich weiter ganz normal starten und bedienen. Es besitzt nun aber weder ein eigenes Menü noch wird es im Dock angezeigt.



Abbildung 19.16 Menü ausblenden mit einer Info.plist-Einstellung

Im Prinzip sind Sie damit auch schon fertig. Sie müssen nur darauf achten, dass beim Schließen des einzigen bzw. letzten Fensters das Programm beendet wird – andernfalls läuft es im Hintergrund weiter, ohne dass es eine Möglichkeit gibt, es wieder zu aktivieren.

# **Menubar-Apps**

In der Praxis werden menülose Programme zumeist als sogenannte »Menubar-Apps« ausgeführt. Solche Programme präsentieren sich im rechten Bereich der Menüleiste, also dort, wo die Uhrzeit angezeigt wird, in Form eines kleinen Icons. Das Anklicken dieses Icons führt wahlweise in ein kleines Menü zur Steuerung der Programmfunktionen oder zeigt einfach das Programmfenster an.

Der rechte Bereich der Menüleiste wird »Menubar« genannt. Etwas inkonsequent ist es, dass die Namen der dafür verantwortlichen Klassen alle mit NSStatusBar beginnen. Das folgende Minibeispiel besteht aus einem Fenster, das mit einem Button aus- und über das Menü des Menubar-Icons wieder eingeblendet werden kann (siehe Abbildung 19.17).

	Fe Be	nster an enden	zeigen	J					
			Menubar	-App					
				Ausble	enden	E)			
<u> </u>	_			_	_	-	-		

Abbildung 19.17 Eine winzige Menubar-App

Die Gestaltung von Menubar-Apps erfolgt weitestgehend durch Code. Dem Storyboard-Editor von Xcode fehlen momentan Werkzeuge zur visuellen Gestaltung von Status-Menüs. Das hat aber den Vorteil, dass dieser Abschnitt mir die Gelegenheit bietet, Ihnen zu zeigen, wie Sie eigene Menüs per Code zusammensetzen.

# Die AppDelegate-Klasse

Der Großteil des Codes befindet sich in AppDelegate.swift. Dort wird, wie schon in vielen Beispielen, eine Variable definiert, die auf den View-Controller des einzigen Fensters des Programms zeigt. Die Variable wird später in viewDidLoad des View-Controllers initialisiert.

statusBarItem enthält ein neues NSStatusBarItem-Objekt, das mit der Methode statusItemWithLength erzeugt wurde. In applicationDidFinishLaunching wird dieser Eintrag in der Menubar mit einem Icon aus Images.xcassets ausgestattet. Die Bitmap sollte 16 × 16 Pixel groß sein, die Retina-Variante (x2) 32 × 32 Pixel.

Die weiteren Zeilen erzeugen ein neues Menü mit zwei Einträgen und verbinden dieses mit dem neuen Statusbar-Element. Die action-Eigenschaften verweisen jeweils auf die Methoden, die bei einer Menüauswahl auszuführen sind. showMyWindow greift über die Eigenschaften view und window auf das einzige Fenster des Programms zu und bringt dieses mit makeKeyAndOrderFront in den Vordergrund.

```
// Projekt osx-menubar, Datei AppDelegate.swift
class AppDelegate: NSObject, NSApplicationDelegate {
 var mainVC:ViewController!
 let statusBarItem =
    NSStatusBar.systemStatusBar().statusItemWithLength(-1)
  func applicationDidFinishLaunching(aNotification:
   NSNotification) {
    // Icon für MenuBar-Eintrag
    let icon = NSImage(named: "flag") // aus Images.xcassets
    statusBarItem.image = icon
    // Menü zusammensetzen
    let menu: NSMenu = NSMenu()
    var menuItem = NSMenuItem()
    menuItem.title = "Fenster anzeigen"
    menuItem.action = "showMyWindow:"
    menu.addItem(menuItem)
    menuItem = NSMenuItem()
    menuItem.title = "Beenden"
    menuItem.action = "quit:"
    menu.addItem(menuItem)
    // Menü mit Statusbar-Element verbinden
    statusBarItem.menu = menu
 }
 // Reaktion auf Menüeintrag
 func showMyWindow(sender:NSMenuItem) {
    // Fenster anzeigen
    mainVC?.view.window?.makeKeyAndOrderFront(self)
  }
  func quit(sender:NSMenuItem) {
    NSApplication.sharedApplication().terminate(self)
 }
}
```

#### **View-Controller**

Der View-Controller initialisiert die mainVC-Eigenschaft der AppDelegate-Klasse. Die Action-Methode btnHide zeigt das Gegenstück zu makeKeyAndOrderFront: Die Methode orderOut blendet das Fenster aus.

```
class ViewController: NSViewController {
  override func viewDidLoad() {
    super.viewDidLoad()
    let app =
        NSApplication.sharedApplication().delegate as! AppDelegate
        app.mainVC = self
    }
    // Fenster ausblenden
    @IBAction func btnHide(sender: NSButton) {
        view.window?.orderOut(self)
    }
}
```

# 19.8 Bindings

Bindings bieten die Möglichkeit, den Zustand bzw. Inhalt von Steuerelementen mit Eigenschaften zu verbinden. Das Steuerelement und die Eigenschaft werden also synchronisiert – und das, ohne eine Zeile Code zu schreiben. Über den NSUser-DefaultsController können auch in den User-Defaults gespeicherte Einstellungen mit Steuerelementen verbunden werden.

Das Bindings-Framework geht noch viel weiter. Werte können nicht nur 1:1 weitergereicht, sondern auch formatiert, umgerechnet oder umgewandelt werden. Eine recht umfassende Einführung, deren PDF-Version fast 100 Seiten umfasst, finden Sie auf der Apple-Developer-Seite:

```
https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/
CocoaBindings
```

Bindings sind einerseits faszinierend, andererseits aber auch ziemlich komplex in ihrer Anwendung. Ihre unsichtbare Natur macht nicht nur die Fehlersuche schwierig, sondern auch die Orientierung in fremden Projekten. Dort eingesetzte Bindings müssen Sie in Xcode richtiggehend suchen. Insofern bin ich außer in trivialen Fällen kein großer Fan von Bindings und schreibe lieber die erforderlichen Zeilen Code in Action-Methoden.

#### **Hello Bindings!**

Gewissermaßen das »klassische« Einführungsbeispiel zum Umgang mit Bindings sind ein Slider, ein Textfeld und eine Eigenschaft, die alle miteinander verbunden werden (siehe <u>Abbildung 19.18</u>). Wird eine Variable im Code geändert, passen sich die Steuerelemente entsprechend an. Analog führt das Verschieben des Sliders oder die Eingabe einer neuen Zahl dazu, dass sowohl das jeweils andere Steuerelement als auch die Eigenschaft verändert wird. Der RESET-Button setzt die Variable zurück auf O.

• • •	Hello Bindi	ngs!
-	0	47
		Reset

Abbildung 19.18 Simple Bindings-Demonstration

Um das auszuprobieren, starten Sie ein neues Cocoa-Projekt mit der Option USE STORYBOARDS. In den View-Controller fügen Sie nun einen Slider und ein Textfeld ein, in die ViewController-Klasse die Eigenschaft myvalue. Außerdem fügen Sie mit [ctr1]-Drag eine Action-Methode für den RESET-Button in den Code ein. Die Methode setzt myValue auf O und beweist, dass die Synchronisation in beide Richtungen funktioniert.

```
// Projekt osx-bindings-hellow, Datei ViewController.swift
class ViewController: NSViewController {
   dynamic var myValue = 50 // Startwert
   @IBAction func btnReset(sender: NSButton) {
     myValue = 0
   }
}
```

#### Vergessen Sie »dynamic« nicht!

Das Schlüsselwort dynamic ist hier bei der Definition der Eigenschaft myValue unbedingt erforderlich! Die Kennzeichnung macht die Eigenschaft kompatibel zu Objective C und führt dazu, dass an myValue durchgeführte Änderungen an die verbundenen Steuerelemente gemeldet werden. Ohne dynamic funktioniert das Binding nur als Einbahnstraße, d. h. vom Steuerelement hin zur Eigenschaft, aber nicht in die andere Richtung.

Mit dem Code sind wir damit schon fertig – jetzt folgen noch einige Einstellungen in Xcode. Dazu klicken Sie zuerst den Slider an und aktivieren dann im Attributinspektor die Option CONTINUOUS. Sie bewirkt, dass der Slider seine Bindings bereits während des Verschiebens aktualisiert – und nicht erst zum Abschluss des Verschiebevorgangs. Anschließend wechseln Sie in den Bindings Inspector und wählen dort die Eigenschaft VALUE aus. Wir möchten also den aktuellen Wert des Sliders (und nicht eine der vielen anderen Eigenschaften) für das Binding verwenden. Von den unzähligen Einstellmöglichkeiten im Bindings Inspector müssen Sie nur zwei ändern (siehe Abbildung 19.19):

- ► BIND TO = VIEW CONTROLLER stellt die Verbindung zur View-Controller-Klasse her. (Alternativ würde BIND TO = SHARED USER DEFAULTS CONTROLLER bewirken, dass Sie eine Verbindung zu einem User-Defaults-Eintrag herstellen möchten.)
- ► MODEL KEY PATH = myValue gibt an, welche Eigenschaft dieser Klasse mit der Value-Eigenschaft des Sliders verbunden werden soll.



