

Der Inhalt (im Überblick)

	Einführung	xxi
1	Erste Schritte: <i>Ein Sprung ins kalte Wasser</i>	1
2	Basistypen und -variablen: <i>Eine Variable sein</i>	31
3	Funktionen: <i>Raus aus main</i>	59
4	Klassen und Objekte: <i>Etwas mehr Klasse</i>	91
5	Subklassen und Superklassen: <i>Vererbung</i>	121
6	Abstrakte Klassen und Interfaces: <i>Ernsthafter Polymorphismus</i>	155
7	Datenklassen: <i>Mit Daten umgehen</i>	191
8	Nullwerte und Ausnahmen: <i>Gesund und munter</i>	219
9	Collections: <i>Dinge organisieren</i>	251
10	Generische Programmierung: <i>Innen und außen unterscheiden</i>	289
11	Lambdas und Funktionen höherer Ordnung: <i>Code wie Daten behandeln</i>	325
12	Eingebaute Funktionen höherer Ordnung: <i>Dem Code Beine machen</i>	363
i	Koroutinen: <i>Code parallel ausführen</i>	397
ii	Testen: <i>Ziehen Sie Ihren Code zur Rechenschaft</i>	409
iii	Was übrig bleibt: <i>Die Top Ten der Themen, die wir nicht behandelt haben</i>	415
	Index	435

Inhalt (jetzt ausführlich)

Einführung

Ihr Gehirn und Kotlin. Sie versuchen, etwas zu *lernen*, und Ihr *Hirn* tut sein Bestes, damit das Gelernte nicht *hängen bleibt*. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z. B. für das Wissen darüber, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, wie man in Kotlin programmiert?

Für wen ist dieses Buch?	xxii
Wir wissen, was Sie gerade denken.	xxiii
Und wir wissen, was Ihr <i>Gehirn</i> gerade denkt.	xxiii
Metakognition: Nachdenken übers Denken	xxv
Das haben WIR getan:	xxvi
Lies mich	xxviii
Danksagungen	xxix
Das Team der Fachgutachter	xxx
Über den Übersetzer dieses Buchs	xxxix

Erste Schritte

1

Ein Sprung ins kalte Wasser

Kotlin schlägt Wellen.

Seit seiner ersten Veröffentlichung hat Kotlin Programmierer mit seiner *freundlichen Syntax, Knappheit, Flexibilität und Leistungsfähigkeit* beeindruckt. In diesem Buch zeigen wir Ihnen, wie Sie **Ihre eigenen Kotlin-Applikationen erstellen** können. Wir beginnen, indem wir ein einfaches Programm schreiben und laufen lassen. Unterwegs stellen wir Ihnen wesentliche Teile der Kotlin-Syntax vor, z. B. *Anweisungen, Schleifen und bedingungs-basierte Verzweigungen*. Ihre Reise hat gerade erst begonnen ...

Die Möglichkeit, auszuwählen, gegen welche Plattform Ihr Code kompiliert wird, bedeutet, dass Kotlin auf Servern, in der Cloud, in Browsern, auf Mobilgeräten und mehr funktioniert.



Willkommen in Kotlinville	2
Sie können Kotlin fast überall benutzen	3
Was wir in diesem Kapitel tun	4
IntelliJ IDEA (Community Edition) installieren	7
Eine einfache Applikation erstellen	8
Eine einfache Applikation erstellen (Fortsetzung)	9
Eine einfache Applikation erstellen (Fortsetzung)	10
Das erste Kotlin-Projekt ist erstellt	11
Fügen Sie dem Projekt eine Kotlin-Datei hinzu	12
Anatomie der main-Funktion	13
Bauen Sie die main-Funktion in App.kt ein	14
Probefahrt	15
Was können Sie in der main-Funktion sagen?	16
Schleifen, Schleifen, Schleifen ...	17
Ein Beispiel mit Schleifen	18
Bedingungs-gesteuerte Verzweigungen	19
Rückgabewerte für if	20
Aktualisieren Sie die main-Funktion	21
Die interaktive Kotlin-Shell benutzen	23
REPL versteht auch mehrzeilige Codeabschnitte	24
Vermischte Nachrichten	27
Ihr Kotlin-Werkzeugkasten	30

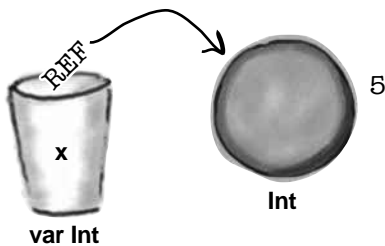
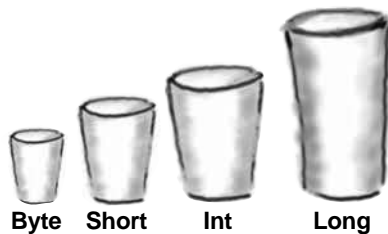
Basistypen und Variablen

Eine Variable sein

2

Es gibt eine Sache, von der jeder Code abhängt: Variablen.

