Der Inhalt (im Überblick)

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?

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 *bedingungsbasierte Verzweigungen*. Ihre Reise hat gerade erst begonnen …

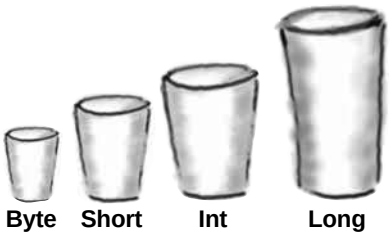
Die Möglichkeit, auszuwählen, gegen welche Plattform Ihr Code kompiliert wird, bedeutet, dass Kotlin auf Servern, in der Cloud, in Browsern, auf Mobilgeräten und mehr funktioniert.

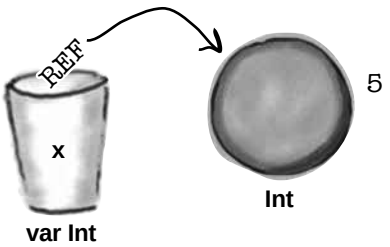
Basistypen und Variablen

2 Eine Variable sein

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?*«



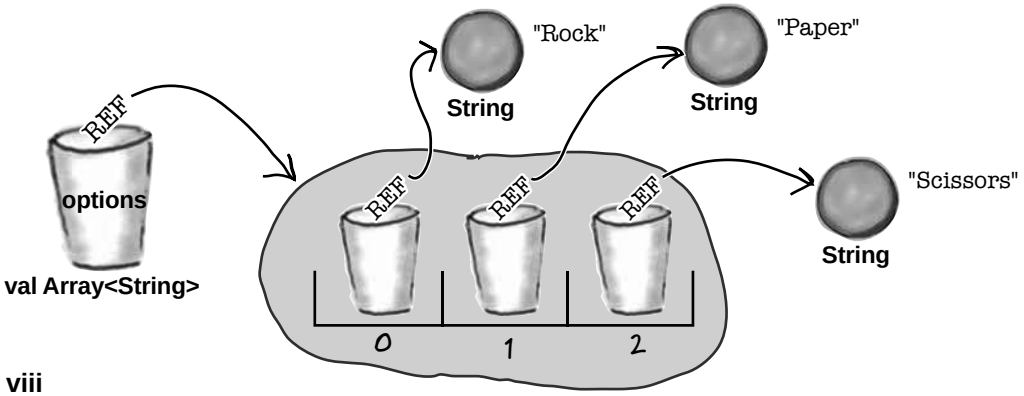


Funktionen

3 Raus aus main

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.



Klassen und Objekte

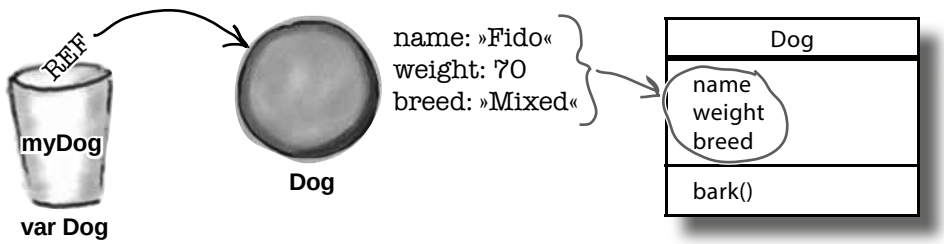
4 Etwas mehr Klasse

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 ***Konstruktoren*** 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