Abbildung 19.19 Einstellungen im Bindings Inspector

Ganz ähnlich nehmen Sie nun die Einstellungen für das Textfeld vor. Dort können Sie auf das Setzen der Option Continuous im Attributinspektor verzichten – sie ist für das Textfeld nicht relevant. Dafür müssen Sie aber im Bindings Inspector die Option CONTINUOUSLY UPDATE VALUE aktivieren. Ohne diese Option würden Texteingaben erst dann berücksichtigt, wenn das Textfeld den Eingabefokus verliert.

# Sonderfälle

Wenn Sie das Miniprogramm nun ausprobieren, werden Sie sehen, dass die Synchronisation zwischen den beiden Steuerelementen und der Variablen wunderbar funktioniert. Allerdings gibt es einige Spezialfälle, mit denen das Programm in der aktuellen Form nur schlecht zurechtkommt: Wenn Sie den Inhalt des Textfelds löschen oder dort Text eingeben, der sich nicht in eine Zahl umwandeln lässt, zeigt Xcode eine Fehlermeldung an. Immerhin läuft das Programm weiter.

Eine winzige Änderung am Code zeigt außerdem, dass auch Fließkommazahlen das einfache Beispiel an seine Grenzen bringen. Verschieben Sie nun den Slider, enthält das Textfeld die entsprechende Dezimalzahl mit dem bei uns üblichen Komma als Dezimaltrenner. Führen Sie aber im Textfeld selbst eine Eingabe durch, müssen Sie den in den USA gebräuchlichen Dezimalpunkt verwenden!

dynamic var myValue = 50.0

Eine weitere Änderung beweist, dass Bindings und Optionals inkompatibel zueinander sind:

```
// Fehler, Optionals dürfen nicht 'dynamic' sein
dynamic var myValue: Double! = 50.0
```

Verzichten Sie probeweise auf dynamic, zeigt Xcode beim Start eine Fehlermeldung an. Das Fenster erscheint zwar, aber ohne Steuerelemente.

```
// Fehler: this class is not key value coding-compliant
// for the key myValue
dynamic var myValue: Double! = 50.0
```

Die hier skizzierten Probleme lassen sich dadurch umschiffen, dass Sie für das Binding zusätzliche Validierungs- und Formatierungsregeln definieren. Das beweist einerseits die enorme Bandbreite der Binding-Möglichkeiten, deutet aber andererseits auch deren Komplexität in der Praxis an.

# Index

#### abs-Funktion ...... 194 Abstand zwischen zwei Koordinatenpunkten ...... 523 action-Eigenschaft action-Parameter addTarget ...... 384 Gesture Recognizer ..... 492 activateFileViewerSelectingURLs-Methode ...... 730 adaptivePresentationStyleForPresentation-Controller-Methode ..... 463 addAction-Methode ...... 471 addButtonWithTitle-Methode ...... 661 addObserver-Methode ..... 511 addSubview-Methode 384, 553, 598, 640, 732 advance ...... 109 Alert-Dialog ...... 470 alertStyle-Eigenschaft ...... 661 alignment-Eigenschaft ...... 702 Stack-View ...... 387 alpha-Eigenschaft ..... 572 AlternateKeyMask-Eigenschaft ...... 667 animateWithDuration ...... 569 animateWithDuration-Methode ...... 598 Animationen ...... 569 im Spiel 5-Gewinnt ..... 598 Any-Datentyp ..... 287 AnyClass-Datentyp ..... 289 AnyObject-Datentyp ..... 287 API-Version testen ...... 158 App Archiv erzeugen ...... 566 auf iOS-Geräten ausführen ...... 348 Hello World ...... 327 Icon ...... 556, 717

im App Store einreichen	566
im Simulator ausprobieren	332
Lebenszyklus	365
Lokalisierung	398
Name	557
Sprache	398
User-Defaults	391
weitergeben (iOS)	558
weitergeben (OS X)	742
Willkommensbildschirm	555
App Store	558
AppDelegate-Klasse	740
Init-Funktion	656
iOS	365
OS X	631
appdmg-Kommando	745
append-Methode	137
appendBezierPathWithArcWithCenter-	
Methode	669
appendBezierPathWithRoundedRect-	
Methode	709
appendNewline-Parameter	194
Apple Developer Program 30,	349
application-Methode	366
applicationDidBecomeActive-Methode	366
applicationDidEnterBackground-	
Methode	366
applicationDidFinishLaunching-	
Methode	741
applicationWillEnterForeground-	266
application Will Design Active Mathada	200
application will Resignative-Methode	300
Methode 366 632	740
ARC	127
Archiv (Xcode)	566
archiveBootObject-Methode	507
Arrays	133
assoziative	143
auslesen	135
Donnelaänger entfernen	141
durchwürfeln	141
filter man und reduce	140
initialisieren	134
mehrdimensionale 142	579
sortieren	138
verändern	137
	101
zweidimensionale	579

ArraySlice-Datentyp	136
ArrowView-Klasse (Beispiel)	513
as-Operator 77, 126, 268,	270
Aspect-Fill-Einstellung	383
Aspect-Fit-Einstellung	383
assert-Funktion	317
Assiciated Values	227
associativity	. 84
Assoziative Arrays	143
Assoziativität	. 82
Attribute	322
Aufräumarbeiten durchführen (defer)	177
Auto Layout 341,	368
deaktivieren	369
Maßeinheit	368
Regeln per Code definieren	384
View-Größe fixieren	648
Vorschau	371
autoclosure-Attribut	209
Automatic Reference Counting	127
availability-Attribut	322
available-Test	158
availableFonts-Methode	662

# B

Back-Button 42	3
Badge-Eigenschaft (Tab-Bar-Item) 42	8
Bar-Button-Item 42	3
Bechmarktests 11	7
becomeFirstResponder-Methode 498, 67	3
Bedingte Protokollerweiterungen	7
beginDraggingSessionWithItems-	
Methode 70	6
Benannte Parameter 174, 25	3
Funktionen 18	5
in Protokollen 27	5
Init-Funktion	2
Methoden 25	1
Benannte Typen 12	4
bezeled-Eigenschaft 70	2
Binäre Zahlen	2
Binärer Operator	0
Bitmaps	
als PNG-Datei speichern 72	7
Images.xcasset-Datei 432, 48	0
NSImage-Klasse	4
skalieren	4
UIImage-Klasse 48	0
Bitweises Rechnen	3
Bool-Datentyp	4
Boolesche Werte	4
borderColor-Eigenschaft	3
borderWidth-Eigenschaft	3
-	

5
5
5
9
5
0
4
0
2
0
0
0
4

# <u>C</u>

Cache-Verzeichnis	541
CALayer-Klasse	598
cancelOperation-Methode	674
canCreateDirectories-Eigenschaft	661
canEditRowIndexPath-Parameter	496
canMoveRowAtIndexPath-Parameter	497
Canvas-Value-Einstellung	374
Capabilities	440
Maps	435
capacity-Eigenschaft	139
capitalizedString-Eigenschaft	106
Capture List	213
Capturing Values	211
case-Schlüsselwort	158
Casting 126,	268
catch-Schlüsselwort 306,	308
cellForRowAtIndex-Parameter	476
CGBitmapContextCreate-Funktion	725
CGBitmapContextCreateImage-Funktion	725
CGColor-Struktur	363
CGContextAddArc-Methode	453
CGContextAddLineToPoint-Methode	450
CGContextDrawImage-Funktion	725
$CGContextDrawPath-Methode \$	453
$CGContextMoveToPoint-Methode\$	450
CGContextSetLineWith-Methode	450
CGContextSetStrokeColorWithColor-	
Methode	450
CGContextStrokePath-Methode	450
CGFloat-Datentyp	451
CGFloat-Typ	347
CGImage-Klasse	725
CGImageDestination-Klasse	727
CGImageDestinationFinalize-Funktion	727
$CGImageForProposedRect\text{-}Methode \$	725
CGPoint-Struktur	666