In diesem Kapitel werfen wir einen Blick unter die Motorhaube und zeigen Ihnen, wie **Kotlin-Variablen tatsächlich funktionieren**. Sie werden Kotlins **Basisdatentypen** wie *Integer*, *Floats* und *boolesche Werte* kennenlernen. Sie werden sehen, wie Kotlins Compiler den **Typ einer Variablen anhand des übergebenen Werts feststellen** kann. Außerdem lernen Sie den Einsatz von **String-Templates** für die Erstellung komplexer Strings mit wenig Code sowie das Anlegen von **Arrays**, um mehrere Werte zu speichern. Abschließend kümmern wir uns noch um die Frage: »*Warum sind Objekte für das Leben in Kotlinville so wichtig?*«



Ihr Code braucht Variablen	32
Was passiert, wenn Sie eine Variable deklarieren	33
Kotlins grundsätzliche Datentypen	35
Variablentypen explizit angeben	37
Den richtigen Wert für den Variablentyp verwenden	38
Einen Wert einer anderen Variablen zuweisen	39
Wir müssen den Wert konvertieren	40
Was passiert, wenn Sie einen Wert konvertieren?	41
Aufpassen, dass nichts überläuft	42
Mehrere Werte in einem Array speichern	45
Die Phras-O-Matic-Applikation erstellen	46
Den Code zu PhrasOMatic.kt hinzufügen	47
Der Compiler leitet den Arraytyp aus dessen Werten ab	49
var heißt, die Variable kann auf ein anderes Array verweisen	50
val bedeutet, die Variable verweist während der gesamten Laufzeit auf dasselbe Array ...	51
Vermischte Referenzen	54
Ihr Kotlin-Werkzeugkasten	58

Funktionen

Raus aus main

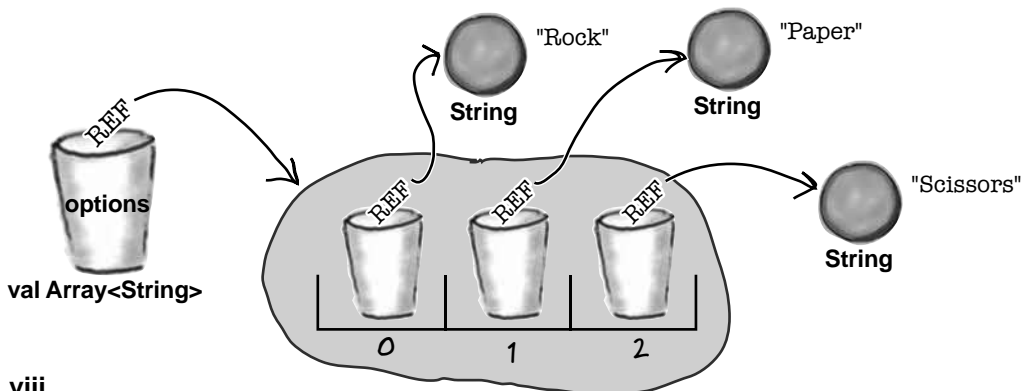
3

Es ist Zeit für den nächsten Schritt: Funktionen.

Bisher befand sich der Code ausschließlich in der *main*-Funktion Ihrer Applikation. Wenn Sie Ihren Code **besser organisieren** und **leichter pflegen** wollen, müssen Sie wissen, **wie Sie den Code in separate Funktionen aufteilen können**. In diesem Kapitel lernen Sie, wie man **Funktionen schreibt** und damit **interagiert**, indem Sie ein Spiel programmieren. Wir zeigen Ihnen, wie man kompakte **Einzelausdrucksfunktionen** schreibt. Und unterwegs finden Sie auch noch heraus, wie man **über Bereiche (Ranges) und Sammlungen (Collections) iteriert** und wie die mächtige *for*-Schleife funktioniert.



Ein Spiel programmieren: Stein, Schere, Papier	60
Zuerst das allgemeine Konzept	61
Das Spiel soll eine Auswahl treffen	63
Funktionen erstellen	64
Funktionen können mehrere Parameter haben	65
Funktionen können Dinge zurückgeben	66
Funktionskörper mit einzelnen Ausdrücken	67
Die <code>getGameChoice</code> -Funktion in <code>Game.kt</code> einbauen	68
Die <code>getUserChoice</code> -Funktion	75
Wie <i>for</i> -Schleifen funktionieren	76
Benutzer zur Eingabe ihrer Auswahl auffordern	78
Vermischte Ausgaben	79
Wir müssen die Benutzereingaben validieren	81
Die <code>getUserChoice</code> -Funktion in <code>Game.kt</code> einbauen	83
Die <code>printResult</code> -Funktion in <code>Game.kt</code> einbauen	87
Ihr Kotlin-Werkzeugkasten	89



Klassen und Objekte

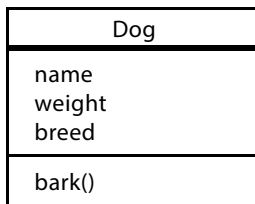
Etwas mehr Klasse

4

Jetzt ist es Zeit, über Kotlins Basistypen hinaus weiterzublicken.

