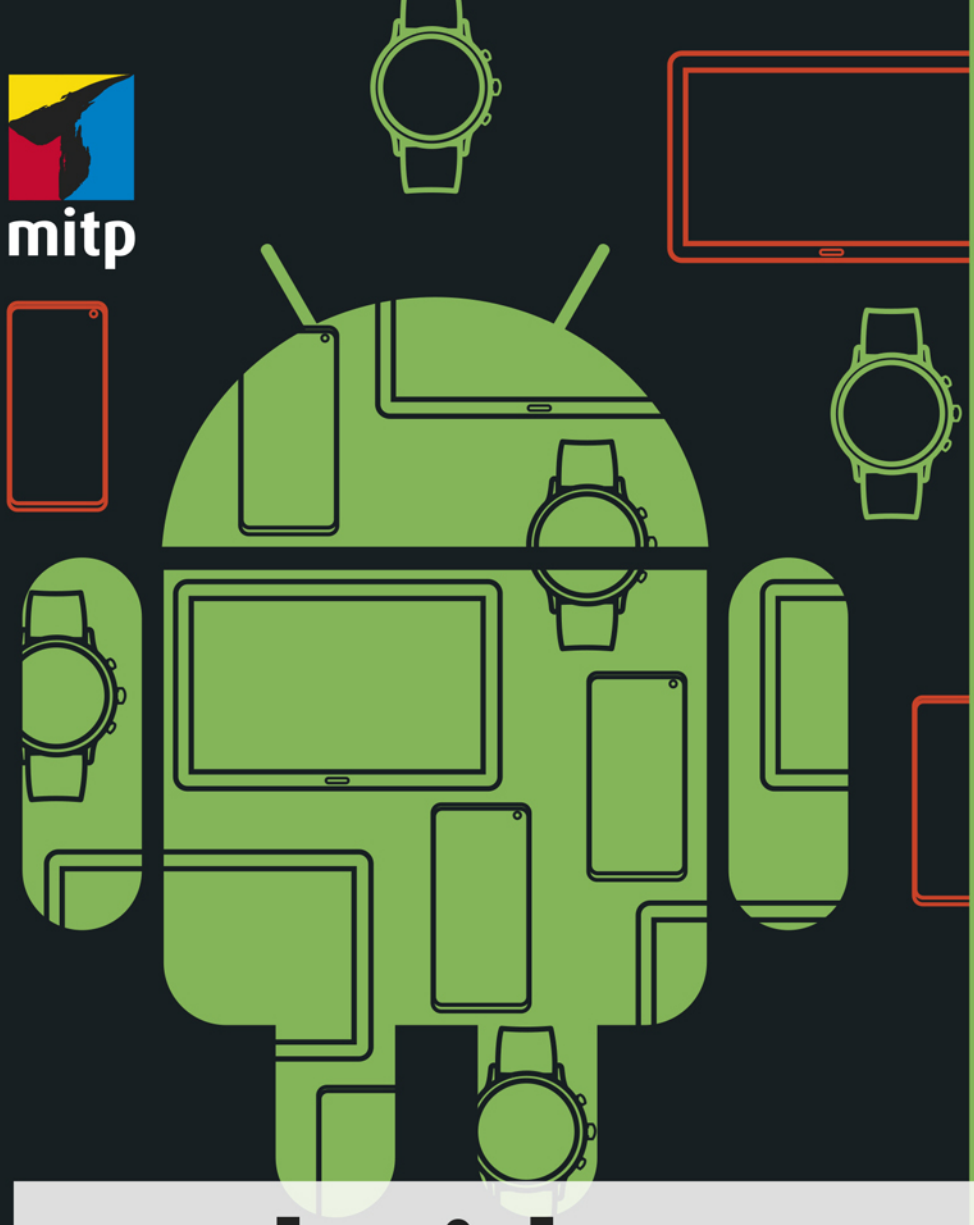

mitp

Eugen Richter

3. Auflage

Android-Apps programmieren

Professionelle App-Entwicklung mit Android Studio 4

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im mobilen Segment hat Android mittlerweile einen Marktanteil in Deutschland von über 73% (Quelle: Statista 2020). An dieser großen Verbreitung möchten Sie sicherlich teilhaben, ob als Hobby-Programmierer oder in einem Unternehmen. Mit diesem Buch finden Sie einen praktischen Einstieg in die Android-Programmierung.

