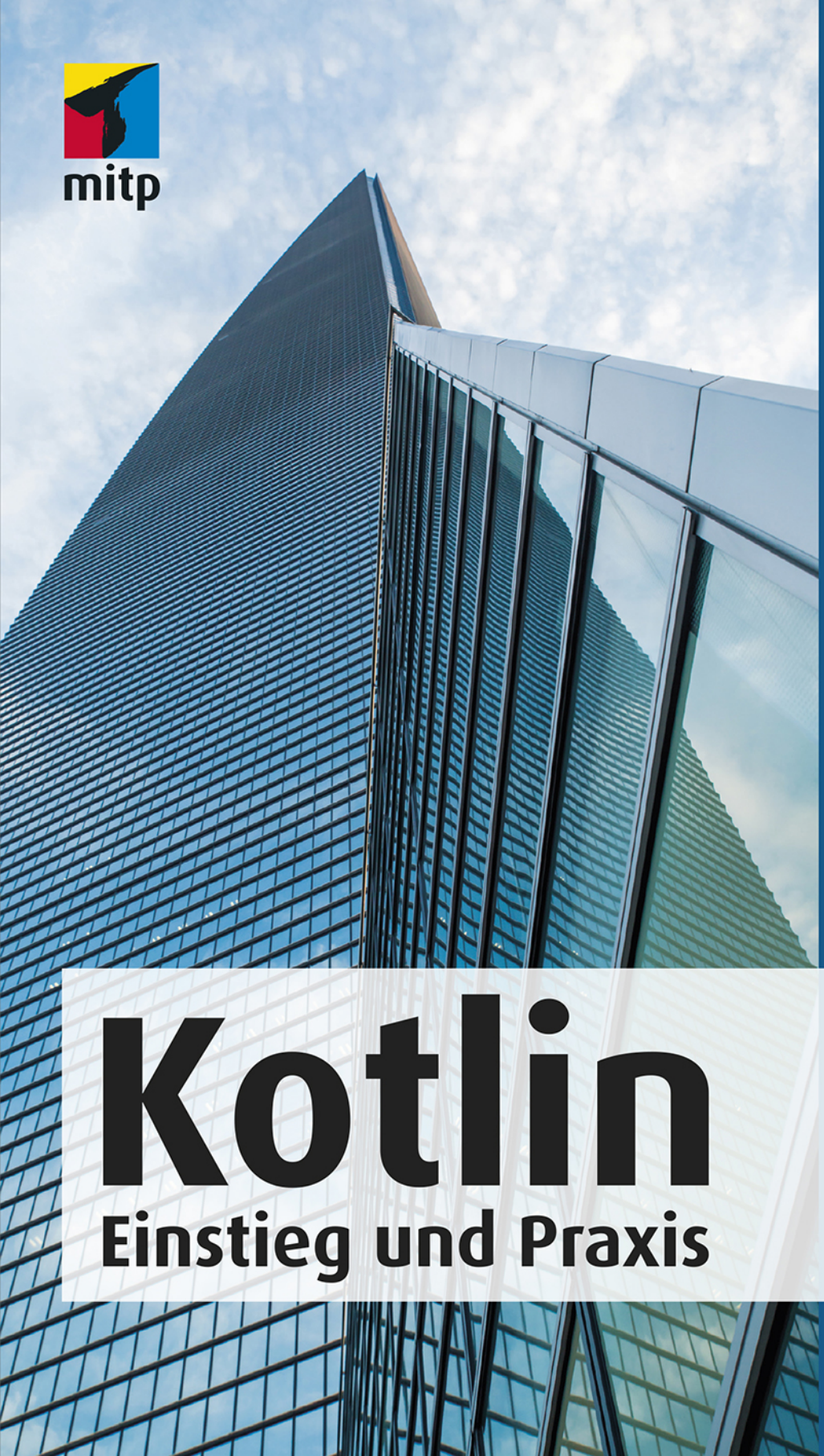

mitp

Karl
Szwillus

Kotlin

Einstieg und Praxis

Inhaltsverzeichnis

Einleitung

Fast jeder kennt das unangenehme Gefühl, einer wildfremden Person zuzuwinken, weil man kurz dachte, die freundliche Geste gälte einem selbst. Galt sie natürlich nicht, sondern der Person, die sich in direkter Linie hinter der eigenen Schulter im toten Winkel versteckte. Auf der Google I/O 2017 habe ich erlebt, wie sich ein ganz ähnliches Empfinden einstellt, wenn man eine halbe Stunde lang in eine Fernsehkamera blickt, die in etwas mehr als Armeslänge vor einem positioniert ist. Selbstverständlich wusste ich, dass es keinen Grund gab, mich in dieser Weise zu portraitieren. Doch nach fünf Minuten ... und weiteren zehn ... es konnte doch nicht?

Konnte es natürlich nicht. Es ging um einen 17-jährigen High-School Studenten, den der Lebensweg seiner Eltern als Einwanderer aus Afghanistan und die in Amerika erlebte Hilfsbereitschaft inspiriert hat, selbst etwas zurückzugeben. In der Reihe vor mir. Sein Projekt verbesserte ohne teure Hardware die Aussagefähigkeit von Mammographien mithilfe von künstlicher Intelligenz. Die von ihm programmierten Routinen erkennen Muster in den Bildern und können so auch ohne medizinisches Fachpersonal präzisere Vorabdiagnosen zur Verfügung stellen.[1]

Eine knappe halbe Stunde zuvor gab es eine Ankündigung, die mir diesen Tag auch ohne die bemerkenswert große Kamera im Gedächtnis gehalten hätte: Die Ankündigung von Stephanie Saad Cuthbertson (Director Product Management Android zu der Zeit), Kotlin zu einer vollständig unterstützten Programmiersprache für Android zu erklären und mit dem Entwickler *JetBrains* eine Gesellschaft zur Weiterentwicklung der Sprache zu gründen. Kotlin ist eine vergleichsweise junge Programmiersprache, deren erste stabile Version 2016 erschien. Seit 2011 wird sie von einem Team des Herstellers der Entwicklungsumgebung *IntelliJ* weiterentwickelt und gepflegt. Hier besteht auch die offensichtliche Verbindung zu Java und Android, denn auch die von Google primär unterstützte *IDE* (Integrated Development Environment oder Entwicklungsumgebung) Android Studio basiert auf IntelliJ.