CGRect-Struktur	666,	702
CGRectMake-Funktion		385
CGRectMake-Methode	553,	702
CGSize-Struktur		666
changeFont-Methode		662
Character-Datentyp		. 94
characters-Eigenschaft	104.	674
children-Eigenschaft	······	126
class-Schlüsselwort	220.	250
CLLocation-Klasse	,	523
CLL ocation Manager-Klasse	437	442
Kompass	457,	447
teilen		509
CLL ocation Manager Delegate-		505
Protokoll	441	447
close-Methode	,	650
Closed-Range-Operator		78
Closed Interval Datentum		. 70
closeumerval-Datentyp		. 19
closePath-Methode		212
Closures		206
Auto-Closures		209
Capture List		213
Capturing Values		211
für Lazy Properties		231
in Animationen		569
Trailing Closures		208
UIAlert-Beispiel		471
unowned self	213,	571
verzögert ausführen		581
weak		213
Cocoa Touch		359
Cocoa-Framework		618
CollectionType-Protokoll		299
Color Panel		663
Color Well		663
color-Eigenschaft		663
CommandKeyMask-Eigenschaft		667
commitEditingStyle-Parameter		497
compare-Methode		100
CompassView-Steuerelement (Beispie	1)	449
Compiler	-,	44
completion-Parameter	571	610
components-Methode	115	147
components operated PuString Metho	11 <i>5</i> ,	106
Compound Types	Jue	124
Compression Basistance Driarity		200
Compression Resistance Phonity		202
Computed Properties	88,	235
Exterisions		295
renier ausiosen		313
vererbung		262
concludeDragOperation-Methode	705,	716
Connections Inspector	475,	696
constrainMaxCoordinate-Parameter .		732

constrainMinCoordinate-Parameter 732
Constraints (Auto Layout) 341, 368
Containment Segues 645
contains-Methode 145, 146, 198, 713
Content Compression Resistance
Priority
Content Hugging Priority 530
contentMode-Eigenschaft 383, 455
contents-Eigenschaft 711
contentView-Eigenschaft 640
continue-Schlüsselwort (Schleifen) 166
Continuous-Eigenschaft 690
Controller 329
ControllKeyMask-Eigenschaft 667
Convenience Init Function 244
Vererbung 265
convenience-Schlüsselwort 244
convertFont-Methode 663
convertPoint-Methode 666, 711, 735
Copy-on-Write (Zeichenketten) 96
Core Location (CL) 437
cornerRadius-Eigenschaft 363
count-Eigenschaft 135, 144
count-Funktion
count-Methode 197
Crashlog 407
createArray2D-Funktion 579
createDirectoryAtPath-Methode 741
CustomStringConvertible-Protokoll

# D

darker-Methode 53	80
Data-Source-Protokoll 475, 696, 7	00
dataSource-Eigenschaft	
UIPickerView 5	50
UITableView 4	74
dateByAddingComponents-Methode 1	16
Dateien	
auswählen 60	60
iOS	93
Datenquelle 475, 69	96
Datentypen 12	23
Aliase 12	25
ermitteln	25
Funktionstypen 20	02
Datum 1	15
decimalSeparator-Eigenschaft 54	47
decodeXxx-Methoden 5	07
default-Schlüsselwort (switch) 1	59
defaultCenter-Methode 5	10
defaultManager-Methode 74	41
defaults-Kommando 6	59
Defaultwerte für Parameter 1	87

defer-Schlüsselwort 177
in try-catch-Konstruktionen
Deinit-Funktion 128, 246
Dekrement-Operator
delay-Funktion 581
delegate-Eigenschaft
CLLocationManager 442
MKMapView 443
NSApplicationDelegate
NSFontManager 662
NSWindow
UIApplication
UIPickerView 550
UIPopoverPresentationController 463, 469
UITableView 474
UITapGestureRecognizer 548
UITextField 421, 547
Delegation 274, 355, 420
CLLocationManagerDelegate-Beispiel 441
Connections Inspector 475, 696
eigenes Delegation-Protokoll
NSTableView 696, 700
SetPieceDelegate-Beispiel 593
UITableView 474
UITextFieldDelegate-Beispiel 420
deleteRowsAtIndexPaths-Methode 497
deleteXxx-Methoden
dequeueReusableCellWithIdentifier-
Methode
description-Eigenschaft
Designated Init Function
Vererbung
desiredAccuracy-Eigenschaft
destinationViewController-Eigenschaft 417
Developer Program
Dictionaries
Hashable-Protokoll
didMoveToParentViewController-
didReceiveMemoryWarning-Methode 362
didSelectRow-Darameter 554
didSet-Funktion 232 262 491 544
<i>Poispiol</i> 452 601
Disk-Image 745
dismissController 649
dismissUiewControllerAnimated-
Methode
dispatch after-Funktion
distance-Funktion
distance Frenz Legation Matheda 522
distancefromLocation-Methode
Distribution Provisioning Profile

Dokumentverzeichnis (iOS)	393
domain-Eigenschaft	319
Double-Datentyp	92
Rundungsfehler	162
Double-Init-Funktion	114
doubleValue-Eigenschaft	114
Downcast	6, 268
Drag & Drop	704
Dateinamen empfangen	3, 736
Dateinamen senden	735
weitergeben	729
Zeichenkette empfangen	713
Zeichenkette senden	. 710
dragFile-Methode	735
draggingEntered-Methode	5, 737
draggingExited-Methode 70	5, 716
draggingPasteboeard-Methode	715
draggingSession-Methode 706, 712	2, 735
draggingUpdated-Methode	705
drand48-Funktion	4, 582
drawInRect-Methode	725
drawRect-Methode	9, 676
5-Gewinnt-Beispiel	. 596
Drag-und-Drop-Beispiel	709
Kompass	450
Schatzsuche/Richtungspfeil	513
drawsBackground-Eigenschaft	702
Dropdown-Liste	550
dropFirst-Funktion	191
dropLast-Funktion	191
dynamic-Schlüsselwort 224	4, 690

# E

editable-Eigenschaft	702
Eigenschaften	230
beobachten	232
Computed Properties	235
Extensions	295
Read-Only-Eigenschaft	236
statische Eigenschaften	234
Zugriff mit Optional Chaining	121
Eingabefokus einstellen	498
Einstellungsdialog (OS X)	651
Element-Typ	299
enabled-Eigenschaft	493
encodeXxx-Methoden	507
endedAtPoint-Parameter	706
endEditing-Methode 421,	548
enum-Schlüsselwort	. 89
Beispiele	577
Enumerationen	
als Datentyp	225
Assiciated Values	227

Beispiele	577, 722
Definition von Konstanten	89
indirekt/rekursiv	228
Equatable-Protokoll	285
als Extension implementieren	293
ErrorType-Protokoll	313
Eulersche Zahl	195
Exceptions	305, 320
exclusiveOr-Methode	146
Exit-Icon (ViewController)	414
Extended Grapheme Cluster	
Extensions	291
Beispiele	580
bequemer Zeichenkettenzugriff	109
extension-Schlüsselwort	292
Protokolle	296

# F

Fünf-Gewinnt-App	574
Fade-In-Effekt	571
Failable Init Functions	245
fallthrough-Schlüsselwort (switch)	159
false	94
Farben	
abdunkeln	580
aufhellen	580
auswählen	660
Fatal Error	305
Fehler	
Absicherung	305
auslösen (throw)	310
do-try-catch	306
NSError-Klasse	318
Weitergabe	315
Fenster	
ausblenden	689
Größe fixieren 623,	649
in den Vordergrund bringen	687
per Code erzeugen	650
schließen	649
fileExistsAtPath-Methode	741
fill-Methode	513
filter-Funktion	107
filter-Methode	140
final-Schlüsselwort 224	, 264
finally-Schlüsselwort	316
find-Funktion	550
Finder per Code anzeigen	730
First Responder 498	, 678
Menüs	681
Tastatur	673
first-Eigenschaft 136	, 197
first-Methode	108

flagsChanged-Methode 674
flatMap-Methode 199
Fließkommazahlen
Font auswählen 660
Font Manager 662
Font Panel
Font-Attribute ändern 658
font-Eigenschaft 347
fontWithSize-Methode
for-in-Schleife 165
for-Schleife 162
forced try 315
forEach-Methode 139, 200
ForwardIndexType-Protokoll 192
Foundation-Framework 40
frame-Eigenschaft
Frameworks
Xcode 435
Free Provisioning
func-Schlüsselwort 175
Funktionale Programmierung 202
Funktionen 175
als Parameter
als Rückgabewert 205
Funktionstypen 202
Gültigkeitsebenen 179
globale Funktionen 189
Namen 179
optionale
Parameter 182
Rückgabewert 175
Standardfunktionen 189
verschachtelte Funktionen 180
verzögert ausführen 582
Funktionsabschluss 206

# G

Ganze Zahlen 91
Garbage Collector 127
Gatekeeper 743
Generalisierung 268
Generated Interface 303
Generator-Typ 299
GeneratorType-Protokoll 299
Generics 270
Array-Beispiel 143
Extensions 293
Protokolle
Type Constraints 273
Gestures
Long Press 491, 493
Tap 548
get-Schlüsselwort 88, 235, 255

Git	. 63
Globale Funktionen	189
Gomoku	576
GPS-Funktionen	435
Grafikkontext	450
Grafikprogrammierung	
Drag&Drop (UIBezierPath)	709
Hockey-Spieler zeichnen	676
in einer MapView (MK-Methoden)	444
Kompasssteuerelement	
(CG-Funktionen)	449
Kreise zeichnen (UIBezierPath)	667
Richtungsanzeige (UIBezierPath)	513
groupingSeparator-Eigenschaft	545
Grundrechenarten	. 71
guard-Schlüsselwort 156,	315
Gültigkeitsebenen	179

# Н

Half-Open-Range-Operator	
HalfClosedInterval-Datentyp	
hAny-Einstellung	
Hashable-Protokoll	144, 285
hasPrefix-Methode	100
hasSuffix-Methode	100, 742
Haversine-Formel	
Header-Code erzeugen	
height-Eigenschaft	666
heightOfRow-Parameter	700
Hello World	
Animationen	569
iOS-App	
iOS-App mit Popup	461
MapView/GPS	435
OS-X-Programm	617
Playground	
Script	43
Terminal App	
Hexadezimale Zahlen	92
Hintergrund-App	440
HTTPS-Probleme mit NSURL	536