Früher oder später sind Kotlins Basisdatentypen nicht mehr genug. Sie wollen *mehr*. Und da kommen *Klassen* ins Spiel. Klassen sind *Vorlagen*, mit denen Sie **Ihre eigenen Objekttypen erstellen** und deren Eigenschaften und Funktionen Sie selbst definieren können. In diesem Kapitel lernen Sie, wie Sie **eigene Klassen entwickeln und definieren** und wie diese verwendet werden, um **neue Arten von Objekten zu erstellen**. Sie werden **Konstrukturen** und **Initialisierungsblocks**, **Getter und Setter** kennenlernen und herausfinden, wie sie benutzt werden können, um Ihre Eigenschaften zu schützen. Schließlich werden Sie lernen, wie Verkapselung (»Data Hiding«) **in sämtlichem Kotlin-Code** bereits eingebaut ist, wodurch Sie Zeit, Aufwand und eine Menge Tipparbeit sparen können.

Eine Klasse



Viele Objekte



Objekttypen werden über Klassen definiert	92
Eigene Klassen entwickeln	93
Eine Dog-Klasse erstellen	94
Ein Dog-Objekt erstellen	95
Auf Eigenschaften und Funktionen zugreifen	96
Eine Songs-Applikation programmieren	97
Das Geheimnis der Objekterstellung	98
Objekterstellung im Detail	99
Hinter den Kulissen: Aufruf des Dog-Konstruktors	100
Eigenschaften im Detail	105
Flexible Eigenschafteninitialisierung	106
Initialisierungsblocks verwenden	107
Sie MÜSSEN Ihre Eigenschaften initialisieren	108
Eigenschaftswerte validieren	111
Einen eigenen Getter schreiben	112
Einen eigenen Setter schreiben	113
Der komplette Code für das Dogs-Projekt	115
Ihr Kotlin-Werkzeugkasten	120



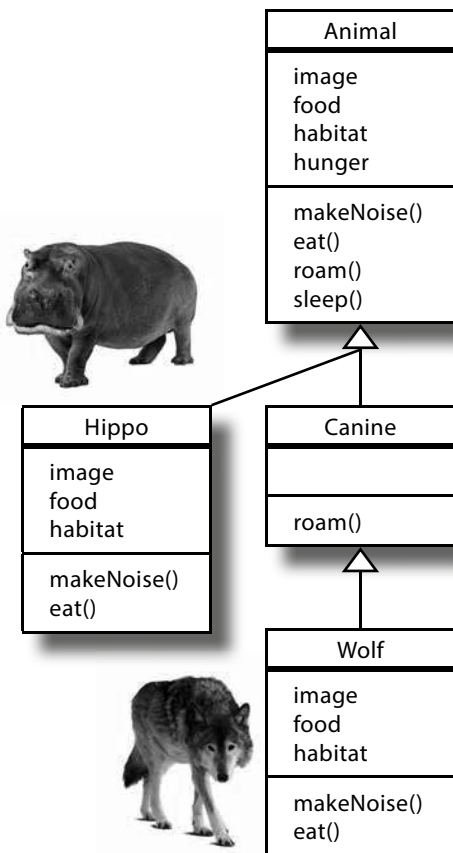
Subklassen und Superklassen

5

Vererbung

Manchmal begegnet man Objekttypen, die ideal wären, wenn man nur ein paar Kleinigkeiten ändern könnte!

Genau das ist einer der Vorteile von **Vererbung**. In diesem Kapitel lernen Sie das Erstellen von **Subklassen** und erfahren, wie Sie Eigenschaften und Funktionen einer **Superklasse** erben können. Sie lernen, wie man **Funktionen und Eigenschaften überschreibt**, damit sich Klassen so verhalten, wie **Sie** es wollen. Außerdem erfahren Sie, wann Vererbung sinnvoll ist (und wann nicht). Schließlich zeigen wir Ihnen, wie Vererbung dabei hilft, **doppelten Code** zu vermeiden, und wie Sie Ihre Flexibilität mit Hilfe von **Polymorphismus** steigern können.