Dadurch war auch gewährleistet, dass es in Bezug auf die Tools keiner großen Umgewöhnung bedurfte. Und so haben sicherlich einige Android-Entwickler nach

1 Im Video der Keynote bei 1:50:50 können sich die Leserinnen und Leser dieses Buchs ein Bild von mir im Jahr 2017 machen. Ganz Kamera-Unprofi erkennt man auch den Moment, in dem ich begriffen habe, dass es ganz sicher nicht um mich geht.

dieser Ankündigung 2017 ihre ersten Schritte in Kotlin mithilfe der automatischen Konvertierungsfunktion gemacht und schnell eines erkannt: Die neue Sprache löst sofort Probleme im bestehenden Code ihrer Anwendung:

- Der Code wird kürzer und übersichtlicher
- Der Code wird sicherer
- Es ist kein Problem, Kotlin und Java miteinander in einem Projekt einzusetzen
- Keine Semikolons(!)

Oder, um es mit den Worten von Google und Jetbrains zu sagen: Wichtige Merkmale von Kotlin sind: Es ist effektiv durch präzise Schreibweisen (*concise*), sicher (*safe*) und gut mit verschiedenen Umgebungen zu integrieren (*interoperable*).

Wie das in der Praxis aussieht und warum ich schon nach den ersten Arbeitstagen mit Kotlin kein Java mehr sehen mochte, versuche ich in diesem Buch zu erklären.

Ausrichtung des Buchs

Dieses Buch soll Entwicklern einen Einstieg in die Programmiersprache Kotlin ermöglichen. Es konzentriert sich auf die Grundlagen der Sprache und ihre Strukturen, Befehle und Sprachfeatures. Zusätzlich zur Erläuterung der ursprünglichen Aufgabe als ein verbesserter Java-Ersatz wird das aktuelle Haupteinsatzfeld Android behandelt und wie Kotlin die Android-Entwicklung verbessern – oder neutral: verändern – kann. Dabei werden aber Kenntnisse von Android vorausgesetzt, um den Fokus auf die Sprache Kotlin beizubehalten.

Das vorliegende Buch möchte dazu motivieren, vor einem Volleinstieg in Kotlin die Konvertierung Schritt für Schritt zu probieren. Das Stichwort lautet Interoperabilität. Auch in einem großen Java-Spring-Projekt können einzelne Features oder Klassen in Kotlin geschrieben oder in der Webanwendung einzelne Controller oder Datenklassen ergänzt werden. Bei den Beispielen benutze ich wenig Framework-Code und fokussiere mich auf die Umsetzung der Logik in Kotlin. Wenn, dann wird vor allem Android verwendet.

Generell ist es aber gut möglich, auch sein favorisiertes JavaScript- oder TypeScript-Framework weiterzunutzen. Je elaborierter die Build- und Deployment-Prozesse jedoch sind, desto weniger kann ich als App-Entwickler Details der Umsetzung beleuchten.

Darüber hinaus wagt das Kapitel 11 einen Ausflug in die Welt der Cross-Plattform-Entwicklung. Das ist wichtig für alle, deren Code in Zukunft aus einer Kotlin-Codebasis nicht nur auf Android-Smartphones **oder** im Browser laufen soll.

Zielgruppe

Dieses Buch ist für alle geschrieben, die sich für die Entwicklung mit Kotlin interessieren und die bereits grundsätzliche Kenntnisse in der Softwareentwicklung haben.

Dazu gehört zum Beispiel, dass Sie grundlegende Quelltext-Elemente wie Variablen, Konstanten, Kontrollstrukturen und Funktionen kennen und Ihnen Begriffe aus der objektorientierten Entwicklung wie Klasse, Objekt und Instanz geläufig sind. Programmierparadigmen und Muster der Softwareentwicklung werden ebenfalls nicht ausführlich erklärt, sondern vor allem dort erwähnt, wo es zum Verständnis der Unterschiede zwischen Kotlin und seinen Alternativen dient. Einblicke in die Programmiersprache Java helfen, sind aber nicht zwingend erforderlich.

Die meisten Entwickler sammeln ihre ersten Programmiererfahrungen in Kotlin nicht als Einsteiger in die Softwareentwicklung. Sie gehen mit Erwartungen aus bekannten Sprachen oder sogar mit existierendem Code an die Arbeit.

Mit Kotlin lässt sich in dieser Situation besonders gut arbeiten, denn es erfordert für viele Entwickler (Java, Android) keinen Umstieg auf eine andere Entwicklungsumgebung und verträgt sich mit bestehenden Infrastrukturen. Durch die hohe Flexibilität im Bereich der Interoperabilität ist es eine gute Strategie, in bestehendem Code einfach einmal einen konkreten Anwendungsfall zu ergänzen oder einzelne Klassen zu ersetzen. Damit können Entwicklungsteams eine informierte Entscheidung treffen, ohne zwangsläufig vor einem kompletten Neuanfang zu stehen.

Aufbau des Buchs

Der Aufbau dieses Buchs folgt meinen Erfahrungen als Hands-On-Entwickler und Geschäftsführer einer Agentur, in der Softwareentwickler verschiedener Erfahrungsstufen arbeiten.

In unserem Team haben alle Kotlin auf der Basis von Android kennengelernt. Sie waren und sind erfahrene Java-Entwickler und Experten in der Umsetzung von Apps. Wichtige Muster aus der App-Entwicklung und auch die Best Practices in Android waren also von vorneherein bekannt. Folgerichtig handelte es sich bei den ersten Experimenten denn auch eher um Java ohne Semikolons. Auf Basis einer fundierten Kenntnis der APIs und Tools für die Android-Entwicklung hat sich erst im Verlauf der Projekte ein tieferes Verständnis für die weitergehenden Möglichkeiten von Kotlin entwickelt. Das scheint mir typisch für eine Sprache, die entwickelt wurde, um den Unzulänglichkeiten einer existierenden Vorgängerin zu begegnen. Ich werde deshalb auf den folgenden Seiten ähnlich vorgehen.

In **Kapitel 1** gebe ich zunächst einen Überblick über die Eigenschaften und Vorteile von Kotlin, seine Konzepten und die positiven Effekte auf die eigene Codebasis.

In **Kapitel 2** erläutere ich kurz die zur Verfügung stehenden Tools, die ich für die ersten Schritte empfehle.

Kapitel 3 enthält kommentierte Beispiele, um zunächst ein Gefühl für die Sprache zu geben und ihre Möglichkeiten zu erkunden. Für diejenigen Leser, die bereits Vorkenntnisse in Android oder zum Beispiel Java haben, wird dieser Einstieg vielleicht für die ersten eigenen Versuche ausreichen. Einsteiger können dieses Kapitel auch erst mal überspringen. In den darauffolgenden Kapiteln folgt die systematische Betrachtung der einzelnen Sprachelemente.