IBAction-Attribut 322,	336
IBDesignable-Attribut 322,	458
IBInspectable-Attribut 322,	458
IBOutlet-Attribut	336
Icon	717
Арр	556
iTunes Connect	563
Resizer	717
icon-Eigenschaft (NSAlert)	661
IconSize-Struktur (Icon-Resizer-Beispiel)	723

if-available-Test153if-verzweigungen153 $let$ (Optionals)120, 154, 715image-Eigenschaft702, 711Image-View-Steuerelement481, 531imageComponentsProvider-Eigenschaft711Images.xcasset-Datei432UIImage-Objekt erzeugen480Implicitly Unwrapped Optionals119imort-Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indirect-Schlüsselwort228infix84Info, plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zrückgeben245Overloading243Redundarz vermeiden265Inkrement-Operator73insertNewsAtIndexPath-Methode495insertRowsAtIndexPath-Methode495insertRowsAtIndexPath-Methode674instantiateController WithIdentifier- Methode660instantiateViewControllerWithIdentifier- Methode265Int-Init-Funktion113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort <th>id-Datentyp (Objective C)</th> <th>288</th>	id-Datentyp (Objective C)	288
if-Verzweigungen153 $let$ (Optionals)120, 154, 715image-Eigenschaft702, 711Image-View-Steuerelement481, 531imageComponentsProvider-Eigenschaft711Images.xcasset-Datei432 $UIImage-Objekt$ erzeugen480Implicitly Unwrapped Optionals119import-Anweisung320in-Schlüsselwort (Closures)207indexPathForCell-Methode198indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insertNewsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode660InstantiateControllerWithIdentifier- Methode265Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336interval-Datentypen79Intriliteselwort223Internal-Schlüsselwort223interval-Datentypen79Intriliteselwort332issenutator332issenutator332issenutator33	if-available-Test	158
let (Optionals)120, 154, 715image-Eigenschaft702, 711Image-View-Steuerelement481, 531imageComponentsProvider-Eigenschaft711Image-Sxcasset-Datei432UIImage-Objekt erzeugen480Implicitly Unwrapped Optionals119import-Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden265Inkrement-Operator763inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode467InstantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World328simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	if-Verzweigungen	153
image-Eigenschaft 702, 711 Image-View-Steuerelement 481, 531 imageComponentsProvider-Eigenschaft 711 Images.xcasset-Datei 432 UIImage-Objekt erzeugen 480 Implicitly Unwrapped Optionals 119 import-Anweisung 320 in-Schlüsselwort (Closures) 207 indexOf-Methode 198 indexPathForCell-Methode 486, 487, 518 indexPathForRow-Methode 200, 241 Designated versus Convenience 244 Fehler auslösen (throw) 311 nil zurückgeben 245 Overloading 243 Redundarz vermeiden 2667 UIViewController 362 Vererbung 265 Inkrement-Operator 73 inout-Schlüsselwort 184 insert-Methode 137, 145, 146 insertRowsAtIndexPath-Methode 495 insertText-Methode 137, 145, 146 instantiateControllerWithIdentifier- Methode 650 instantiateViewControllerWithIdentifier- Methode 467 Instanzmethoden 246 Int-Datentyp 91 Int-Init-Funktion 113 integerValue-Eigenschaft 113 Interface Builder 336 internal-Schlüsselwort 223 Internal-Schlüsselwort 324 Interval-Datentypen 79 Intrinsic Size 719 iOS Grundlagen 353 Hello World 328 Simulator 77, 125, 270 isEmpty-Eigenschaft 135, 144	<i>let (Ontionals)</i> 120, 154.	715
Image-View-Steuerelement481, 531ImageComponentsProvider-Eigenschaft711Images.xcasset-Datei432 $UIImage-Objekt$ erzeugen480Implicitly Unwrapped Optionals119import-Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495instantiateControllerWithIdentifier- Methode674instantiateViewControllerWithIdentifier- Methode246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	image-Eigenschaft 702.	711
imageComponentsProvider-Eigenschaft 711 Images.xcasset-Datei 432 UIImage-Objekt erzeugen 480 Implicitly Unwrapped Optionals 119 import-Anweisung 320 in-Schlüsselwort (Closures) 207 indexOf-Methode 198 indexPathForCell-Methode 486, 487, 518 indexPathForRow-Methode 487 indirect-Schlüsselwort 228 infix 84 Info.plist 557 Init-Funktion 220, 241 Designated versus Convenience 244 Fehler auslösen (throw) 311 nil zurückgeben 245 $Overloading 243$ $Redundanz$ vermeiden 267 UIViewController 362 Vererbung 265Inkrement-Operator 73 inout-Schlüsselwort 184 insert-Methode 137, 145, 146 insertRowsAtIndexPath-Methode 495 insertText-Methode 674 instantiateControllerWithIdentifier- Methode 650 instantiateViewControllerWithIdentifier- Methode 467 Int-Init-Funktion 113 integerValue-Eigenschaft 113 Interface 274 Interface 800 Internal-Schlüsselwort 223 Internal-Schlüsselwort 223 Internal-Schlüsselwort 223 Internal-Schlüsselwort 223 Interface 774 Interface 774 Interface 774 Interface 774 Interface 775 Interface 775 Interval-Datentypen 79 Intrinsic Size 719 IOS <i>Grundlagen 353 Hello World 324</i> <i>Simulator 77</i> , 125, 270 isEmpty-Eigenschaft 135, 144	Image-View-Steuerelement 481.	531
ImagecomponentInternetInternetImages.xcasset-Datei432 $UIImage-Objekt$ erzeugen480Implicitly Unwrapped Optionals119import-Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode487, 518indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Overloading243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateController/WithIdentifier-MethodeMethode650InstantiateViewControllerWithIdentifier-MethodeMethode361Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-ControllerWithIdentifier-Methode365Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Inte	imageComponentsProvider-Figenschaft	711
Imageories173 $UIImage-Objekt$ erzeugen480Implicitly Unwrapped Optionals119import-Anweisung320in-Schlüsselwort (Closures)207indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateController/WithIdentifier-MethodeMethode650InstantiateViewControllerWithIdentifier-MethodeMethode265Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface274Interface274Interface274Interface274Interface774Interface774Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World322Simulator777, 125, 270isEmpty-Eigenschaft135, 144	Images vesset-Datei	432
Implicitly Unwrapped Optionals119Import-Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Overloading243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertText-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650Methode650instantiateViewControllerWithIdentifier- Methode91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort	IIIImaae-Ohiebt erzeugen	480
Impert Anweisung113import Anweisung320in-Schlüsselwort (Closures)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Overloading243Redundanz vermeiden265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertText-Methode674instantiateControllerWithIdentifier- Methode650Methode650instantiateViewControllerWithIdentifier- Methode91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223<	Implicitly Unwrapped Optionals	119
$\begin{tabular}{lllllllllllllllllllllllllllllllllll$	import-Anweisung	320
In Schlusselwort (closules)207indexOf-Methode198indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650Int-Init-Funktion113integerValue-Eigenschaft113interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Interface774Interface774Interface774Interface719iOSGrundlagenGrundlagen353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	in-Schlüsselwort (Closures)	207
IndexFor Nicturate195indexPathForCell-Methode486, 487, 518indexPathForRow-Methode487indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertText-Methode674instantiateControllerWithIdentifier- Methode650Int-Init-Funktion113integerValue-Eigenschaft113interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort23instantiateViewControllerWithIdentifier- Methode361Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort239intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagen353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft<	indexOf-Methode	198
Index PathForRow-Methode480, 480, 918indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertText-Methode674instantiateControllerWithIdentifier- Methode650Int-Init-Funktion113interface274Interface274Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort236interface274Interface749interface749interstord328simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	indexDathEorCell-Methode 486,487	510
IndexPathFolkow-Methode437indirect-Schlüsselwort228infix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650Intranzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Datentypen <td>index Path For Pow Mothodo</td> <td>107</td>	index Path For Pow Mothodo	107
Indirect-schlusselwort 228 infix	indirect Schlüsselwort	407
Initix84Info.plist557Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650Intranzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Interface774Interface719ioSGrundlagenGrundlagen332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	indirect-schlusserwort	220
Initopiist537Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650Intranzmethoden246Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen332, 439is-Operator77, 125, 270isEmpty-Eigenschaft1135, 144	IIIIX	84
Init-Funktion220, 241Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650InstantiateOutrollerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113Interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Interval-Datentypen79Intrinsic Size719iOS332, 439simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Info.plist	557
Designated versus Convenience244Fehler auslösen (throw)311nil zurückgeben245Overloading243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertText-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode467Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft1135, 144	Init-Funktion	241
Fehler auslosen (throw)311nil zurückgeben245Overloading243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode660instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Designated versus Convenience	244
nil zuruckgeben245Overloading243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650InstantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113intergace Builder336internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Internal-Schlüsselwort223Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World328Simulator732, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Fehler auslosen (throw)	311
Overloading243Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	nil zuruckgeben	245
Redundanz vermeiden267UIViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674instantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Overloading	243
UlViewController362Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674insertText-Methode674instantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Internal-Schlüsselwort223Internal-Schlüsselwort223Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Redundanz vermeiden	267
Vererbung265Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674insertXxx-Methoden674instantiateControllerWithIdentifier- Methode650instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	UIViewController	362
Inkrement-Operator73inout-Schlüsselwort184insert-Methode137, 145, 146insertRowsAtIndexPath-Methode495insertText-Methode674insertText-Methode674instantiateControllerWithIdentifier- Methode660instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Vererbung	265
inout-Schlüsselwort	Inkrement-Operator	. 73
insert-Methode	inout-Schlüsselwort	184
insertRowsAtIndexPath-Methode	insert-Methode 137, 145,	146
insertText-Methode	insertRowsAtIndexPath-Methode	495
insertXxx-Methoden 674 instantiateControllerWithIdentifier- Methode 650 instantiateViewControllerWithIdentifier- Methode 467 Instanzmethoden 246 Int-Datentyp 91 Int-Init-Funktion 113 integerValue-Eigenschaft 113 Interface 274 Interface Builder 336 internal-Schlüsselwort 223 Internationalization (i18n) 398 intersect-Methode 146, 713 Interval-Datentypen 79 Intrinsic Size 719 iOS Grundlagen 353 Hello World 328 Simulator 332, 439 is-Operator 77, 125, 270 isEmpty-Eigenschaft 135, 144	insertText-Methode	674
instantiateControllerWithIdentifier- Methode	insertXxx-Methoden	674
Methode650instantiateViewControllerWithIdentifier- Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	instantiateControllerWithIdentifier-	
instantiateViewControllerWithIdentifier- Methode	Methode	650
Methode467Instanzmethoden246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	instantiateViewControllerWithIdentifier-	
Instanzmethoden246Int-Datentyp91Int-Init-Funktion113integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenSimulator328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Methode	467
Int-Datentyp       91         Int-Init-Funktion       113         integerValue-Eigenschaft       113         Interface       274         Interface Builder       336         internal-Schlüsselwort       223         Internationalization (i18n)       398         intersect-Methode       146, 713         Interval-Datentypen       79         Intrinsic Size       719         iOS       Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	Instanzmethoden	246
Int-Init-Funktion       113         integerValue-Eigenschaft       113         Interface       274         Interface Builder       336         internal-Schlüsselwort       223         Internationalization (i18n)       398         intersect-Methode       146, 713         Interval-Datentypen       79         Intrinsic Size       719         iOS       Grundlagen         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	Int-Datentyp	91
integerValue-Eigenschaft113Interface274Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Int-Init-Funktion	113
Interface       274         Interface Builder       336         internal-Schlüsselwort       223         Internationalization (i18n)       398         intersect-Methode       146, 713         Interval-Datentypen       79         Intrinsic Size       719         iOS       Grundlagen       353         Hello World       328       328         Simulator       332, 439       352, 439         is-Operator       77, 125, 270       isEmpty-Eigenschaft	integerValue-Eigenschaft	113
Interface Builder336internal-Schlüsselwort223Internationalization (i18n)398intersect-Methode146, 713Interval-Datentypen79Intrinsic Size719iOSGrundlagenGrundlagen353Hello World328Simulator332, 439is-Operator77, 125, 270isEmpty-Eigenschaft135, 144	Interface	274
internal-Schlüsselwort 223 Internationalization (i18n)	Interface Builder	336
Internationalization (i18n)       398         intersect-Methode       146, 713         Interval-Datentypen       79         Intrinsic Size       719         iOS       353         Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	internal-Schlüsselwort	223
intersect-Methode       146, 713         Interval-Datentypen       79         Intrinsic Size       719         iOS       353         Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	Internationalization (i18n)	398
Interval-Datentypen       79         Intrinsic Size       719         iOS       353         Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	intersect-Methode 146,	713
Intrinsic Size       719         iOS       Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	Interval-Datentypen	79
iOS <i>Grundlagen</i>	Intrinsic Size	719
Grundlagen       353         Hello World       328         Simulator       332, 439         is-Operator       77, 125, 270         isEmpty-Eigenschaft       135, 144	iOS	
Hello World         328           Simulator         332, 439           is-Operator         77, 125, 270           isEmpty-Eigenschaft         135, 144	Grundlagen	353
Simulator         332, 439           is-Operator         77, 125, 270           isEmpty-Eigenschaft         135, 144	Hello World	328
is-Operator	Simulator	439
isEmpty-Eigenschaft 135, 144	is-Operator 77, 125,	270
	isEmpty-Eigenschaft 135,	144

