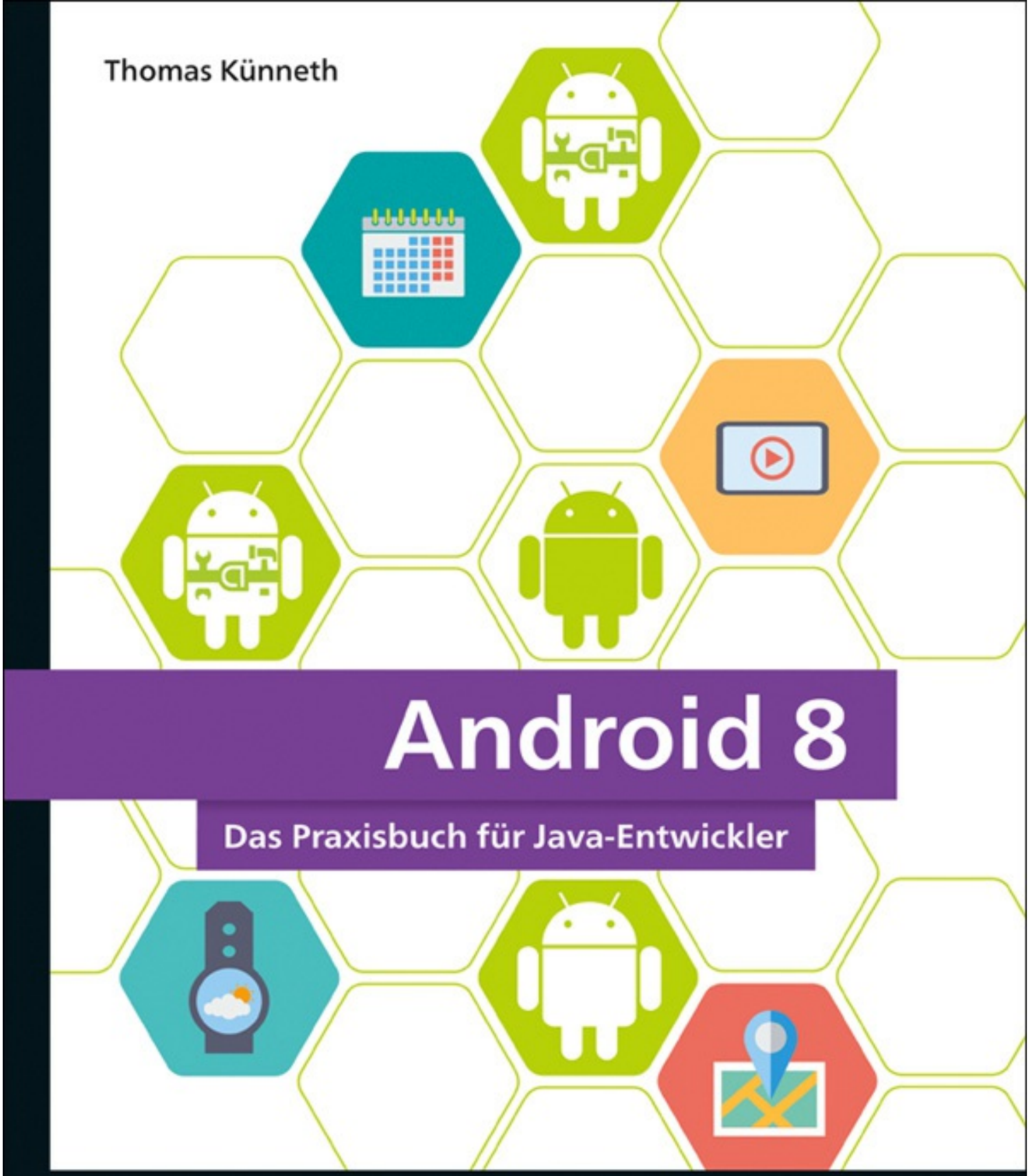
Thomas Künneth



Android 8

Das Praxisbuch für Java-Entwickler

- Professionelle Apps für Smartphone, Tablet und Watch
- Von der Idee bis zur Veröffentlichung in Google Play
- Multimedia, Bluetooth, Kamera, GPS, Kalender, GUIs, Multitasking, Android Wear u. v. m.