Kapitel 4 stellt die grundlegende Syntax von Kotlin in den Mittelpunkt, und konzentriert sich auf die Grundstrukturen von Programmen und die Datentypen.

Tipp: Schnelleinstieg mit Kotlin KOANS

Wer direkt sehen möchte, wie Kotlin funktioniert, dem sei schon an dieser Stelle `https://kotlinlang.org` empfohlen. Unter dem Menüpunkt LEARN finden sich dort die sogenannten *Kotlin Koans* – kleine Übungen, die direkt im Browser kompiliert und getestet werden. Darüber hinaus zeigen sie auch, wie man von Anfang an mit Tests sinnvolle Validierungen der eigenen Überlegungen durchführen kann. Insofern ein Must-Read (and try) für den angehenden Kotlin-Entwickler.

Auch die Beispiele aus dem Buch können aus dem GitHub-Repository dorthin kopiert, ausprobiert und variiert werden.

In **Kapitel 5** werden die Kontrollstrukturen vorgestellt. Es geht also um Verzweigen und Wiederholen – und weitere daran angelehnte Sprachfeatures.

Das **Kapitel 6** widmet sich den Funktionen und ihren verschiedenen Sonderformen. Ich werde dort eine längere Zeit mit höheren Funktionen und Lambdas verbringen, aber auch die Grundlagen von Funktionen ausführlich erklären.

Darauf folgt in **Kapitel 7** ein ausführlicher Block zu den objektorientierten Anteilen von Kotlin. Hier geht es um alle Besonderheiten von Klassen, Objekten und Funktionen im Zusammenspiel. Dafür wird neben einzelnen Code-Schnipseln ein erstes Projekt vorgestellt.

Dieses Projekt beinhaltet eine Expression-Bibliothek, die in der Lage ist, mathematische Ausdrücke zu interpretieren und auszurechnen. Bislang liegt dieser Open-Source-Code in Objective-C, Swift und Java vor, in diesem Kapitel entsteht eine Kotlin-Version sowie eine Kommandozeilen-Anwendung, die sie nutzbar macht.

Kapitel 8 ist dem Thema automatische Tests in Kotlin gewidmet. Für die Nutzung der Test-Frameworks werden die Inhalte der vorherigen Kapitel als Handwerkszeug benötigt. Die Tests selbst beziehen sich vor allem auf das Testen von Funktionen und Klassen.

Kapitel 9 steigt tiefer in die komplexeren Konzepte von Kotlin ein, die auf dieser Basis aufbauen. Thema ist hier die Organisation von größeren Projekten durch Architekturmuster. Die Nebenläufigkeit mit Kotlin-Coroutines nimmt einen guten Teil des Kapitels ein und last but not least bespreche ich die Veränderungen im Denken und im Code, die sich aus der Kombination von funktionalem und objektorientiertem Stil ergeben.

Kapitel 10 ist Android gewidmet – hier entsteht eine vollständige App in Kotlin. Auch wenn das Kapitel mit den Grundlagen beginnt, ist es hier nicht möglich, alle Aspekte und Bibliotheken von Android zu behandeln. Der Schwerpunkt liegt auf moderner Architektur und den Anteilen, die sich durch Kotlin besser umsetzen lassen.

Kotlin ist seit der Version 1.2 in Bezug auf Interoperabilität einen Schritt weitergekommen und bietet auch Optionen zum Kompilieren in JavaScript und nativen iOS-Code. Dazu mehr in **Kapitel 11**.

Downloads zum Buch

Im Repository `https://github.com/kotlin-karlszwillus` finden sich Android-Beispiele, die Kotlin-Portierung der Expression-Bibliothek und die Listings aus den anderen Kapiteln zum Ausprobieren, mit Sourcecode-Dokumentation.

Wer mit GitHub nicht vertraut ist, kann den Großteil der Inhalte auch über die Website des Verlags herunterladen: `www.mitp.de/853`

Über die Projekte in diesem Buch

Es gibt im Buch kein durchgehendes Projekt, mit dem alle verschiedenen Bereiche von Kotlin gestreift werden. Vielmehr habe ich aus der Praxis verschiedene Versatzstücke aus Projekten herausgelöst, die einzelne Schritte und Techniken der jeweiligen Kapitel illustrieren. So können Sie sich auch einzelne Themen im Buch ansehen, ohne sich in ein großes Beispielprojekt eindenken zu müssen.

Am Anfang des Buchs stehen noch die einzelnen Anweisungen und Ausdrücke im Mittelpunkt, hier handelt es sich um einfach gewählte Beispiele.

Damit ich in der Lage bin, auch komplexere Aufgaben zu zeigen, benutze ich eine Open-Source-Bibliothek, die ich einmal von der Kommandozeile und später auch in einer App verwende. Sie stellt eine vollwertige Sprache zum Lösen von Ausdrü-

cken innerhalb von Formularen oder anderen Anwendungen zur Verfügung. Ihr Name ist NRExpressionLib, sie stammt vom iOS-Entwickler Nikolai Ruhe aus Berlin – wir haben in mehreren Projekten zusammengearbeitet. Da unsere Firma sie bereits von Swift nach Java portiert hat, gehe ich nun einen Schritt weiter und konvertiere sie in die Sprache Kotlin. Diese Aufgabe ist kein klassischer Einstiegspunkt, doch die Konvertierung, auch mithilfe der automatischen Konvertierungsfunktion, führte auch mich zu einigen interessanten Herausforderungen, an denen sich zum Beispiel die Inkompatibilitäten zwischen der Handhabung von *Generics* und den strengen Regeln für Typen bei Kotlin zeigen lassen.

Ein zweites Projekt stammt aus der Praxis und bildet eine Bibliothek von Meditationsübungen ab. Aufgaben wie Navigation und die Darstellung von Listen kann ich anhand dieser Listings einfacher demonstrieren. Diese App steht nicht als vollständiges Codebeispiel zur Verfügung, sondern lediglich in Auszügen, bzw. einzelnen Komponenten, die ausprobiert werden können. Sie ist aber im Play Store als »Key Meditation« kostenlos verfügbar.

Nicht alle Listings im Buch stammen aus diesen beiden Projekten. Einige stehen einfach für sich oder kommen in leicht abgewandelter Form aus dem Arbeitsalltag eines App-Entwicklers.