isViewLoaded-Method	2	431
iTunes Connect		558

#### J

Ja-Nein-Dialog	470
join-Funktion	193
join-Methode	105

#### К

Key/Value-Paare 14	43
keyCode-Eigenschaft 6'	74
keyDown-Methode 6'	74
keys-Eigenschaft 14	44
keyUp-Methode 6'	74
Klassen 22	18
verschachteln 22	24
Kommazahlen	92
Kommentare	46
MARK 70	29
Playground	35
Kompass 435, 44	47
Kalibrierung 44	48
Konstanten	87
Eigenschaften23	30
Eulersche Zahl 19	<del>9</del> 5
Pi 19	95
Kontextmenü 68	85
Koordinatensystem 66	65
Kreis zeichnen 60	59
Kreisteilungszahl 19	<del>9</del> 5

#### Ŀ

Label (OS X)	623
Label (break/continue)	166
Lambda-Ausdruck	206
last-Eigenschaft 13	36, 197
last-Methode	108
LaunchScreen.xib-Datei	555
layer-Eigenschaft	598
Layoutregeln	368
View-Größe fixieren	648
Lazy Properties	231
lazy-Funktion	194
LazyBidirectionalCollection-Datentyp	144
Lebenszyklus	
Арр	365
View-Controller	361
Left-Shift (bitweises Rechnen)	73
let-Schlüsselwort	87
in switch-Konstruktionen	161
mit if 15	64, 715
mit switch-case	227

mit while	164
lineWidth-Eigenschaft	669
Listen-Steuerelement (iOS)	472
veränderliche Listen	487
Listenfeld	550
LiteralConvertible-Protokolle	289
locale-Eigenschaft 112,	115
Localization (l10n)	398
Localization native development region	399
$localized Case Insensitive Compare \$	100
localizedDescription-Eigenschaft	319
Location Manager	442
Kompass	447
teilen	509
locationInView-Methode	601
locationInWindow-Eigenschaft	666
locationManager-Methode 443,	447
locationManagerShouldDisplayHeading-	
Calibration-Methode	448
lockFocus-Methode	725
Logische Operatoren	78
Lottozahlen	145
Benchmarks	166
OS-X-Beispiel	620
lowercaseString-Eigenschaft	106

#### Μ

mainBundle-Methode	395
makeKeyAndOrderFront-Eigenschaft	687
Mangled Name	127
map	
Map Kit (MK)	435
map-Methode	141, 199
mapOverlay-Methode	444
mapView-Methode	446
MapView-Steuerelement	435
MARK-Kommentar	709
Markdown-Kommentare	35, 46
Mathematische Funktionen	195
Maus	664
max-Eigenschaft	91
max-Funktion	194
maxElement-Funktion	194
maximumFractionDigits-	
Eigenschaft	112, 545
Mehrblättrige Dialoge	651
Mehrdimensionale Arrays	142
erzeugen	579
Mehrfachvererbung	260
Menüs	678
deaktivieren	686
Kontextmenüs	685

menu-Eigenschaft	
NSApplication	684
NSView	685
Menubar-App	686
Message-Box	660
messageText-Eigenschaft	661
Metatypen	301
Methoden	246
Aufruf mit Optional Chaining	121
Mutating Methods	248
optionale	280
Signatur	253
statische Methoden	250
Typmethoden	250
min-Eigenschaft	. 91
min-Funktion	194
minElement-Funktion	194
minimumFractionDigits-	
Eigenschaft 112,	545
minimumPressDuration-Eigenschaft	491
Mirror-Datentyp	126
MKMapView-Steuerelement	435
MKMapViewDelegate-Protokoll	441
MKPolyline-Klasse	444
MKPolylineRenderer	446
modalPresentationStyle-Eigenschaft	468
modifierFlags-Eigenschaft 667,	674
Modifizier	224
Module 222,	320
Modulo-Operator	. 71
mouseDown-Methode 665, 671,	735
mouseDragged-Methode 665, 710,	735
mouseEntered-Methode	665
mouseExited-Methode	665
mouseUp-Methode	665
moveRowAtIndexPath-Parameter	497
moveXxx-Methoden	674
MutableCollectionType-Protokoll	191
MutableSliceable-Protokoll	191
mutating-Schlüsselwort	248