Mit über 70 Beispiel-Apps zum Download

Rheinwerk Computing

Android Runtime (ART) übersetzte Java-Anwendungen schon während der Installation in Maschinensprache, was in vielen Situationen zu besserer Performance führte und zudem die Nutzung vieler moderner Konzepte des Linux-Kerns ermöglichte.

Die im März 2015 veröffentlichte Version 5.1 (API-Level 22) unterstützte endlich einen zweiten Slot für SIM-Karten. Bis dahin hatten die Gerätehersteller selbst für die Integration einer zweiten SIM-Karte sorgen müssen. Ebenfalls neu war die Sprachübertragung in verbesserter Qualität (*HD Voice*), sofern ein Mobiltelefon diese vorsah. Ein verbesserter Geräteschutz, Stabilitätsverbesserungen sowie die Verwaltung von Bluetooth- und WLAN-Geräten über die *Quick Settings* rundeten dieses Service-Release ab.

Android 6 (*Marshmallow*) brachte ein neues Berechtigungssystem und erlaubte Benutzern, Apps gezielt den Zugriff auf bestimmte Ressourcen zu entziehen. Ausführliche Hinweise hierzu finden Sie in Kapitel 4, »Activities und Broadcast Receiver«. Neben der nativen Unterstützung von Fingerabdruckscannern kamen unter anderem der Standby-Modus *Doze* und *Google Now on Tap* sowie die Nutzbarkeit von SD-Karten als interner Speicher hinzu.

Android 7 (*Nougat*) führte unter anderem die gleichzeitige Benutzung von zwei Apps auf dem Bildschirm ein, ferner ein überarbeitetes Benachrichtigungscenter, zahlreiche *Java-8*-Features, eine zusätzliche 3D-Grafik-Bibliothek sowie die Unterstützung von Virtual-Reality-Anwendungen. Android 7.1 brachte Unterstützung für runde Icons sowie *App Shortcuts*.

Android 8

Mit *Oreo* (API-Level 26) ändert Google den Umgang mit Hintergrund-Apps. Ziel ist, dass sich die Plattform flüssiger anfühlt und weniger Strom verbraucht. Was Sie als Entwickler beachten müssen, damit sich Ihre App vorbildlich verhält, erkläre ich in Kapitel 6, »Multitasking«. Außerdem gibt es nun *Benachrichtigungskanäle*, die dem Benutzer helfen sollen, sich besser in der Flut an *Benachrichtigungen* zurechtzufinden. Android 8.1 (API-Level 27) enthält zusätzlich Programmierschnittstellen für neuronale Netze und unterstützt *Shared Memory*.

Googles teilweise beängstigend hohes Entwicklungstempo hat aus Android eine stabile und anwenderfreundliche Plattform gemacht. Ein Kernproblem konnte bis Oreo allerdings nicht zufriedenstellend gelöst werden: Es dauerte immer mehrere Monate, bis Gerätehersteller ihren Kunden aktuelle Android-Versionen zur Verfügung stellten. Nicht wenige Modelle wurden nie aktualisiert. Das ab Android

8 enthaltene *Project Treble*[3] soll dieses Problem lösen. Es trennt die sogenannte *Vendor Implementation* (hardwarespezifische Software, die direkt von den SoC-Herstellern geliefert wird) von Betriebssystem und Framework durch eine neue, standardisierte Herstellerschnittstelle ab. Früher mussten Hersteller bei einem Android-Update auch diese unterste Schicht mitliefern, da sie fest mit Android verwoben war. Dies ist nun nicht mehr nötig. Selbst wenn es von den SoC-Produzenten keine neue Software gibt, kann auf solchen Geräten dennoch eine neue Android-Version aufgespielt werden. Sie setzt einfach auf dem vorhandenen *Vendor Interface* auf. Damit das klappt, muss ein Gerät freilich einmal Android 8 mit Project Treble erhalten haben. Bei Updates auf Oreo ist dies derzeit leider optional. Nur neue Geräte müssen Treble beinhalten.

Auch wenn damit in Zukunft mehr Geräte in den Genuss neuer Android-Versionen kommen werden, lohnt es sich, regelmäßig auf der Seite *http://developer.android.com/about/dashboards/index.html* den aktuellen Verbreitungsgrad der verschiedenen Android-Versionen zu prüfen, denn erst ab einer bestimmten Anzahl von potenziellen Nutzern lohnt sich die Verwendung neuer Programmierschnittstellen oder Funktionen.