- Die Codebeispiele im Buch sind daher Auszüge aus längeren Listings, die auf der Seite des Verlags unter `www.mitp.de/853` und im öffentlichen Repository unter `https://github.com/kotlin-karlszwillus` zur Verfügung stehen.
- Auf eine vollständige Befehlsreferenz habe ich zugunsten der bekannten Online-Quellen verzichtet, der Anhang A enthält aber kürzere Übersichten zu den Standardpackages, die sich auch plattformübergreifend verwenden lassen.
- Eine Mischung von Deutsch und Englisch lässt sich nicht ganz vermeiden. Bei Fachbegriffen gebe ich den englischen Versionen den Vorzug und verwende eingedeutschte oder übersetzte Bezeichner nur dort, wo sie vergleichbar allgemeingültig sind.

Was nicht in diesem Buch steht

Jeder Autor entscheidet über Schwerpunkte und Inhalte eines Buchs, doch für den einen oder anderen mag es wichtiger sein, welche Themen **nicht** behandelt werden. Für mich sind dies in aller Kürze: keine RX-basierte Entwicklung, so gut wie keine experimentellen Features (mit Ausnahmen bei Coroutines und Contracts), kein *Kotlin/Native* und nur wenig über Multiplattform-Projekte.

Kotlin ist eine Sprache, deren Entwicklung noch in vollem Gang ist. Dabei werden vor allem auch im Bereich der Interoperabilität immer wieder neue Welten erschlossen. Mit der Version 1.2 ist *Kotlin/Native* dazugekommen, ein Setup für die Anbindung verschiedener Umgebungen, die auf C und moderneren Ableitungen

davon basieren. Damit ist theoretisch ein Einstieg in die Entwicklung von iOS- und Desktop-Varianten möglich. Hierfür sind zum jetzigen Zeitpunkt noch sehr viel Wissen und Basteltalent in den ursprünglichen Sprachen notwendig, das mir fehlt. Ich freue mich aber auf weiteren Fortschritt in diesem Bereich.

Das zweite Thema, das lediglich gestreift wird, sind Multiplattform-Projekte. Tatsächlich ist ein Projekt in einem solchen Szenario umgesetzt, doch die Unterstützung ist auch in der Version 1.3 noch als experimentell gekennzeichnet. Aus diesem Grund habe ich mich entschieden, auch diesen Teil nur anzureißen und für eine spätere Ergänzung aufzuheben.

Zuletzt habe ich persönlich keine starke Zuneigung zur reaktiven Programmierung mit *RxJava* oder *RxKotlin*, auch wenn sie ihre Vorteile hat und auch regelmäßig Teil von datenintensiven Projekten ist. In der vorliegenden Ausgabe war mir der Fokus auf den Kotlin-eigenen Mechanismen im Bereich Coroutines, Observer und Mapping wichtiger. Für Android habe ich die Vorstellung von LiveData und ViewModel vorgezogen, die in dem gleichen Bereich im Einsatz sind, der heute häufig von Rx-Komponenten abgedeckt wird.

Weiterentwicklung von Kotlin

Ein wichtiges Anliegen habe ich in Bezug auf Aktualität und Updates der Sprache: Selbstverständlich werden sich von der Erstellung dieses Manuskripts bis zur Veröffentlichung Dinge ändern.

Insbesondere im Bereich von Bibliotheken verändern sich Möglichkeiten und APIs schnell und häufig. Zwar habe ich darauf geachtet, welche Veränderungen zur Erstellung des Manuskripts bereits absehbar waren, doch bereits mit dem nächsten größeren Update können sich auch grundlegende Mechanismen ändern.

Als Beispiel seien hier die Coroutines genannt, die mit der Version 1.2 eingeführt wurden und ein Ersatz für Java-basiertes Multithreading sind. Daher möchte dieses Buch vor allem die Grundlagen vermitteln, mit denen Sie funktionierenden, gut strukturierten und durchdachten Kotlin-Code erstellen können.

Über den Autor

Meine eigene Ausbildung nennt sich Medienwirt – ein Diplomstudium, das mir weitreichende Freiheiten und Interpretationsmöglichkeiten gewährt hat. So konnte ich die Programmierung, bis dahin Hobby und Schulfach, auch an der Universität weiterverfolgen, dann unter dem Stichwort »Neue Medien«. Heute arbeite ich als Geschäftsführer der Agentur *Karlmax Berlin* an verschiedenen Kundenprojekten im Bereich Apps.

Meine tägliche Arbeit hat bis vor Kurzem auch noch stark in den Projekten stattgefunden, sodass ich auch mit Kotlin dort meinen ersten Kontaktpunkte hatte.

Unsere Agentur hatte das Glück, dass wir mit zwei Kunden zum Zeitpunkt der Bekanntgabe von Kotlin als offizieller Entwicklungssprache für Android bereits über ein Jahr an Projekten mit Kotlin arbeiten durften. Die positiven Einflüsse auf eine Android-App kannten wir so bereits durch konkrete Projekte. Nicht zuletzt mit dieser Erfahrung im Rücken möchte ich alle Entwickler aus dem Java- und Android-Umfeld einladen, sich mit Kotlin zu beschäftigen.

Kapitel 2

Arbeiten mit Kotlin

2.1 Entwicklungsumgebung einrichten

Um die ersten Schritte mit den Beispielen der kommenden Kapitel ausführen zu können, ist jetzt der Zeitpunkt, eine entsprechende Umgebung zu konfigurieren, um Kotlin auszuprobieren. Meine Empfehlung ist es, hierzu auf eine der frei verfügbaren Entwicklungsumgebungen zu setzen.

Da die ersten Schritte in einem neuen Tool eventuell nicht ganz selbsterklärend sind, stelle ich die beiden wichtigsten Umgebungen vor und zeige sie bis zum ersten minimalen funktionierenden Beispiel. Nun ändern sich Softwareversionen in der Regel schneller als Bücher, doch die grundlegenden Fragen der Organisation haben höchstwahrscheinlich noch eine Weile Bestand, auch wenn die Screens sich bis zur Drucklegung eventuell in Details schon wieder geändert haben.

Es besteht die Wahl zwischen zwei Varianten:

- *IntelliJ* Community Edition: Für alle, deren Fokus nicht direkt auf Android liegt, oder die eventuell auch komplexere Cross-Plattform-Projekte ausprobieren möchten
- *Android Studio*: Für diejenigen, deren Projekte auf Android beschränkt sind

2.1.1 IntelliJ

Nach der Auswahl PROJEKT > NEU stehen verschiedene Optionen für Kotlin-Projekte zur Verfügung.