# Ν

Nachrichten anzeigen	660
Named Types	124
Navigation-Controller	422
Detailansicht einer Liste	484
mit Tab-Bar-Controller verbinden	429
Nested Functions	180
next-Methode	299
NextStep	34
NIB-Datei	362
Nil	81
Nil-Coalescing-Operator	81

nil-Schlüsselwort		117
noescape-Attribut	214,	322
noreturn-Attribut		322
Notification Center		510
npm-Kommando		745
NSAlert-Klasse		661
NSApplication-Klasse	629,	730
NSAppTransportSecurity-Eintrag	·····	536
NSBezierPath-Klasse	669.	709
NSBundle-Klasse	,	395
NSButton-Klasse		739
NSCalendar-Klasse		115
NSCalender Unit-Struktur		147
NSCocoaFrrorDomain		310
NSCoder-Klasse		507
NSCoding-Protokoll	363	507
NSColorDanol Klasso	505,	662
NGColorMall Stowardow ant		603
NSCOlor well-Steuerelement		115
NSDate-Klasse		115
NSDateFormatter-Klasse		115
NSDirectory-Klasse		656
NSDraggingDestination-Protokoll		705
NSDraggingImageComponent-Klasse		711
NSDraggingInfo-Klasse		715
NSDraggingItem-Klasse	706,	710
NSDraggingSource-Protokoll	710,	735
NSError-Klasse		318
NSError-Klasse		318 314
NSError-Klasse Casting von ErrorType NSEvent-Klasse		318 314
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus		318 314 666
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur		318 314 666 674
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur		318 314 666 674 667
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse		318 314 666 674 667 320
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse		318 314 666 674 667 320 741
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante		318 314 666 674 667 320 741 715
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante NSFont-Klasse		318 314 666 674 667 320 741 715 658
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante NSFont-Klasse NSFont-Klasse	394, 707, 658,	318 314 666 674 667 320 741 715 658 662
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse	394, 707, 658,	318 314 666 674 667 320 741 715 658 662 662
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSFontPanel-Klasse	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFoltManager-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImage-Klasse NSImage-Klasse	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724 708
NSError-Klasse <i>Casting von ErrorType</i> NSEvent-Klasse <i>Maus</i> <i>Tastatur</i> NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFoltManager-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSFontPanel-Klasse NSImage-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Steuerelement	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 652 652 724 708 702
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFoltManager-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImage-Klasse NSImageView-Klasse NSImageView-Steuerelement NSImageView-Steuerelement NSIndexPath-Klasse	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFoltManager-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Steuerelement NSImageView-Steuerelement NSIndexPath-Klasse NSKevedArchiver-Klasse	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 507
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImage-Klasse NSImageView-Klasse NSImageView-Steuerelement NSImageView-Steuerelement NSIndexPath-Klasse NSKeyedArchiver-Klasse NSLavoutConstraint-Klasse	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 507 385
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImage-Klasse NSImageView-Klasse NSImageView-Steuerelement NSImageView-Steuerelement NSIndexPath-Klasse NSKeyedArchiver-Klasse NSLayoutConstraint-Klasse NSLANA NSLAN	394, 707, 658, 702,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 507 385 115
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Steuerelement NSImageView-Steuerelement NSIndexPath-Klasse NSKeyedArchiver-Klasse NSLayoutConstraint-Klasse NSLocale-Klasse NSLocale-Klasse NSLocalizedString-Klasse	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 724 708 702 487 507 385 115 405
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFilenamesPboardType-Konstante NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSKeyedArchiver-Klasse NSLocale-Klasse NSLocalizedString-Klasse NSMenu-Klasse	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 724 708 702 487 507 385 115 405 680
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImage-View-Klasse NSImageView-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSMenu-Klasse NSMenu-Klasse NSMenu-Klasse	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 724 708 702 487 507 385 115 405 680 680
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSMenu-Klasse NSMenu-Klasse NSMenuValidation-Protocoll	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 507 385 115 405 680 680 680 680
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSLayoutConstraint-Klasse NSLayoutConstraint-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSMenu-Klasse NSMenu-Klasse NSMenuValidation-Protokoll NSMutableDictionary-Klasse	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 724 708 702 487 507 385 115 405 680 680 685 541
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFont-Klasse NSFontPanel-Klasse NSImage-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSMenu-Klasse NSMenu-Klasse NSMenuValidation-Protokoll NSMutableDictionary-Klasse NSNotificationCentar-Klasse NSNotificationCentar-Klasse NSNotificationCentar-Klasse	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 702 487 507 385 115 680 680 680 685 541
NSError-Klasse Casting von ErrorType NSEvent-Klasse Maus Tastatur NSEventModifierFlags-Struktur NSException-Klasse NSFileManager-Klasse NSFont-Klasse NSFontManager-Klasse NSFontPanel-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Klasse NSImageView-Steuerelement NSIndexPath-Klasse NSLayoutConstraint-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocale-Klasse NSLocalizedString-Klasse NSMenu-Kla	394, 707, 658, 702, 1112,	318 314 666 674 667 320 741 715 658 662 662 724 708 702 487 702 487 702 487 702 487 507 385 115 680 680 680 685 541 510

NSObject-Klasse 260	, 506,	699
description-Eigenschaft		284
NSObject-Protokoll		509
NSObjectProtocol-Protokoll	260,	509
NSOpenPanel-Klasse		661
Datei auswählen		738
Verzeichnisauswahl		730
NSPasteBoard-Klasse		715
NSPipe-Klasse		323
NSPoint-Klasse		711
NSPoint-Struktur		666
NSRect-Klasse		711
NSRect-Struktur		666
NSRectFill-Methode		669
NSRegularExpression-Klasse		102
NSResponder-Klasse	. 673,	681
NSSavePanel-Klasse		661
NSSearchPathForDirectoriesInDomai	ns-	
Funktion	393,	541
NSSearchPathForDirectoriesInDomai	ns-	
Klasse		507
NSSize-Klasse		711
NSSize-Struktur		666
NSSortDescriptor-Klasse		703
NSSplitter-Klasse		655
NSSplitView-Steuerelement		719
Delegation		732
NSSplitViewController-Klasse		720
NSSplitViewDelegate-Protokoll		732
NSStatusBar-Klasse		687
NSString-Datentyp		94
NSStringPboardType-Konstante		707
NSTableView-Steuerelement		693
NSTableViewDataSource-Protokoll	696,	732
NSTableViewDelegate-Methode		734
NSTableViewDelegate-Protokoll		696
NSTabViewController-Steuerelement		651
NSTask-Klasse		323
NSTemporaryDirectory-Funktion		741
NSTextField-Steuerelement	623,	702
NSURL-Klasse		536
NSUserDefaults-Klasse 391	, 541,	655
Beispiel		490
Defaultwerte		652
NSView-Klasse	. 619,	701
NSViewController-Klasse	619,	639
NSWindow-Klasse	619,	623
NSWindowController-Klasse	619,	635
NSWindowDelegate-Protokoll	634,	638
NSWorkspace-Klasse		730
NSXMLParser-Klasse		538
NSxxx-Klassennamen		34
null-Schlüsselwort		117

numberOfComponentsInPickerView-	
Methode	552
numberOfRowsInComponent-Parameter	552
numberOfRowsInSection-Parameter	476
numberOfRowsInTableView-Methode	696
numberOfSectionsInTableView-Methode	476

# <u>0</u>

objc-Attribut	280,	323
objectForKey-Methode		542
objectValueForTableColumn-Methode		696
Objektorientierte Programmierung		217
Observer (Eigenschaften)		232
Observer (Notification Manager)		511
Oktale Zahlen		. 92
on-Eigenschaft		573
Operatoren		. 69
Assoziativität		. 82
Priorität		. 82
selbst definieren		. 83
selbst definieren, Beispiel		112
Optional Chaining	. 82,	120
optional-Schlüsselwort		280
Optionale Funktionen/Methoden		280
Optionale Parameter		187
Optionale Protokollregeln		280
Optionals	. 81,	117
als Rückgabewert von Funktionen		176
if-let-Kombination	154,	715
Init-Funktion		245
OptionSetTyp-Protokoll		146
Beispiel		713
orderFrontFontPanel-Methode		662
orderOut-Methode		689
origin-Eigenschaft		666
Outlets		355
Collections		339
OS X		626
umbenennen		340
Overloading		
Funktionen		179
Init-Funktion		243
override-Schlüsselwort		261
Computed Properties		262
Property Observers		262
View-Controller		364