Vererbung hilft, doppelten Code zu vermeiden	122
Was wir vorhaben	123
Eine Vererbungsstruktur für Tier-Klassen entwickeln	124
Duplizierten Code in Subklassen durch Vererbung vermeiden	125
Was sollen die Subklassen überschreiben?	126
Wir können einige Tiere gruppieren	127
Canine- und Feline-Klassen hinzufügen	128
Die Klassenhierarchie mit dem IST-EIN-Test überprüfen	129
Der IST-EIN-Test funktioniert im gesamten Vererbungsbaum	130
Wir erstellen ein paar Kotlin-Tiere	133
Die Superklasse und ihre Eigenschaften als »offen« deklarieren	134
Wie eine Subklasse von einer Superklasse erbt	135
Wie (und wann) Eigenschaften überschrieben werden	136
Beim Überschreiben von Eigenschaften können nicht nur Standardwerte zugewiesen werden	137
Funktionen überschreiben	138
Eine überschriebene Funktion oder Eigenschaft bleibt »offen« ...	139
Die Hippo-Klasse in das Animals-Projekt einbauen	140
Die Canine- und Wolf-Klassen einbauen	143
Welche Funktion wird aufgerufen?	144
Wenn Sie eine Funktion an einer Variablen aufrufen, antwortet die Version des Objekts	146
Sie können einen Supertyp für die Parameter und den Rückgabetyt einer Funktion verwenden	147
Der aktualisierte Animals-Code	148
Ihr Kotlin-Werkzeugkasten	153

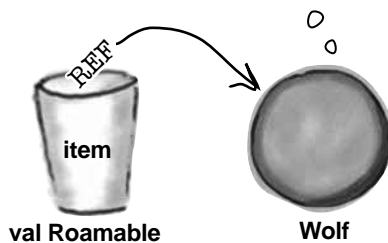
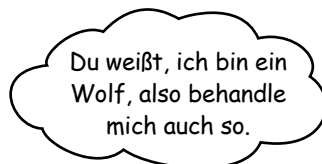
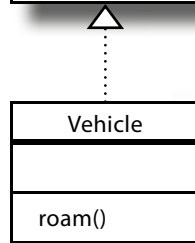
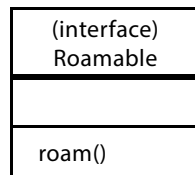
Abstrakte Klassen und Interfaces

Ernsthafter Polymorphismus

6

Eine Superklassen-Vererbungshierarchie ist nur der Anfang.

Wenn Sie alle Möglichkeiten des *Polymorphismus nutzen wollen*, müssen Sie beim Design **abstrakte Klassen und Interfaces (Schnittstellen) einsetzen**. In diesem Kapitel lernen Sie, wie man abstrakte Klassen verwendet, um zu kontrollieren, welche Klassen Ihrer Hierarchie **instanziiert werden können und welche nicht**. Sie werden sehen, wie man konkrete Subklassen dazu zwingt, **ihre eigenen Implementierungen zu verwenden**. Und Sie werden erfahren, wie man Interfaces benutzt, um **Verhalten zwischen unabhängigen Klassen** zu teilen. Zwischendurch zeigen wir Ihnen noch, was es mit **is**, **as** und **when** auf sich hat.



Ein weiterer Blick auf die Animal-Klassenhierarchie	156
Einige Klassen sollten nicht instanziiert werden	157
Abstrakt oder konkret?	158
Abstrakte Klassen können abstrakte Eigenschaften und Funktionen enthalten	159
Die Animal-Klasse hat zwei abstrakte Funktionen	160
Eine abstrakte Klasse implementieren	162
Abstrakte Eigenschaften und Funktionen MÜSSEN implementiert werden	163
Das Animals-Projekt aktualisieren	164
Unabhängige Klassen können gemeinsames Verhalten haben	169
Über ein Interface können Sie gemeinsames Verhalten AUSSERHALB der Superklassenhierarchie definieren	170
Das Roamable-Interface definieren	171
Eigenschaften für Interfaces definieren	172
Deklarieren, dass eine Klasse ein Interface implementiert ...	173
Mehrfache Interfaces implementieren	174
Wie kann ich entscheiden, ob ich eine Klasse, eine Unterklasse, eine abstrakte Klasse oder ein Interface benutzen soll?	175
Das Animals-Projekt aktualisieren	176
Polymorphismus funktioniert auch mit Interfaces	181
Wann sollte man den is-Operator verwenden?	182
Benutzen Sie when, um eine Variable mit einer Reihe von Optionen zu vergleichen	183
Der is-Operator führt eine automatische Typumwandlung (Smart Casting) durch	184
Explizite Typumwandlung mit as	185
Das Animals-Projekt aktualisieren	186
Ihr Kotlin-Werkzeugkasten	189

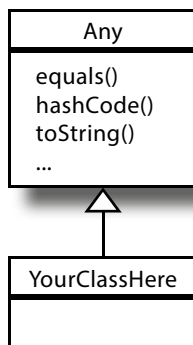
Datenklassen

7

Mit Daten umgehen

Niemand will sein Leben lang das Rad neu erfinden.