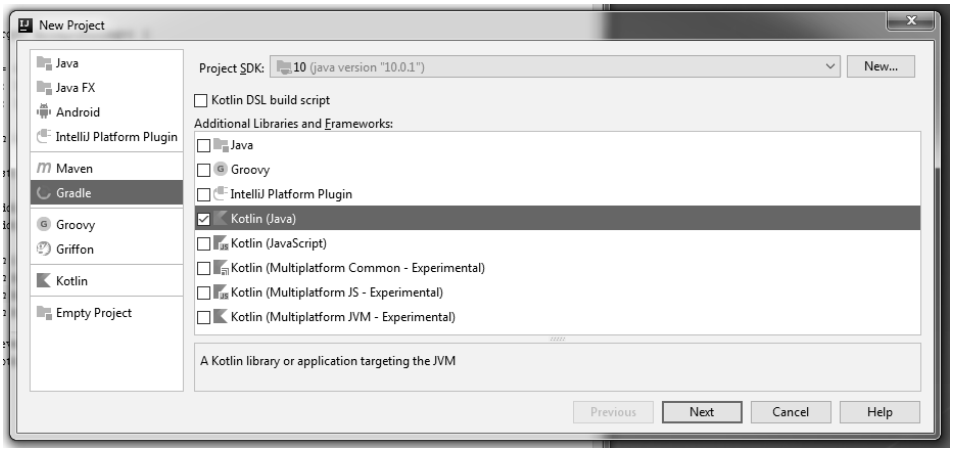

Abb. 2.1: Projektassistent von IntelliJ

Für die Beispielprojekte in den ersten Kapiteln reicht in der Regel das einfache KOTLIN(JAVA)-Projekt aus. In den späteren Kapiteln wird dann auch die Variante für JavaScript, beziehungsweise ein Cross-Plattform-Projekt, verwendet. Wie an den Bezeichnern bereits ersichtlich ist, handelt es sich zum Teil um experimentelle Features. Diese sind nutzbar, es ist jedoch nicht garantiert, dass nach einem Update der Sprache oder der Tools alles noch läuft wie zuvor.

Im nächsten Schritt muss das Kind einen Namen bekommen. Wer Maven (ein anderes Tool zur Buildorganisation, vergleichbar mit Gradle) nutzt, wird die Einteilung in *Artifact-ID*, *Group-ID* und *Version* kennen. Die Group-ID kann leer bleiben, die Artifact-ID sollte den Projektnamen erhalten, und die Version kann auch zunächst auf dem Default-Wert stehen bleiben.

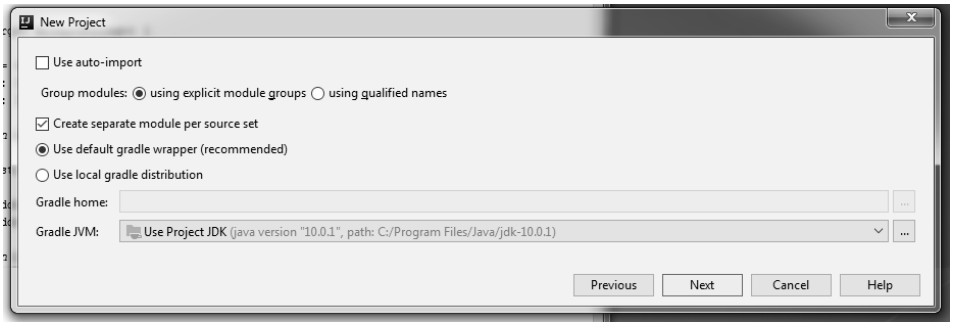

Abb. 2.2: Projektdetails

Die Projekte in IntelliJ werden standardmäßig mit Gradle konfiguriert und gebaut. Daher ist dieser Schritt wichtig. Im Normalfall sollte hier der Gradle-Wrapper aus der Basisinstallation gute Dienste tun.

Es kann im Einzelfall vorkommen, dass sich der aktuelle Wrapper nicht mit der Version von Plug-ins oder auch Java verträgt, in diesem Fall kann Gradle auch manuell installiert werden. Sollte das Tool Update-Bedarf anmelden, empfiehlt es sich in der Regel, diesem Rat zu folgen.

Tipp: Update Gradle-Wrapper

Falls sich Fehlermeldungen beim Erstellen des ersten Projekts zeigen, die anzeigen, dass Komponenten nicht gefunden werden oder der Gradle Sync nicht erfolgreich durchgeführt wird, empfiehlt sich ein Check der Version(en) von Java, IntelliJ, Kotlin-Plug-in für IntelliJ und Gradle-Wrapper. Sollte dieser nicht aktuell sein, ist ein Update möglich:

```
./gradlew wrapper --gradle-version 4.8.1 (aktuelle Version)
```

Mehr Informationen zu Gradle und Stand-Alone-Downloads gibt es unter folgender Adresse:

`https://gradle.org/`

Abb. 2.3: Positives Feedback vom gradle in einem leeren Projekt

Mit diesem leeren Projekt kann es im Prinzip losgehen.

Als Nächstes müssen noch grundlegende Verzeichnisse angelegt werden (oder die Option NEUE MODULE mit leerem Source-Verzeichnis anlegen und in den Settings anhaken.)

Die Ordnerstruktur sollte folgenden Aufbau haben: *src/main/kotlin*.

Nun ist es an der Zeit, darin die erste Datei anzulegen. Passend zum Projekt soll sie den Namen `firstrun.kt` tragen. Diese Datei benötigt ein wenig »overhead«, damit die Ausführung leicht möglich ist. Die Anweisungen machen die Kotlin-Anwendung kompatibel zur JVM. Das Ergebnis dieser Schritte sollte dann so aussehen:

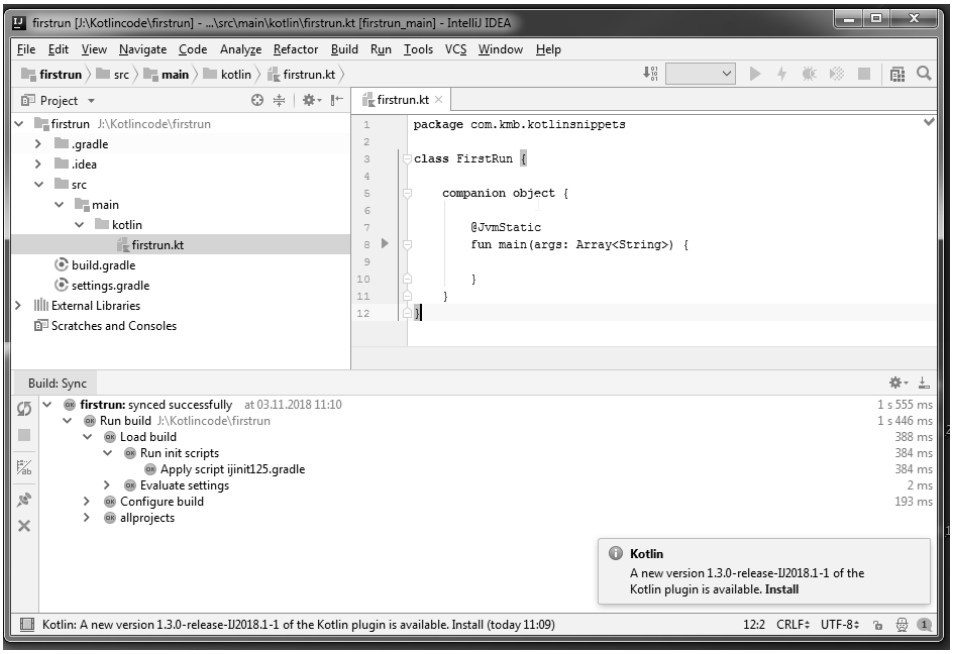

Abb. 2.4: Projektbildschirm mit Struktur (links), Code (rechts) und Build-Fenster (unten)

Der letzte Schritt ist dann, eine *Run-Configuration* zu erstellen, damit der Play-Button zum Ausprobieren des Codes glühen kann.

Dazu wählen Sie in dem leeren Dropdown-Menü neben dem ausgegrauten Play-Button (vorheriger Screen rechts oben) den Punkt EDIT CONFIGURATIONS ... oder KONFIGURATIONEN BEARBEITEN .... Dort folgt mit dem Klick auf das PLUS das Auswahlmenü für die Konfigurationstypen.

Hier wählen Sie KOTLIN aus, um die grundlegenden Einstellungen für das erste Projekt zu öffnen (s. Abbildung 2.5)

Bei diesem letzten Schritt müssen Sie einmal die gerade erzeugte Klasse in dem hervorgehobenen Bereich USE CLASSPATH OF SUBMODULE auswählen.

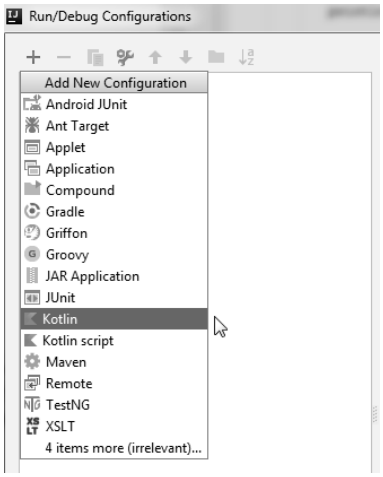

Abb. 2.5: Build-Konfigurationen

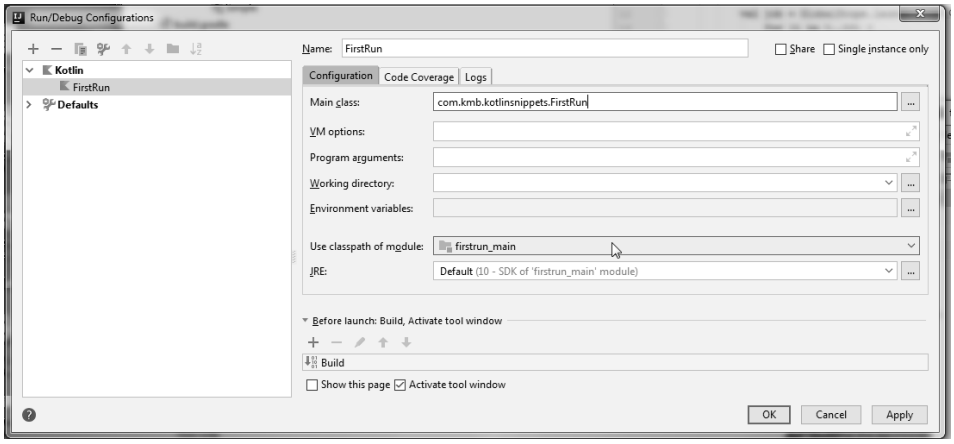

Abb. 2.6: Build-Konfigurationsdetails

Im folgenden Kapitel folgt Inhalt, mit dem die `main`-Methode gefüllt und ausprobiert werden kann.

2.1.2 Android Studio

Mit dem Projektassistenten legen Sie auch in Android Studio eine leere Applikation an. Dieser Schritt generiert bereits ersten Kotlin-Code und ein leeres Layout, in dem Sie direkt arbeiten können und das auf dem Simulator oder dem eigenen Handy einen App Screen erzeugen sollte.

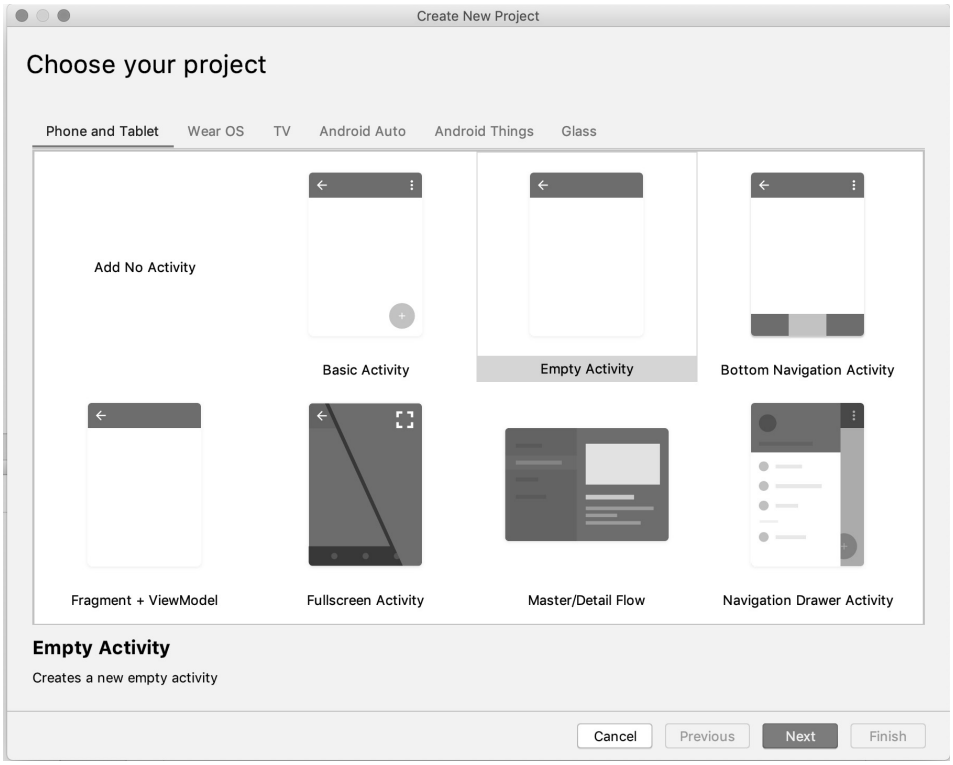

Abb. 2.7: Projektassistent Android Studio

Im ersten Schritt scheint das Template EMPTY ACTIVITY zu passen, da es eine funktionierende Grundlage schafft, ohne viel Android-Infrastruktur zu erstellen, die in den ersten Schritten nicht notwendig ist. Wer sich mit Android auskennt, kann selbstverständlich auch mit einer ANDROID TV APP oder einer FRAGMENT VIEWMODEL-Variante beginnen.