1.2 Systemarchitektur

Vielleicht fragen Sie sich, was das Wort *Plattform* im Zusammenhang mit Android bedeutet. Handelt es sich nicht einfach um ein Betriebssystem für mobile Geräte, für das Sie in Java Programme schreiben?

1.2.1 Überblick

Aus der Sicht des Endanwenders bildet Android eine mittlerweile sehr große Gruppe von mobilen und stationären Geräten, beispielsweise Smartphones, Tablets, Medienabspielgeräten, TV-Settop-Boxen und Armbanduhren. Zahlreiche Hersteller bieten Modelle in unterschiedlichsten Ausstattungsvarianten an. Neben preisgünstigen Einsteigerprodukten finden sich im Hochpreissegment Geräte mit viel Arbeitsspeicher, großen, auflösungsstarken Bildschirmen und hoher Prozessorleistung. Auf all solchen Produkten können Versionen von Android laufen. Dennoch ist Android nicht nur ein Betriebssystem, weil zu Android standardmäßig noch mehr gehört, beispielsweise ein Anwendungsstarter mit Unterstützung für Widgets, eine Kontaktdatenbank, eine Uhr mit Weckfunktion, ein Browser und ein E-Mail-Client. Sehr oft ist auch der *Play Store* enthalten, eine Anwendung zum Kaufen und Herunterladen von Apps und Medien, beispielsweise Musik, Filmen, Büchern, Zeitungen und Magazinen.

Android und Java

Ebenfalls ein Bestandteil von Android, allerdings für den Endanwender nicht sichtbar, ist die Laufzeitumgebung ART (*Android Runtime*). Sie bildet den Rahmen für die Ausführung aller Programme, die der Benutzer auf einem Android-System startet. Dies betrifft die weiter oben genannten Standardanwendungen, aber auch selbst geschriebene Programme, von denen die meisten in Java entwickelt werden. Wenn Sie mit dieser Technologie bereits Erfahrung haben, fragen Sie sich vielleicht, ob Android für die Ausführung der Programme dann nicht eine virtuelle Maschine enthalten müsste, denn schließlich wird Java-Code üblicherweise in Bytecode umgewandelt.

Android hatte von der ersten Version an eine virtuelle Maschine an Bord. Allerdings hat diese *Dalvik* genannte Komponente nie den Bytecode verstanden, der vom Standard-Java-Compiler erzeugt wird, weil Dalvik seinen eigenen Befehlssatz hat. Auch die *registerbasierte* Architektur weicht von einer klassischen Java Virtual Machine ab, die einen *Kellerautomaten* realisiert. Dennoch war `javac`

lange Zeit der erste Schritt vom Quelltext hin zur ausführbaren App. Der erzeugte Bytecode wurde von dem Tool `dx` in ein *Dalvik Executable* (*.dex*) genanntes Format umgewandelt, das von Dalvik verstanden wird. Hierin liegt der Hauptgrund, warum Android-Entwickler immer recht lange auf die Nutzung neuer Java-Sprachfeatures in ihren Apps warten mussten. Beispielsweise führen die beliebten *Lambda*-Ausdrücke zu neuen Konstrukten im Bytecode, die `dx` kennen und zu sinnvollen Dalvik-Anweisungen umformen muss.

Dalvik wurde erst mit Android 5 vollständig abgelöst. Schon in Android 4.4 war der Nachfolger ART enthalten und konnte auf Wunsch aktiviert werden. Seit Android 5, *Lollipop*, werden Apps schon während der Installation in Maschinensprache übersetzt. Android Nougat allerdings stellte ART wieder einen Just-in-time-Compiler zur Seite (Dalvik besaß zeitweise auch einen). Dieser nutzt *Code Profiling*, um das Laufzeitverhalten einer App kontinuierlich zu verbessern. Hierzu wird mithilfe von Profilen entschieden, wann welche Teile des Codes übersetzt werden: Schlüsselfunktionen einer App werden so schnell wie möglich abgearbeitet.