Die meisten Applikationen enthalten Klassen, die ausschließlich für die *Datenspeicherung* zuständig sind. Um Ihnen das Programmieren zu erleichtern, haben die Kotlin-Entwickler sich das Konzept der **Datenklassen** ausgedacht. In diesem Kapitel zeigen wir Ihnen, wie Sie Datenklassen verwenden, um Code zu schreiben, der **sauberer und knapper** ist, als Sie es je für möglich gehalten haben. Sie werden die Datenklassen-**Hilfsfunktionen** kennenlernen und entdecken, wie man ein **Datenobjekt in seine Bestandteile destruktuieren** kann. Unterwegs zeigen wir Ihnen außerdem, wie **Standardparameter** Ihren Code flexibler machen können. Außerdem stellen wir Ihnen **Any** vor, die *Mutter aller Superklassen*.



**Datenobjekte
gelten als gleich,
wenn sie die
gleichen Eigen-
schaftswerte
enthalten.**

== ruft eine Funktion namens equals auf	192
equals wird von einer Superklasse namens Any geerbt	193
Das von Any definierte gemeinsame Verhalten	194
Vielleicht soll equals testen, ob zwei Objekte gleichwertig sind	195
Mit einer Datenklasse können Sie Datenobjekte erstellen	196
Datenklassen überschreiben das geerbte Verhalten	197
Daten mit der copy-Funktion kopieren	198
Datenklassen definieren componentN-Funktionen ...	199
Das Recipes-Projekt anlegen	201
Vermischte Nachrichten	203
Erzeugte Funktionen verwenden nur die im Konstruktor definierten Eigenschaften	205
Die Initialisierung vieler Eigenschaften kann zu schwerfälligem Code führen	206
Die Standardwerte des Konstruktors verwenden	207
Auch Funktionen können Standardwerte verwenden	210
Funktionen überladen	211
Das Recipes-Projekt aktualisieren	212
Der Code (Fortsetzung)	213
Ihr Kotlin-Werkzeugkasten	217

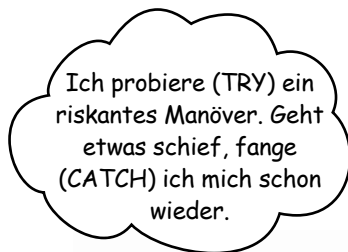
Nullwerte und Ausnahmen

8

Gesund und munter

Alle wollen sicheren Code schreiben.

Die gute Nachricht ist: Kotlin wurde *mit dem Ziel der Codesicherheit* entwickelt. Wir beginnen, indem wir Ihnen Kotlin's **nullwertfähige Datentypen** zeigen und warum dadurch *in Kotlinville so gut wie keine Fehler vom Typ NullPointerException auftreten*. Sie werden erfahren, wie man *sichere Aufrufe* durchführt und wie Kotlin's **Elvis-Operator** verhindert, dass Sie vollkommen durcheinanderkommen (*»All shook up«*). Wenn wir damit fertig sind, zeigen wir Ihnen noch, wie Sie **Ausnahmen auslösen und abfangen können wie ein Profi**.



Wie entfernt man Objektreferenzen aus Variablen?	220
Eine Objektreferenz mit null entfernen	221
Nullwertfähige Typen können überall genutzt werden, wo auch nicht nullwertfähige Typen möglich sind	222
Ein Array mit nullwertfähigen Typen erstellen	223
Auf Funktionen und Eigenschaften eines nullwertfähigen Typs zugreifen	224
Sichere Aufrufe (<i>»Safe Calls«</i>)	225
Sichere Aufrufe können verkettet werden	226
Die Geschichte geht weiter ...	227
Sichere Aufrufe für die Zuweisung von Werten verwenden ...	228
let verwenden, um Code auszuführen, wenn Werte nicht null sind	231
let mit Arrayelementen verwenden	232
Anstatt einen Ausdruck zu benutzen ...	233
Der !!-Operator löst absichtlich einen NullPointerException-Fehler aus	234
Das Projekt Null Values bauen	235
Der Code (Fortsetzung)	236
In außergewöhnlichen Situationen wird eine Ausnahme ausgelöst	239
Ausnahmen mit try/catch abfangen	240
Dinge mit finally auf jeden Fall ausführen	241
Eine Ausnahme ist ein Objekt vom Typ Exception	242
Sie können Ausnahmen selbst auslösen	244
try und throw sind Ausdrücke	245
Ihr Kotlin-Werkzeugkasten	250