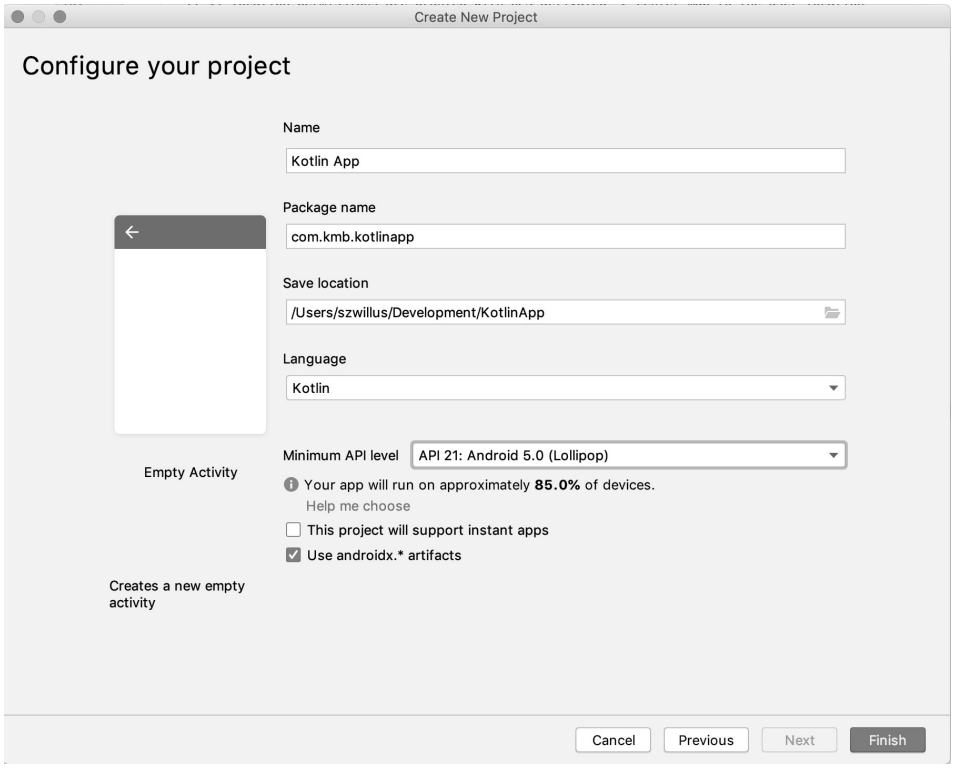

Abb. 2.8: Projektdetails konfigurieren

Im nächsten Schritt legen Sie die Details fest, eventuell fällt hier auch erneute Downloadzeit für die richtige Android-Version an. Im Beispiel habe ich mich für eine recht alte API-Level entschieden, in einem Testprojekt kann das gerne aktueller sein. Der Name der App kann frei gewählt werden, der Package-Name muss den Konventionen für Java-Packages folgen und darf nur Kleinbuchstaben ohne Sonderzeichen enthalten.

Das Resultat ist eine Minimalapp im Code, die auch direkt zum Ausführen konfiguriert ist.

Die ersten Beispiele in diesem Buch gehen davon aus, dass der Code auf der Konsole ausgeführt wird. Es ist aber kein Problem, die Beispiele leicht anzupassen und sie dann entweder aus der Konsole von Android zu sehen oder aber in einem `TextView` und damit auch direkt in einer App auszugeben.

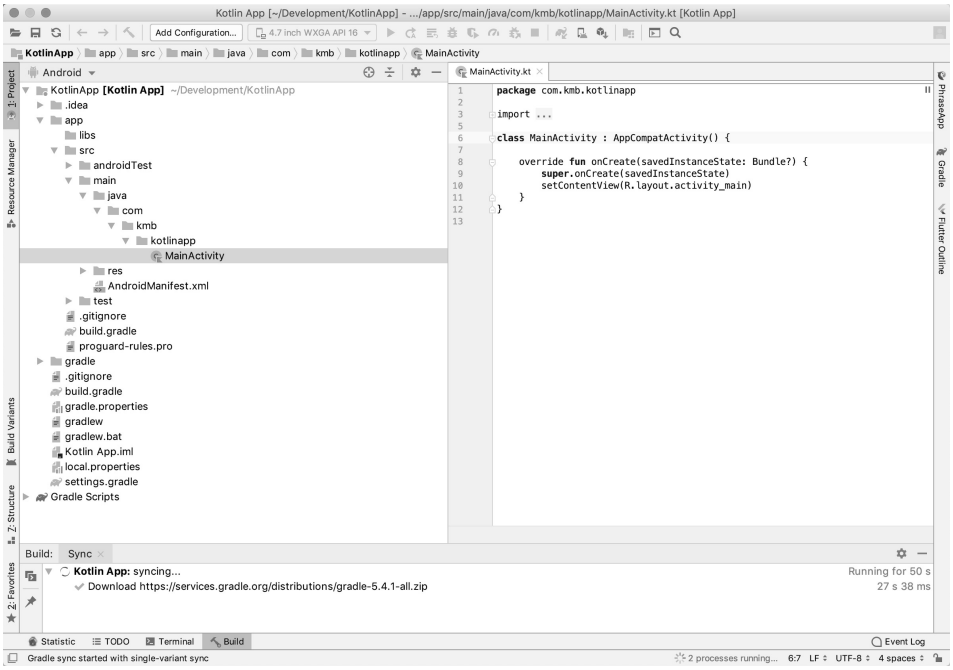

Abb. 2.9: Hauptfenster Android Studio mit Projektdaten

Hinweis: Nutzung mit einem Android-Projekt

Um die einfache Konsolenausgabe zu erreichen, die in den Codebeispielen mit `print("Irgendwas")` gemacht wird, kann in einer Android-App `System.out.print("Irgendwas")` verwendet werden – bzw. die Variante mit `println`.