#### P

Palindromtest	107
Parameter	182
autoclosures	209
benannte 174, 18	5, 253
benannte, in Init-Funktionen	242

benannte, in Methoden	251
benannte, in Protokollen	275
Inout-Parameter	184
noescape	214
optionale Parameter	187
variable Anzahl	189
pathForResource-Methode	395
Pattern-Zeichen	. 70
PDFs in Xcasset-Dateien	432
performDragOperation- Methode	737
performSegueWithIdentifier- Methode 419, 421,	650
Navigation-Controller	425
Pi-Konstante	195
Picker-View-Steuerelement 531.	550
Rollover	552
nickerView-Methode	552
Pin-Button	369
Divel versus Dunkt	368
Placeholder (Vcode Einstellung)	710
Placenoider (Acode-Enistending)	/19
Playground	. 30
PNG-Datel erzeugen	121
Polymorphie	268
Popover-Segue	461
popToRootViewControllerAnimated- Methode	425
popToViewController-Methode	425
Popups	461
5-Gewinnt-App	609
Größe einstellen	464
per Code anzeigen	467
Richtung festlegen	466
popViewControllerAnimated-	526
Methode 425,	520
postiix	. 84
postivotificationName-Methode	511
Potenzieren	. 72
pow-Funktion	. 72
precedence	. 84
predecessor-Methode	108
preferredContentSize-Eigenschaft	464
prefix	. 84
prepareForSegue	646
prepareForSegue-Methode	417
Detailansicht bei Table-View	486
prepareForSeque-Methode	
Popups auf dem iPhone	463
preprareForDragOperation-Methode	705
Presentation-Controller	412
presentedViewController-Eigenschaft	367
Presenting Segues	645
presentingViewController-Eigenschaft	465
presentViewController-Methode 467,	472

print-Funktion	194
CustomStringConvertible-Protokoll	283
Printable-Protokoll	283
printf-Syntax	111
Priorität von Operatoren	. 82
private(set) für Read-only-Eigenschaften	236
private-Schlüsselwort	223
Programm signieren/weitergeben	742
Programmende	
iOS	365
OS X 629, 634,	638
prompt-Eigenschaft	423
Properties	230
Computed Properties 235,	262
Extensions	295
Lazy Properties	231
Property Observers 232, 262, 453, 491,	544
Property Observers (Beispiel)	601
Read-Only Computed Property	236
Static Properties	234
Type Properties	234
Property Lists	
User-Defaults	490
propertyListForType-Methode	715
protocol-Schlüsselwort	276
Protocol-Schlüsselwort	301
Protokolle	274
erweitern	296
Extensions	292
für generische Typen	281
nur für Klassen	277
optionale Regeln	280
Protocol Composition	278
Standardprotokolle	283
Vererbung	277
Provisioning Profile	566
public-Schlüsselwort	223
Punkt versus Pixel	368

#### R

Rückgabewerte (Funktionen)	175
radix-Parameter	
raise-Methode	320
Rand eines Steuerelements	
Random Numbers	
Range-Datentyp	
Range-Operatoren	
rangeOfCharacterFromSet-Methode	107
rangeOfString-Methode	100
RawOptionSetType-Protokoll	667
rawValue-Eigenschaft	90, 226
Read-Only-Eigenschaft	236
readDataToEndOfFile-Methode	323

readLine-Funktion	195
readObjectsForClasses-Methode	715
Rechenoperatoren	. 71
Redraw-Einstellung (contentMode)	455
reduce-Methode 141, 199,	200
Reference Counting	127
Referenztypen	122
Reflection	126
Regeln (Auto Layout)	368
registerDefaults-Methode	656
registerForDraggedTypes-	
Methode 705, 707,	729
Reguläre Ausdrücke	102
Rekursion	181
Enumerationen	228
reloadData-Methode 487,	703
reloadRowsAtIndexPaths-Methode	496
remove-Methode 145.	146
removeAtIndex-Methode	137
removeFromSubview-Methode	600
removeItemAtPath-Methode	741
removel ast-Methode	137
removeObserver-Methode	512
removeRange-Methode	137
Reniu	576
repeat-while-Schleife	165
PEDI Modus	105
replaceRange-Methode	127
representedObject Figenschaft	625
represented Object-Eigenschaft	025
Methode 437	112
requestWhenInLiseAuthorization-	772
Methode	437
required-Schlüsselwort	266
reserveCapacity-Methode	139
resignFirstResponder-Methode	673
Resistance Priority	389
Resolve-Lavout-Issues-Button	369
Responder	505
Menüauswahl	681
Responderbette	678
Tastaturerejanisse	673
Ressourcen	39/
reStructuredText-Kommentare	35
Restwert-Operator	71
Retroactive Modeling	201
return-Schlüsselwort	173
Reverse Polish Notation	210
reverse-Funktion	107
reverse-Methode 120	201
Richtung zwischen zwei Koordinaten	201
nunkten	523
Right-Shift (bitweises Rechnen)	73
rightMouseDown-Methode	665

rightMouseDragged-Methode	665
rightMouseUp-Methode	665
Rollover (Picker-View)	552
Root-View-Controller 3	861, 367
rootViewController-Eigenschaft 3	861, 367
round-Funktion	93
rowHeightForComponent-Parameter	554
RPN-Rechner	210
runModal-Methode	661

# <u>S</u>

Schalter (UISwitch-Steuerelement)	572
Schatzsuche	501
Schlüssel-Wert-Paare	143
Schlüsselwörter als Variablennamen	. 86
Schleifen	162
abbrechen (break)	165
Schnittstelle	274
Schrift auswählen	660
Schriftattribute ändern	658
scrollToRowAtIndexPath-Methode 487,	495
scrollXxx-Methoden	674
Segues	411
Datenübergabe	646
Datenübertragung	415
OS X	645
per Code initijeren	419
Popover	461
Unwind 413	417
von Menüeinträgen	682
selectedImage-Eigenschaft	428
selectedIndex-Figenschaft	431
selectedRow-Figenschaft	704
selectedViewController-Figenschaft	431
selectRow-Figenschaft	550
selectXvv-Methoden	674
Selektor 38/ 492	687
Svntav 254	384
self/Self-Schlüsselwort	222
hei Fnumerationen	577
in Closures	209
in Mutatina Methods	205
Klassentyn	229
Drotokolle 277	205
SequenceType-Protokoll 163	205
Sequencerype-Flotokoli 105,	101
Set Datentum	1/1
Ontion Sate	145
uniqueSet Methode	200
ant Cohligeolwart	300
set-schlusselwort	400
setDeteCause Methode	200
setDataSouce-Methode	700
setDataSouce-Methode (Table-View)	696

setDelegate-Methode 700
Table-View
setDelegate-Methode (Table-View) 696
setEditing-Methode 488, 493
setFill-Methode 513
setNeedsDisplay-Methode 453, 596, 709
setNeedsDisplayInRect-Methode
setObject-Methode 541
setPosition-Methode
setRegion-Methode 444
setSelectedFont-Methode
setStroke-Methode 513
setTarget-Methode (NSColorPanel)
setTitle-Methode
sharedApplication-Methode 367, 629, 730
sharedColorPanel-Methode
sharedFontManager-Methode
sharedWorkspace-Methode 730
ShiftKeyMask-Eigenschaft
Short-Circuit Evaluation
shouldChangeCharactersInRange-
Parameter 547
showWindow-Methode 637, 650
Shuffle-Algorithms 141
Signatur
Signatur von Methoden 253
Simulator (iOS) 332
GPS-Funktionen 439
Singleton-Muster 234
Size Classes
size-Eigenschaft
sizeThatFits-Methode 465
SKU (Stock Keeping Unit) 562
Slices (Arrays) 136
sort-Methode 138, 201
für NSTableView 703
sortDescriptors-Eigenschaft 703
sortDescriptorsDidChange-Parameter 703
Sortierordnung für Zeichenketten 100
sortInPlace-Methode
Source Control
sourceOperationMaskForDragging-
Context-Parameter
sourceRect-Eigenschaft
sourceView-Eigenschaft
spacing-Eigenschaft
speti-kommando
Speicnerverwaltung 127
splice-Methode
Zaichanhattan in Zailen zarlagan
Split View 710
Delegation 722
Split-View-Controller 720
Juit-view-Controller

Splitter-Steuerelement	655
splitView-Methode	732
srand48-Funktion 94,	582
SSLHandshake failed (NSURL-Klasse)	536
Stack-Button	369
Stack-Speicher	182
Stack-View	386
Animationen	572
Standarddialoge	660
iOS	470
Standardeigenschaften	196
Standardfunktionen	189
Standardmethoden	196
Standardprotokolle	283
standardUserDefaults-Methode	656
Startansicht	555
startsWith	198
startUpdatingHeading-Methode	447
startUpdatingLocation-Methode	442
state-Eigenschaft 493 684	739
Static Properties	234
static-Schlüsselwort	250
Statische Methoden	250
Statushar	687
statusItemWithLength-Methode	687
stdlib	007
aetDemanaledTypeName-Euroption	127
getTypename-Funktion	127
Stauaralamonto	127
sieuerereneme	573
eille und dusbienden	110
Seibsi programmeren	449
stored Properties	230
storyboard-Eigenschaft	650
Storyboards	650
Fenster per Code erzeugen	650
105	329
OS X	643
String Interpolation	97
String-Datentyp	. 94
String-Konstruktor	. 92
String.Index-Datentyp 100, 108,	192
stringByAppendingPathComponent- Methode	741
stringByReplacingOccurrencesOfString- Methode	702
stringByTrimmingCharactersInSet-	
Methode	106
stringFromNumber-Methode	112
StringLiteralConvertible-Protokoll	289
stringValue-Eigenschaft 627.	702
stroke-Methode 513.669.	709
strong-Schlüsselwort	129
struct-Schlüsselwort	220