1.1 An wen richtet sich dieses Buch?

Dieses Buch ist für alle geschrieben, die in die Android-Entwicklung einsteigen möchten, richtet sich aber auch an App-Programmierer, die alte Kenntnisse auffrischen wollen.

1.1.1 Voraussetzungen

Die wichtigste Voraussetzung für die Lektüre sind grundlegende Kenntnisse in einer objektorientierten Programmiersprache, im Idealfall in Java. Aber auch ohne Java-Kenntnisse (zum Beispiel als C#-Entwickler) finden Sie sich sehr schnell zurecht, wenn objektorientierte Programmierung für Sie kein Fremdwort ist.

Weiterhin ist ein gutes Verständnis des Dateiformats »XML« notwendig, da Android die Layouts und einige andere Ressourcen in XML beschreibt.

Die letzte Voraussetzung ist ein grundlegendes Verständnis von relationalen Datenbanken. Keine Angst, Sie müssen in diesem Buch keine »CREATE«-Anweisung aus dem Kopf schreiben.

1.2 Technischer Stand

Dieses Buch wurde grundlegend für die Android-Studio-Version 4.0 geschrieben, die zum Druckzeitpunkt aktuell ist. Wenn Sie eine neuere Version verwenden, werden eventuell Teile der Benutzeroberfläche anders aussehen als die Screenshots im Buch. Das grundlegende Vorgehen sollte sich aber nicht ändern. Der

Quellcode für die App aus dem Buch wird kompatibel zu der aktuellen Android-Studio-Versionen gehalten.

Aktualisierungen

Sollten neue Android-Studio-Versionen grundlegende Änderungen enthalten, so dass die Inhalte im Buch nicht mehr nachvollzogen werden können, werden korrigierte Kapitel auf der Projektseite veröffentlicht. Sollten Sie eine solche Änderung feststellen, schreiben Sie bitte an `android-buch@webducer.de`, damit ich die Korrekturen vornehmen und für alle verfügbar machen kann.

wdurl.de/ab3-projekt

Quellcode

Den Quellcode für die Aufgaben im Buch finden Sie bei dem Anbieter Bitbucket. Der Code ist in Kapitel unterteilt und führt Sie von einer Aufgabe zur nächsten – vom Ausgangscode bis zur Lösung der gestellten Aufgaben.

wdurl.de/ab3-code

1.3 Was finden Sie in diesem Buch?

Das Buch ist in Form eines praktischen Workshops aufgebaut. Es startet mit der Installation der Entwicklungsumgebung auf den wichtigsten Betriebssystemen und vermittelt anschließend durch die Entwicklung einer praxistauglichen App zahlreiche Fertigkeiten für die Android-Entwicklung:

- UI-Design mit XML (Designer und Code)
- Logik in Java
- Zugriff auf Datenbanken
- Zugriff auf Internet-Ressourcen (APIs)
- Hintergrundprozesse
- Dialoge
- Binding
- und viele andere

Am Ende entsteht eine fertige App, die in einem Store oder auf der eigenen Homepage veröffentlicht werden kann. Dabei werden alle Schritte durch den Quellcode zu dem jeweiligen Fortschritt unterstützt, den Sie auch auf der Bitbucket-Seite des Projekts zum Download finden: `https://wdurl.de/ab3-code`

1.4 Was behandelt dieses Buch nicht?

Auf der Google I/O wurde die Programmiersprache Kotlin als offiziell unterstützte Sprache angekündigt. Dieses Buch behandelt Kotlin aus folgenden Gründen in der aktuellen Auflage (noch) nicht:

- Die Sprache ist relativ neu und nur wenige Entwickler kennen diese, insbesondere außerhalb des Android-Universums. Programmierung für ein neues mobiles System und zusätzlich eine neue Programmiersprache zu lernen, wäre nicht anfängerfreundlich.
- An den meisten Universitäten und Berufsschulen wird Java als »Ausbildungssprache« unterrichtet. Damit ist auch der Einstieg in die Android-Entwicklung für die zukünftigen Entwickler in Java einfacher.
- Die meisten Beispiele im Netz (aber auch die offiziellen von Google) für Android zeigen den Java-Code. Aus diesem Grund ist es für Android-Einsteiger besser, mit Java zu starten. Ich hoffe, dass sich Kotlin mit der Zeit mehr durchsetzt, da die Sprache im Vergleich zu Java wirklich deutlichen Mehrwert bietet.
- Kennt man Android mit Java, ist der Umstieg zu Android mit Kotlin relativ einfach. Es gibt sehr gute Bücher über Kotlin, die es ermöglichen, die deutlichen Vorteile der Sprache zu erlernen und einzusetzen (z.B.: »Kotlin – Einstieg und Praxis« von Karl Szwillus – ISBN: 9783958458536).

1.5 Konventionen

1.5.1 XML-Dateien

- Dateiname: Kleinschreibung mit Worttrennung durch »_«.

  Beispiel: `activity_main.xml`

- Ressource-Namen (für IDs, Strings, Abstände usw.): »CamelCase«-Schreibweise.

  Beispiel: `<string name="LabelName">Name:</string>` oder `<TextView android:id="@+id/FirstLabel" />`

1.5.2 Java-Dateien

- Dateiname/Klassenname: »CamelCase«-Schreibweise.

  Beispiel: `DbHepler`

- Nicht private Konstanten: »UPPER Case«-Schreibweise Worttrennung durch »_«.

  Beispiel: `public final static String ID_KEY = "TimeDataIdKey";`

- Private Konstanten: »UPPER Case«-Schreibweise mit »_«-Zeichen als Präfix und Worttrennung durch »_«.

  Beispiel: `private final static int _LOADER_ID = 150;`

- Private Klassenvariablen: »pascalCase«-Schreibweise mit »_«-Zeichen als Präfix.

  Beispiel: `private DateFormat _dateTimeFormatter = null;`

- Methodennamen: »pascalCase«-Schreibweise.

  Beispiel: `private void saveEndDateTime() { ... }`

- Parameternamen und lokale Variablen: »pascalCase«-Schreibweise.

  Beispiel:

```
private void setStartDate(Calendar startDate) {
  ...
  String startDateString = _dateTimeFormatter.format(startDate.getTime());
  ...
}
```

1.6 Danksagung

Dieses Buch würde nicht da sein, wenn mich nicht einige Menschen unterstützen würden.

In erster Linie gilt mein größter Dank meiner Familie, die oft genug auf mich verzichten muss, damit ich an dem Buch arbeiten kann.

Weiterhin gilt mein Dank meiner Lektorin, Frau Bahlmann, die das Buch mit Ideen für eine bessere Strukturierung und einen optimalen Sprachgebrauch zugänglicher gemacht hat. Auch ein Dank an meine erste Lektorin, Frau Janatschek, die die Geduld hatte, zwei Jahre auf die erste Veröffentlichung zu warten, und meinen technischen Schreibstil in einen leserlichen verwandelte.

Ein weiterer Dank gilt der Münchner Volkshochschule, in der ich seit 2009 unterrichte und mein Können bei der Arbeit mit Lernwilligen verfeinern darf.

Ohne die Leser wäre dieses Buch sinnlos. An dieser Stelle möchte ich mich bei allen Lesern bedanken, die Feedback zu diesem Buch (über Amazon-Rezensionen oder direkt per E-Mail an `android-buch@webducer.de`) an mich geschrieben haben. Ich habe in dieser dritten Auflage das Feedback umgesetzt und freue mich auf weiteres.