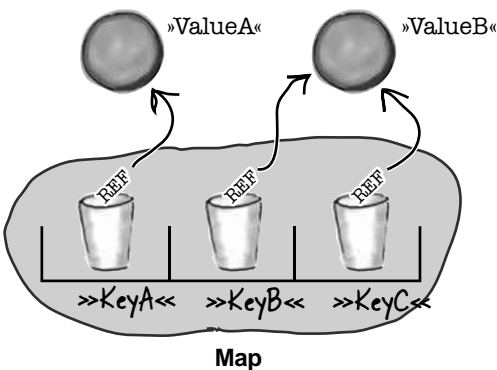
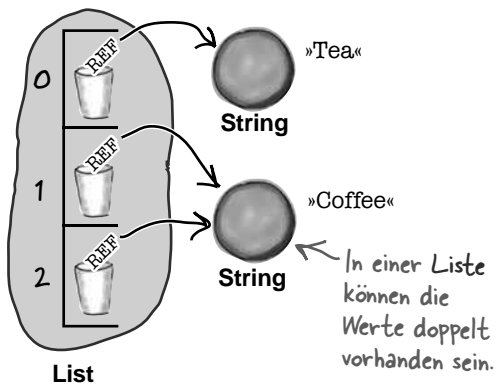
Collections

Dinge organisieren

9

Haben Sie jemals etwas Flexibleres als ein Array gebraucht?

Kotlin besitzt eine Reihe nützlicher **Collections** (Sammlungen), die mehr Flexibilität und eine größere Kontrolle über **die Speicherung und Verwaltung von Objektgruppen** bieten. Brauchen Sie eine **erweiterbare Liste**? Wollen Sie den Inhalt **mischen oder umkehren**? Wollen Sie **etwas anhand seines Namens finden**? Oder **wollen Sie Duplikate entfernen**, ohne auch nur einen Finger rühren zu müssen? Wenn Sie auch nur eines dieser Merkmale brauchen, lesen Sie weiter. In diesem Kapitel finden Sie, was Sie suchen ...



In einer Map dürfen die Werte doppelt vorhanden sein. Die Schlüssel müssen dagegen einmalig sein.

Arrays können nützlich sein ...	252
... mit manchen Dingen können Arrays aber nicht umgehen	253
Im Zweifel gehen Sie zur Bibliothek	254
List, Set und Map	255
Fantastische Listen ...	256
Eine mutable Liste erstellen ...	257
Sie können Werte entfernen ...	258
Reihenfolge ändern und mehrere Änderungen gleichzeitig durchführen ...	259
Das Collections-Projekt anlegen	260
Listen dürfen doppelte Werte enthalten	263
Ein Set anlegen	264
Wie ein Set auf Duplikate testet	265
Hashcodes und Gleichheit	266
Regeln für das Überschreiben von hashCode und equals	267
Ein MutableSet verwenden	268
Das Collections-Projekt aktualisieren	270
Zeit für Maps	276
Eine Map benutzen	277
Eine MutableMap erstellen	278
Einträge aus einer MutableMap entfernen	279
Maps und MutableMaps können kopiert werden	280
Der vollständige Code für unser Collections-Projekt	281
Vermischte Nachrichten	285
Ihr Kotlin-Werkzeugkasten	287

10

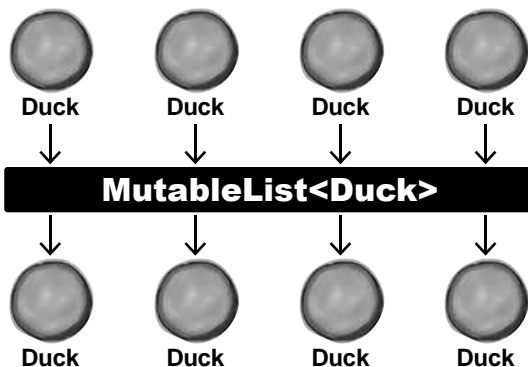
Generische Programmierung

Innen und außen unterscheiden

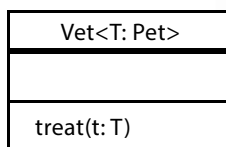
Alle mögen konsistenten Code.

Generics helfen dabei, konsistenteren Code zu schreiben, der weniger Probleme verursacht. In diesem Kapitel zeigen wir Ihnen, wie **Kotlins Collection-Klassen Generics** einsetzen, um zu verhindern, dass Sie versehentlich einen Salat in einer List<Seemöwe> speichern. Sie werden lernen, wann und wie Sie **Ihre eigenen generischen Klassen, Interfaces und Funktionen** erstellen und wie Sie einen **generischen Typ** auf einen bestimmten Supertyp beschränken können. Schließlich lernen Sie, wie Sie mit **Kovarianz und Kontravarianz** das Verhalten generischer Typen SELBST bestimmen können.

Mit Generics werden ausschließlich Referenzen auf Duck-Objekte IN der MutableList gespeichert ...



... und kommen als Referenzen auf den Objekttyp Duck auch wieder HERAUS.