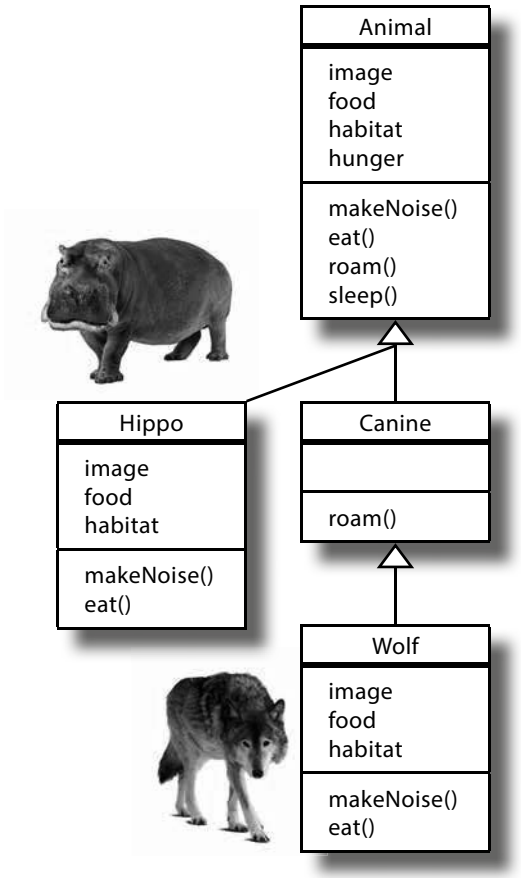


Subklassen und Superklassen

5 Vererbung

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

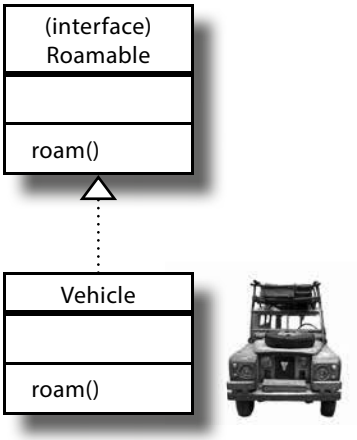
Genau das ist einer der Vorteile von **Vererbung**. In diesem Kapitel lernen Sie das Erstellen von **Subklassen** und erfahren, wie Sie Eigenschaften und Funktionen einer **Superklasse** erben können. Sie lernen, wie man ***Funktionen und Eigenschaften überschreibt***, damit sich Klassen so verhalten, 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 mithilfe von **Polymorphismus** steigern können.

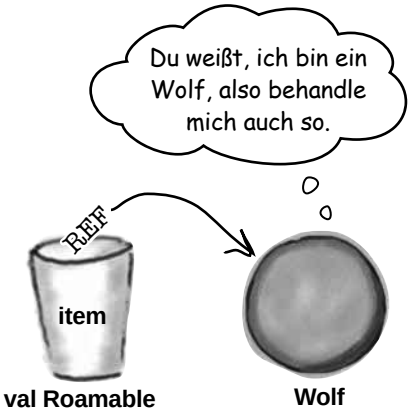


6 Abstrakte Klassen und Interfaces
Ernsthafter Polymorphismus

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.



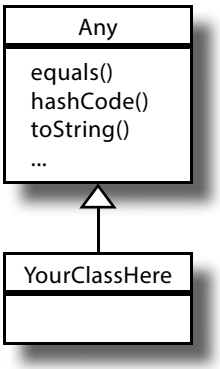


Datenklassen

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 destrukturieren*** 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 Eigenschaftswerte enthalten.

8 Nullwerte und Ausnahmen

Gesund und munter

Alle wollen sicheren Code schreiben.

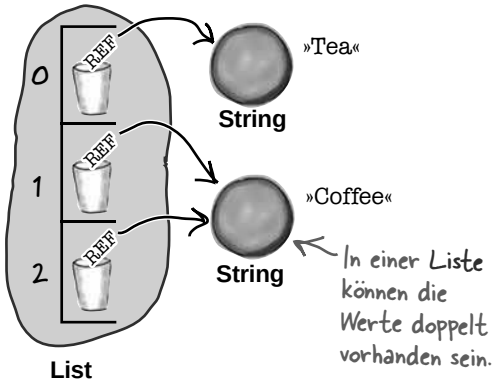
Die gute Nachricht ist: Kotlin wurde *mit dem Ziel der Codesicherheit* entwickelt. Wir beginnen, indem wir Ihnen Kotlins **nullwertfähige Datentypen** zeigen und warum dadurch *in Kotlinville so gut wie keine Fehler vom Typ NullPointerException auftreten*. Sie werden erfahren, wie man *sichere Aufrufe* durchführt und wie Kotlins **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**.

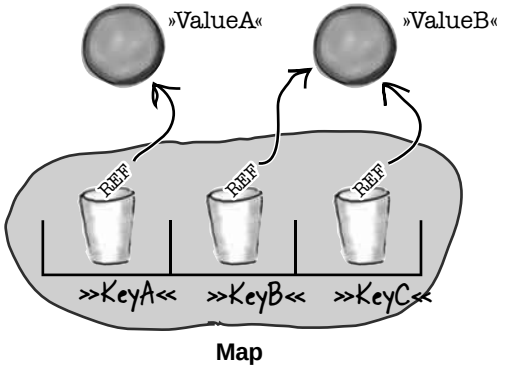
Collections

9 Dinge organisieren

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.