Um Ausgaben mit dem `Textview` in einer App zu machen, muss dieser einmal in der Applikation aus der Layout-Datei gelesen werden (oder mittels *kotlin-android-extensions* direkt importiert) und kann dann mit `textView.text = "Irgendwas"` genutzt werden, um die Ausgabe zu machen.

Führt man die App einmal aus und lässt sie sich anzeigen, so grüßen die Worte »Hello world« in einem ansonsten leeren App-Screen mit dem Titel *Kotlin App*.

2.2 Tools

Kotlin als Programmiersprache ist einigermaßen Tool-agnostisch. Sie können in einem beliebigen Text-Editor entwickeln. Größeren Komfort und sinnvolle Unterstützung bietet jedoch die integrierte Entwicklungsumgebung (IDE) IntelliJ oder das davon abgeleitete Android Studio.

Von meiner Seite gibt es daher auch die klare Empfehlung, mit der IntelliJ-Community-Edition für die ersten Beispiele zu arbeiten. Für Entwickler von Android-Apps gibt es wenig Gründe, nicht in Android Studio weiterzumachen und gegebenenfalls die entsprechende Kotlin-Unterstützung zu aktivieren.

Um kleine Code-Schnipsel einmal direkt in Aktion zu sehen, habe ich außerdem immer wieder auf die Online-Compiler unter `https://try.kotlinlang.org` zurückgegriffen.

2.2.1 Kotlin-Compiler

Die grundlegendste Möglichkeit, Kotlin zu nutzen, ist der Kotlin-Compiler auf der Kommandozeile. Der Compiler wird dabei verwendet, um den bestehenden Code mit der Kotlin-Runtime zu bündeln und dann in ein ausführbares JAR zu konvertieren, welches mit einer Java-Umgebung lauffähig ist.

Darüber hinaus kann der Kotlin-Compiler auch als interaktive Shell oder zum Scripting verwendet werden.

Tipp

Zum Umgang mit dem Kotlin-Compiler und seiner Installation:
`https://kotlinlang.org/docs/tutorials/command-line.html`

Die ersten Code-Beispiele kann man theoretisch auf diese Art ausführen, es ist aber an sich nicht zu empfehlen, da es keinerlei Unterstützung durch eine Umgebung gibt und auch das Management von Bibliotheken manuellen Aufwand bedeutet.

2.2.2 Kotlin-Compiler mit Gradle

Auf der Konsole kann auch das Buildsystem Gradle die Arbeit an größeren Projekten vereinfachen. In beiden IDEs ist es dafür auch standardmäßig eingebunden. Gradle konfiguriert und managt die Abhängigkeiten eines Programms. So benötigt schon die einfachste »Hello world!«-Anwendung mindestens die Kotlin Runtime Library. Falls man unbedingt ohne eine Entwicklungsumgebung arbeiten wollte, wäre die Empfehlung, mindestens Gradle zusätzlich zu nutzen.

Tipp Gradle lokal installieren

Unter der domain `https://gradle.org/` gibt es die Gradle Build Tools als Standalone-Software. Als nächster Schritt helfen dann die Getting Started Guides: `https://gradle.org/guides/#getting-started`.

In diesem Fall wird eine Konfigurationsdatei namens `build.gradle` erstellt, in der die Einstellungen festgehalten werden. Damit kümmert sich das Build Tool dann um den Download von Bibliotheken und Frameworks und konfiguriert den Kotlin/Java-Compiler entsprechend der gewünschten Einstellungen.

2.2.3 Integrierte Entwicklungsumgebungen/IDE

Kotlin ist als Multiplattform-Projekt nicht auf ein Tool festgelegt, aber für eine große Zahl an Umsteigern dürften die Produkte von JetBrains eine bekannte Referenz sein. Der Support im Tool (IntelliJ ab Version 15, Android Studio ab Version 3.0) entspricht dem Standard, der aus der Java-Entwicklung bekannt ist.

Dazu gehören Syntax Highlighting, Code completion und automatische Imports ebenso wie die von IntelliJ bekannten Inspections und Analysetools für den Code.

Wie es der Name schon nahelegt, ist die Android-Studio IDE auf Android-Apps festgelegt, weitergehender Support, wie zum Beispiel Projekt-Templates für eine JS-Anwendung, fehlet hier. Umgekehrt ist das allgemeine IntelliJ-Bundle mit allen notwendigen Komponenten für die komplette Android-Entwicklung ausgestattet.

IntelliJ erlaubt auch die Entwicklung von Android-Anwendungen und zusätzlich auch die weiteren Verwendungen von Kotlin. Die Beispiele in diesem Buch sind größtenteils mit der IntelliJ-IDE und Android Studio entstanden. Beide IDEs sind unter bestimmten Bedingungen frei erhältlich.

Tipp: Download URLs

```
http://www.jetbrains.com/idea/download/index.html

https://developer.android.com/studio/
```

2.2.4 Konvertierung von existierendem Code

Eine sehr praktische Funktion – über beide IDE leicht zu erreichen – ist die Konvertierung von bestehendem Java-Quelltext in Kotlin. So kann man die neue Sprache von bekanntem Terrain her erobern. Bei der Umsetzung entsteht in der Regel nicht die ideale Kotlin-Lösung, aber im besten Fall schon mal kompilierender und funktionierender Code.

Doch es gibt ein paar Aspekte zu beachten, die auch nach der automatischen Code-Konvertierung noch notwendig sind. Zum einen tut sich die Konvertierung mit komplexen `if/switch`-Anweisungen schwer und erzeugt bisweilen duplizierten Code.

Zum anderen wird die Konvertierung alle Unsauberkeiten in Bezug auf den Umgang mit Null-Sicherheit oder Typen aufdecken und damit unter Umständen

auf relativ ungünstige bzw. teure Strukturen konvertieren, die besser manuell korrigiert werden. Zum Teil kam es hier bei meiner Arbeit an Projekten für das Buch auch zu Fehlern, die entweder nicht kompilierte oder aber Laufzeitfehler zeigten.

Last but not least entsteht auch im besten Fall Java-Code in Kotlin-Notierung. Die Paradigmen funktionaler Parameter werden also eher wenig bis gar nicht umgesetzt. Die Konvertierungsroutine erzeugt zum Beispiel veränderliche Variablen, die in Kotlin zum Beispiel besser als `immutable` deklariert werden sollten.

Stichwortverzeichnis

G

H

I

J

L

M

N

O

T