Collections verwenden Generics	290
Eine MutableList definieren	291
Typparameter mit einer MutableList verwenden	292
Möglichkeiten generischer Klassen und Interfaces	293
Diese Schritte wollen wir abarbeiten.	294
Die Pet-Klassenhierarchie erstellen	295
Die Contest-Klasse definieren	296
Die scores-Eigenschaft hinzufügen	297
Die getWinners-Funktion erstellen	298
Ein paar Contest-Objekte erstellen	299
Das Generics-Projekt erstellen	301
Die Retailer-Hierarchie	305
Das Retailer-Interface definieren	306
Wir können CatRetailer-, DogRetailer- und FishRetailer-Objekte erzeugen ...	307
out verwenden, um den generischen Typ kovariant zu machen	308
Das Generics-Projekt aktualisieren	309
Wir brauchen eine Vet-Klasse	313
Vet-Objekte erzeugen	314
Verwenden Sie in, um einen generischen Typ kontravariant zu machen	315
Ein generischer Typ kann lokal kontravariant sein	316
Das Generics-Projekt aktualisieren	317
Ihr Kotlin-Werkzeugkasten	324

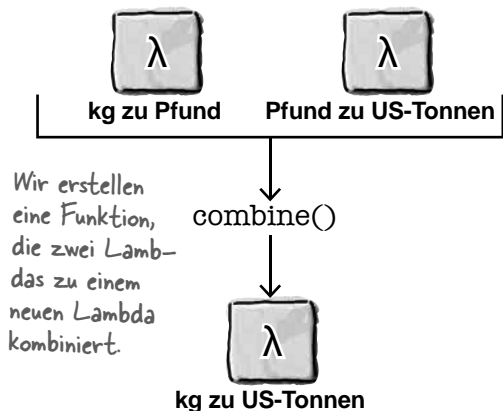
Lambdas und Funktionen höherer Ordnung

Code wie Daten behandeln

11

Wollen Sie Code schreiben, der noch flexibler und mächtiger ist?

Falls ja, brauchen Sie **Lambdas**. Ein *Lambda* – oder *Lambda-Ausdruck* (offiziell auch Lambda-Funktion oder anonyme Funktion) – ist ein Codeblock, den Sie wie ein Objekt herumreichen können. In diesem Kapitel erfahren Sie, wie man ein **Lambda definiert, es einer Variablen zuweist** und **seinen Code ausführt**. Sie lernen verschiedene **Funktionsstypen** kennen und wie diese Ihnen beim Schreiben von **Funktionen höherer Ordnung** helfen können, die Lambdas für ihre Parameter- und Rückgabewerte zu benutzen. Nebenbei zeigen wir Ihnen noch, wie ein wenig **syntaktischer Zucker das Programmiererleben etwas versüßen kann**.

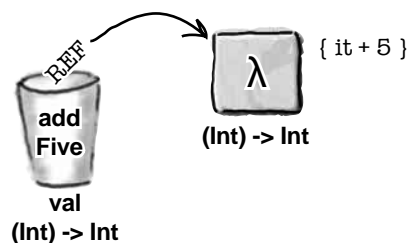


Ich übernehme zwei Int-Parameter namens x und y, addiere sie und gebe das Ergebnis zurück.

```
λ { x: Int, y: Int -> x + y }
```

xvi

Einführung in Lambdas	326
Wie Lambdas aussehen	327
Lambdas können Variablen zugewiesen werden	328
Lambda-Ausdrücke haben einen Typ	331
Der Compiler kann die Lambda-Parametertypen ableiten	332
Das richtige Lambda für einen bestimmten Variablentyp verwenden	333
Das Lambdas-Projekt anlegen	334
Lambdas können an Funktionen übergeben werden	339
Das Lambda im Funktionskörper aufrufen	340
Was beim Aufruf der Funktion passiert	341
Das Lambda aus den runden Klammern befreien ...	343
Das Lambdas-Projekt aktualisieren	344
Funktionen können Lambdas zurückgeben	347
Eine Funktion, die Lambdas übernimmt UND zurückgibt	348
Die combine-Funktion benutzen	349
Verwenden Sie typealias, um einen anderen Namen für einen vorhandenen Typ anzugeben	353
Das Lambdas-Projekt aktualisieren	354
Ihr Kotlin-Werkzeugkasten	361



Eingebaute Funktionen höherer Ordnung

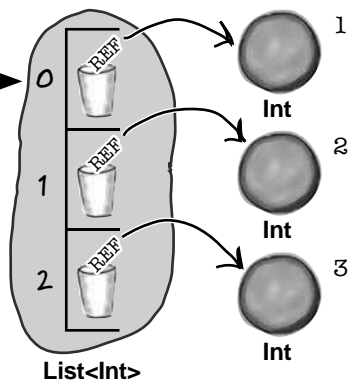
Dem Code Beine machen**12****Kotlin besitzt eine Vielzahl eigener Funktionen höherer Ordnung.**

In diesem Kapitel werden wir Ihnen einige davon vorstellen. Sie werden die flexible **filter-Familie** kennenlernen und erfahren, wie Sie Ihre Collections damit auf die richtige Größe zurechtstutzen können. Außerdem zeigen wir Ihnen, wie Sie eine **Collection mit map umwandeln, per forEach über ihre Elemente iterieren** und die enthaltenen **Elemente per groupBy ordnen können**. Darüber hinaus erläutern wir Ihnen, wie Sie **fold** benutzen können, um komplexe Berechnungen *mit nur einer Codezeile* durchzuführen. Am Ende dieses Kapitels werden Sie **mächtigeren Code** schreiben können, **als Sie je gedacht haben**.