Strukturen	218
verschachteln	224
styleMask-Eigenschaft	649
Subclassing	259
Subscripts	255
bequemer Zeichenkettenzugriff	109
Substrings lesen	108
subviews-Eigenschaft	593
successor-Methode	108
super-Schlüsselwort	263
swap-Funktion	195
Swift	
Compiler	44
Compiler Interpreter	44 42
Compiler Interpreter swiftdoc-Dateien	44 42 321
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien	44 42 321 321
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen	44 42 321 321 158
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration	44 42 321 321 158 227
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel	44 42 321 321 158 227 150
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel Syntaktischer Zucker	44 42 321 321 158 227 150 124
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel Syntaktischer Zucker Synthesized Headers	44 42 321 321 158 227 150 124 303
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel Syntaktischer Zucker Synthesized Headers Systemfunktionen aufrufen	44 42 321 158 227 150 124 303 323
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel Syntaktischer Zucker Synthesized Headers Systemfunktionen aufrufen systemStatusBar-Methode	44 42 321 321 158 227 150 124 303 323 687
Compiler Interpreter swiftdoc-Dateien swiftmodule-Dateien switch-Verzweigungen Enumeration Tupel Syntaktischer Zucker Synthesized Headers Systemfunktionen aufrufen systemStatusBar-Methode Szene (iOS)	44 42 321 158 227 150 124 303 323 687 329

# Ţ

Tab-Bar-Controller	426
Tab-Bar-Items	427
Tab-View-Controller	651
tabBarController-Eigenschaft	431
tabBarController-Methode	430
Table-View-Steuerelement	
iOS	472
<i>OS X</i>	693
tableFooterView-Eigenschaft	477
tableView-Methoden 476, 696, 700,	732
TableView-Steuerelement (iOS)	487
$table View Selection Did Change-Methode\$	704
Tag-Eigenschaft (Tab-Bar-Item)	428
Tap Gesture Recognizer	548
Tastatur	
ausblenden 420,	548
einblenden	498
OS X	672
Teilzeichenketten extrahieren	108
Temporäres Verzeichnis	741
terminate-Methode 629,	730
Ternärer Operator	80
Beispiel 573, 596,	606
Textdatei lesen/schreiben	393
textField-Methode	547
textFieldShouldReturn-Methode	420
Textured Button	720

throw/throws-Schlüsselwort	310
in Init-Funktionen	311
Tilde-Operator (binäre Inversion)	73
timeIntervalSinceDate-Methode	117
titleForRow-Parameter	552
TLSv1.2-SSL-Verschlüsselung für NSURL	536
Todo-Liste	489
Tool Tips	721
touchesBegan-Methode	601
touchesEnded-Methode	601
Trackpad	664
Trailing Closure	208
Trailing Closures	208
traitsOfFont-Methode	662
translatesAutoresizingMaskInto-	
Constraints-Eigenschaft	385
Transparenz	572
true	94
try-catch-Konstruktion	305
try-Schlüsselwort	
forced try	315
mit do-catch	306
ohne do-catch	315
Tupel	149
als Rückgabewert von Funktionen	175
in switch-Konstruktionen	160
switch-Auswertung	150
Type Annotation	86
Type Constraints	273
Type Properties	234
Type-Schlüsselwort	301
typealias-Schlüsselwort 125	, 281
AnyClass-Datentyp	289
Typen	123
Aliase	125
Metatypen	301
Typmethoden	250

# U

Uhrzeit	115
UIActionController-Klasse	526
UIAlertAction-Klasse	471
UIAlertController-Klasse	470
UIApplication-Eigenschaft	367
UIApplication-Klasse	366
UIApplicationDelegate-Protokoll	366
UIApplicationMain-Attribut 323,	365
UIBarButtonItem-Steuerelement	423
UIBezierPath-Klasse	513
UIButton-Steuerelement	384
UIColor-Klasse	363
erweitern	580

UIDeviceOrientationDidChange- Notification-Nachricht
UIFont-Klasse
UIGestureRecognizerDelegate-
Protokoll 491, 548
UIGraphicsGetCurrentContext-Methode 450
UIImage-Klasse
UIImageView-Steuerelement
Instanz per Code erzeugen 553
UIKit-Framework
UILabel-Klasse
UILongPressGestureRecognizer-
Klasse 491, 493
UINavigationController-Klasse 422
UInt-Datentyp 91
UIPickerView-Steuerelement 531, 550
Rollover 552
UIPickerViewDataSource-Protokoll 552
UIPickerViewDelegate-Protokoll 552
UIPopoverPresentationController-
Delegate-Protokoll 463
UIScreen-Klasse
UIStackView-Steuerelement
UIStoryboardSegue-Klasse 417
UISwitch-Steuerelement 572
UITabBarController-Klasse 426
UITabBarControllerDelegate-Protokoll 430
UITabBarItem-Klasse
UITableView-Steuerelement 472
Delete on Swipe 496
veränderliche Listen 487
UITableViewCell-Klasse
UITableViewController-Klasse 473
UITableViewDataSource-Protokoll
UITableViewDelegate-Protokoll 477
UITapGestureRecognizer-Klasse 548
UITextField-Steuerelement
FditinaChanaed-Freianis 549
ValueChanged-Freignis 549
UITextFieldDelegate-Protokoll 420
UITextView-Steuerelement
Umranduna 363
UITouch-Klasse 601
UIView-Klasse 359 449 513
Crößenänderung factstellen 601
Instanz per Code erzeugen 552
UlViewController-Klasse 359 361
Laboraryhlus 201
Lebenszyklus
Unvindume since Stationalements 202
Unärer Operator
unable to simultaneously actively actively and
(Feblermeldung) 205
unarchiveOhiectWithFile-Methoden 507
unarchiveObjectivitinine-Methouen

Unicode-Skalar		96
unicodeScalars-Eigenschaft		104
Beispiel		240
union-Methode		146
uniqueSet-Methode		300
unlockFocus-Methode		725
unowned-Schlüsselwort		129
unowned self in Closures	213,	571
unrecognized-selector-Fehler	340,	384
unregisterDraggedTypes-Methode		729
Unwind Segues	413,	417
Popups		469
Unwrapping-Operator	. 81,	119
Upcast		268
uppercaseString-Eigenschaft		106
URLForRessource-Methode		656
URLs-Eigenschaft		661
useGroupingSeparator-Eigenschaft		112
User-Defaults		541
Beispiel		490
iOS		391
<i>OS X</i>	652,	655
usesGroupingSeparator-Eigenschaft		545
utf16-Eigenschaft		105
utf8-Eigenschaft		105

#### V

validateMenuItem-Methode	685
values-Eigenschaft	144
var-Schlüsselwort	. 85
in switch-Konstruktionen	161
mit if	154
mit while	164
Variablen	. 85
Variablennamen	. 86
Variadics	189
Vererbung	259
bei Steuerelementen	449
Protokolle	277
Vergleichsoperatoren	74
Verschachtelte Funktionen	180
Versionsabhängiger Code	158
Versionsverwaltung	. 63
Verzeichnis	
auswählen 660,	730
erzeugen	741
löschen	741
temporäres	741
Verzweigungen	153
View	
Größe fixieren	648
schließen (OS X)	649
View-Controller 361,	618

OS X	39
Root-View-Controller 361, 36	67
view-Eigenschaft 384, 68	37
viewControllers-Eigenschaft 43	31
viewDidAppear-Methode 361, 64	19
Animationen 57	72
viewDidDisappear-Methode 361, 629, 73	30
viewDidLayoutSubviews-Methode	52
viewDidLoad-Methode	54
Animationen 57	72
OS X	25
viewForRow-Parameter	52
viewForTableColumn-Parameter 701, 73	32
viewWillAppear-Methode	24
viewWillDisappear- Methode	25
viewWillLayoutSubviews-Methode	52

#### W

wAny-Einstellung	381
weak-Schlüsselwort	129
Beispiel	36, 647
Werttypen	70, 122
where-Schlüsselwort	
if	155
Protokollerweiterungen	297
Regeln für generische Datentypen	273
switch/case	161
while-Schleife	164
let (Optionals)	164
width-Eigenschaft	666
Wildcard-Pattern-Zeichen	70
Willkommensbildschirm	555
willSet-Funktion 23	32, 262
Window-Controller	18, 635
window-Eigenschaft 367, 638, 64	49, 687
windowDidLoad-Methode	637
windowNibName-Parameter	637
windowWillClose-Methode	34, 638
writeToFile-Methode 318, 39	93, 541
Wuziqi	576

# X

x-Eigenschaft	666
Xcasset-Datei	432
UIImage-Objekt erzeugen	480
Xcode	30
erste Schritte	37
Hello iOS-World	328
mehrsprachige Apps	398
XIB-Dateien	52, 630
XLIFF-Dateien	402
XML-Bibliothek	535
XMLElement-Klasse	538
XMLIndexer-Klasse	538
XWXMLHash-Bibliothek	538

#### <u>Y</u>

y-Eigenschaft		666
---------------	--	-----

# <u>Z</u>

Zahlen	1
Zeichenketten	4
Copy-on-Write	6
Länge	8
sortieren	9
Substrings bequemer lesen 109	9
Substrings lesen 108	8
Teilzeichenketten extrahieren 10	8
vergleichen	9
zeilenweise zerlegen 32	3
Zeit 11	5
messen 11	7
zip-Funktion 192	2
Zip2Sequenz-Datentyp 192	2
Zufallszahlen 93	3
Zugriffsebenen 222	2
Zusammengesetzte Typen 124	4
Zuweisung 69	9
Zweidimensionale Arrays 579	9