Eine Zeit lang hat Google eine andere Strategie bei der Umwandlung von Java-Quellcode in *.dex*-Dateien verfolgt. An die Stelle von `javac` und `dx` trat der neue Compiler *Jack*. Mit ihm kam der sehnlichst erwartete Support für eine ganze Reihe von Java-8-Sprachfeatures, unter anderem für Lambda-Ausdrücke, Methodenreferenzen und Typannotationen. Diese waren sogar unter älteren Android-Versionen nutzbar, Default- und statische Interface-Methoden sowie wiederholbare Annotationen hingegen nur ab API-Level 24. Jack musste in der Datei *build.gradle* aktiviert werden. Unglücklicherweise ließen sich Tools von Drittanbietern nicht gut in die neue Toolkette integrieren. Mit Android Studio 3 wurde Jack deshalb abgelöst. Natürlich gibt es nach wie vor Unterstützung für Java-8-Features. Wie Sie hierzu vorgehen, erkläre ich Ihnen etwas später.

Erfreuliche Erweiterungen der Syntax sind aber nur ein Teil der Neuerungen von Java 8. Auch die Standardklassenbibliothek hat eine Vielzahl neuer Pakete und Klassen erhalten, unter anderem `java.util.stream` und `java.util.function`. Dass diese auch unter Android verwendet werden können, hat mit einem fundamentalen Wechsel zu tun: Bis einschließlich Android 6 basierte die Klassenbibliothek der Plattform in weiten Teilen auf Code des Open-Source-Projekts *Apache Harmony* der *Apache Software Foundation*.

Ziel dieses im Mai 2005 angekündigten und im November 2011 beendeten Projekts war die Schaffung einer frei verfügbaren, quelloffenen Java-Implementierung einschließlich Compiler und virtueller Maschine. Die

Notwendigkeit hierfür ist schon frühzeitig, nämlich mit der Veröffentlichung von Java als Open Source Ende 2006, entfallen. Daher hat Harmony für Android nie große Bedeutung erlangt – außer eben durch die Nutzung seiner Klassenbibliothek durch Android.

Harmony war als Implementierung von Java 5 und 6 gedacht. Konsequenterweise fehlen alle Klassen und Pakete, die Sun und Oracle erst mit späteren Java-Versionen hinzugefügt haben. Google hatte zwar an einigen Stellen »nachgebessert«. Für die Nutzung des Java-7-Features *try-with-resources* ist beispielsweise das Interface `java. lang.AutoCloseable` nötig, das seit API-Level 19 enthalten ist. Andere Neuigkeiten hingegen wurden nie übernommen. Android 7 und 8 nutzen die Bibliothek des *OpenJDK*. Damit stehen Entwicklern endlich viele wichtige neue Bibliotheksfunktionen zur Verfügung.

Kotlin

Seit Android Studio 3.0 müssen Sie Ihre Apps nicht mehr zwingend in Java schreiben. Die Entwicklungsumgebung bietet ab dieser Version offiziell Unterstützung für die Programmiersprache *Kotlin*. Diese wurde nicht speziell für Android entwickelt. Vielmehr hat sie den Anspruch, eine moderne, schlanke Alternative zu Java zu sein. Üblicherweise wird Kotlin-Quelltext in Java-Bytecode übersetzt. Es steht aber auch ein Transpiler nach JavaScript zur Verfügung. Erfinder der Sprache ist die in Sankt Petersburg ansässige Firma *JetBrains*. Von ihr stammt auch die Basis von Android Studio, *IntelliJ IDEA*. Übrigens ist Kotlin eine russische Insel im Finnischen Meerbusen, 30 km westlich von Sankt Petersburg in der Ostsee gelegen.

Kotlin ist wie Java statisch typisiert, klassenbasiert und objektorientiert. Die Syntax ist nicht kompatibel zu Java, wenngleich viele Konstrukte vertraut wirken. Die Interoperabilität mit Java-Code spielt eine wichtige Rolle. Vorhandene Bibliotheken können deshalb ohne Aufwand angesprochen werden. Lassen Sie uns einen kurzen Blick darauf werfen, wie sich Kotlin anfühlt.

```
fun main(args: Array<String>) {
    println("Hallo Kotlin")
}
```

Listing 1.1 Ein vollständiges Kotlin-Programm

Listing 1.1 zeigt ein vollständiges, ausführbares Kotlin-Programm. Wie in Java bildet `main()` den Einstiegspunkt. Allerdings kennt die Sprache nicht nur Methoden, sondern auch *Funktionen* außerhalb einer Klasse. Der Rückgabewert