Diese Artikel/Elemente haben keine natürliche Reihenfolge. Um den niedrigsten oder höchsten Wert zu finden, müssen wir bestimmte Kriterien angeben, beispielsweise `unitPrice` (Preis) oder `quantity` (Menge).



Die fold-Funktion beginnt mit dem ersten Element der Collection.



Kotlin besitzt eine Vielzahl eingebauter Funktionen höherer Ordnung	364
Die Funktionen <code>min</code> und <code>max</code> arbeiten mit Basisdatentypen	365
Ein näherer Blick auf die Lambda-Parameter von <code>maxBy</code> und <code>minBy</code>	366
Die Funktionen <code>sumBy</code> und <code>sumByDouble</code>	367
Das Groceries-Projekt	368
Willkommen bei der <code>filter</code> -Funktion	371
<code>map</code> verwenden, um eine Collection umzuwandeln	372
Was passiert, wenn der Code ausgeführt wird	373
Die Geschichte geht weiter ...	374
<code>forEach</code> funktioniert wie eine Schleife	375
<code>forEach</code> hat keinen Rückgabewert	376
Das Groceries-Projekt aktualisieren	377
Collections mit <code>groupBy</code> gruppieren	381
Sie können <code>groupBy</code> in verketteten Funktionsaufrufen verwenden	382
Die <code>fold</code> -Funktion	383
Hinter den Kulissen der <code>fold</code> -Funktion	384
Weitere Beispiele für <code>fold</code>	386
Das Groceries-Projekt aktualisieren	387
Vermischte Nachrichten	391
Ihr Kotlin-Werkzeugkasten	394
Raus aus der Stadt ...	395

Koroutinen

Code parallel ausführen



Manche Aufgaben laufen am besten im Hintergrund.

Sollen Daten von einem langsamen externen Server gelesen werden, wollen Sie vermutlich nicht bis zum Ende danebensitzen und Däumchen drehen. In solchen Fällen sind **Koroutinen Ihre neuen besten Freunde**. Mit Koroutinen können Sie Code **asynchron ausführen**, das heißt *weniger Däumchen drehen* und eine *bessere Benutzbarkeit*. Außerdem können Ihre Applikationen durch Koroutinen *skalierbarer* werden. Wenn Sie weiterlesen, werden Sie das Geheimnis lüften, wie Sie gleichzeitig mit Bob reden und Suzy zuhören können.



Bam! Bam! Bam! Bam! Bam! Bam!
Tish! Tish!

↖ Jetzt werden Toms und Becken parallel gespielt.

Testen

Ziehen Sie Ihren Code zur Rechenschaft



Jeder weiß, dass guter Code funktionieren muss.

Aber jede Codeänderung birgt das Risiko neuer Bugs, die verhindern, dass Ihr Code wie gewünscht funktioniert. Darum ist *sorgfältiges Testen* so wichtig: Sie erfahren von möglichen Problemen im Code, *bevor er in einer Produktionsumgebung eingesetzt wird*. In diesem Anhang besprechen wir **JUnit** und **KotlinTest**, zwei Bibliotheken für die Durchführung von **Unit-Tests**. Dadurch haben Sie *grundsätzlich ein Sicherheitsnetz zur Verfügung*.

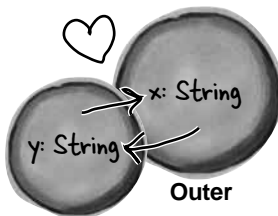
Was übrig bleibt

Die Top Ten der Themen, die wir nicht behandelt haben



Nach allem, was wir behandelt haben, gibt es immer noch ein paar weitere wichtige Dinge.

Ein paar Themen haben wir noch für Sie. Wir wollten sie nicht ignorieren, aber es war uns wichtig, dass man unser Buch noch hochheben kann, ohne vorher ein Fitnessstudio besuchen zu müssen. Bevor Sie das Buch zur Seite legen, sollten Sie sich **diese Leckerbissen** nicht entgehen lassen.



Inner

Die Inner- und Outer-Objekte haben eine besondere Beziehung zueinander. Inner kann auf die Variablen von Outer zugreifen und umgekehrt.

1. Packages und Importe	416
2. Die Sichtbarkeit von Code steuern	418
3. enum-Klassen	420
4. Versiegelte Klassen	422
5. Verschachtelte und innere Klassen	424
6. Objektdeklarationen und -ausdrücke	426
7. Erweiterungen (Extensions)	429
8. return, break und continue	430
9. Mehr Spaß mit Funktionen	432
10. Interoperabilität	434