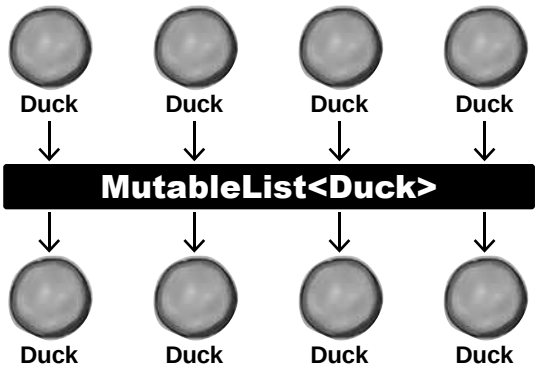
Generische Programmierung

10 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 ...



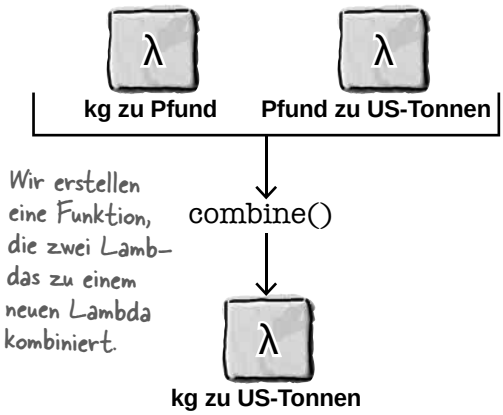
... und kommen als Referenzen auf den Objekttyp Duck auch wieder HERAUS.

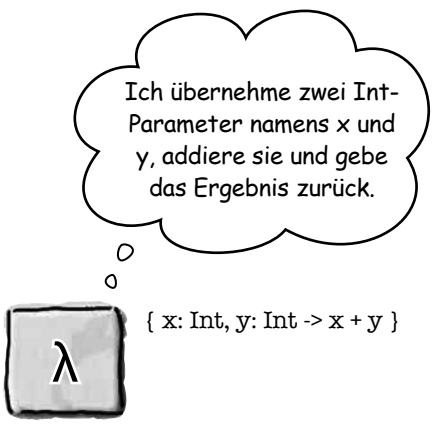
Lambdas und Funktionen höherer Ordnung

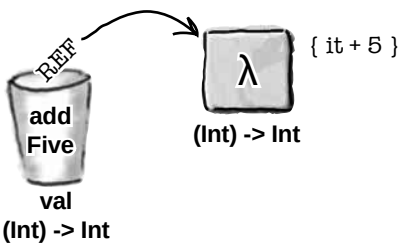
11 Code wie Daten behandeln

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 **Funktionstypen** 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***.





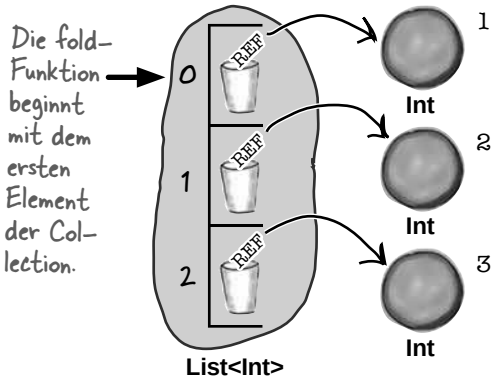


Eingebaute Funktionen höherer Ordnung

12 Dem Code Beine machen

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**.



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.

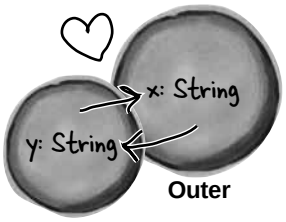
Aber jede Codeänderung birgt das Risiko neuer Bugs, die verhindern, dass Ihr Code wie gewünscht funktioniert. Darum ist *sorgfältiges Testen* so wichtig: Sie erfahren von möglichen Problemen im Code, *bevor er in einer Produktionsumgebung eingesetzt wird*. In diesem Anhang besprechen wir ***JUnit*** 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.



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