1.7 Änderungen in der 3. Auflage

Im Vergleich zur zweiten Auflage wurde Folgendes geändert:

- Aktualisierung der Installationen unter allen Betriebssystemen auf Android Studio 4.0
- Aktualisierung aller Screenshots
- **Neu**: Nutzung von ROOM für den Zugriff auf die Datenbank
- **Entfernt**: Content Provider
- **Neu**: Nutzung von AndroidX-Bibliotheken
- **Neu**: Links zu Quellen als QR-Code
- Bessere Gliederung
- Verweise auf den Quellcode

Kapitel 3

Anlegen einer neuen App

Im Laufe der nächsten Kapitel werden wir eine einfache App entwickeln, die mit der Zeit erweitert und optimiert wird. Im Zuge der Entwicklung werden auch alle neuen Techniken und Technologien am praktischen Einsatz erklärt.

Die neue App soll Ihnen dabei helfen, Ihre eigene Arbeitszeit im Blick zu behalten. Dazu werden in diesem Kapitel folgende Funktionen umgesetzt:

- Oberfläche für die Erfassung der Zeit
  - Möglichkeit zum Start der Zeiterfassung
  - Möglichkeit zum Beenden der Zeiterfassung

3.1 Projektanlage

Zur Projektanlage starten Sie zuerst die Entwicklungsumgebung »Android Studio«.

Abb. 3.1: Startbildschirm von Android Studio

Die Entwicklungsumgebung bietet mehrere Möglichkeiten, mit der Arbeit zu beginnen:

»Start a new Android Studio project«

Hierbei wird ein neues Projekt mithilfe eines Assistenten angelegt.

»Open an existing Android Studio project«

Ein bereits vorhandenes Projekt wird von der Festplatte geöffnet. Mit diesem Punkt können Sie den mitgelieferten Quellcode laden (siehe dazu Anhang A.4 »Vorhandenen Quellcode in Android Studio öffnen« am Ende dieses Buches).

Quellcode zur App

Den Quellcode für die Aufgaben im Buch finden Sie bei dem Anbieter Bitbucket. Der Code ist in Kapitel unterteilt und führt Sie von einer Aufgabe zur nächsten – vom Ausgangscode zur Lösung der gestellten Aufgaben.

wdurl.de/ab3-code

»Check out project from Version Control«

Öffnen eines Projekts aus einer Versionsverwaltung, wie Git, Mercurial oder Subversion. Dabei wird der Source-Code aus der Versionsverwaltung ausgelesen und wenn möglich auch gleich geöffnet.

»Import project (Gradle, Eclipse ADT, etc.)«

Import eines *Nicht*-Android-Studio-Projekts. Oft wird dieser Punkt für den Import der alten Android-Projekte verwendet, die noch mit Eclipse entwickelt wurden.

»Import an Android code sample«

Es wird ein Beispielprojekt aus dem Android SDK importiert. Sie finden hier einen sehr schönen Fundus an Beispielanwendungen zu den unterschiedlichsten Themen, mit denen man viel lernen kann.

3.1.1 Ein neues Projekt starten

Korrekturen

Sollten neue Android-Studio-Versionen grundlegende Änderungen enthalten, so dass die Inhalte im Buch nicht mehr nachvollzogen werden können, werden korrigierte Kapitel auf der Projektseite veröffentlicht. Sollten Sie eine solche Änderung feststellen, schreiben Sie bitte an `android-buch@webducer.de`, damit ich die Korrekturen vornehmen und für alle verfügbar machen kann.

wdurl.de/ab3-projekt

Sie beginnen mit der Option START A NEW ANDROID STUDIO PROJECT. Daraufhin öffnet sich ein Assistent für die Anlage neuer Android-Projekte. Auf dem ersten Bildschirm wird die App-Vorlage ausgewählt.

Die Vorlagen erzeugen für oft gebrauchte Fälle ein Grundgerüst im Projekt. Viele der Vorlagen erfordern ein bestimmtes Vorwissen, um mit diesem Grundgerüst weiterarbeiten zu können. Da Sie gerade mit der Android-Entwicklung anfangen, verwenden Sie die einfachste Vorlage EMPTY ACTIVITY, die kaum Code erzeugt. Damit werden Sie die App praktisch von Grund auf neu aufbauen.

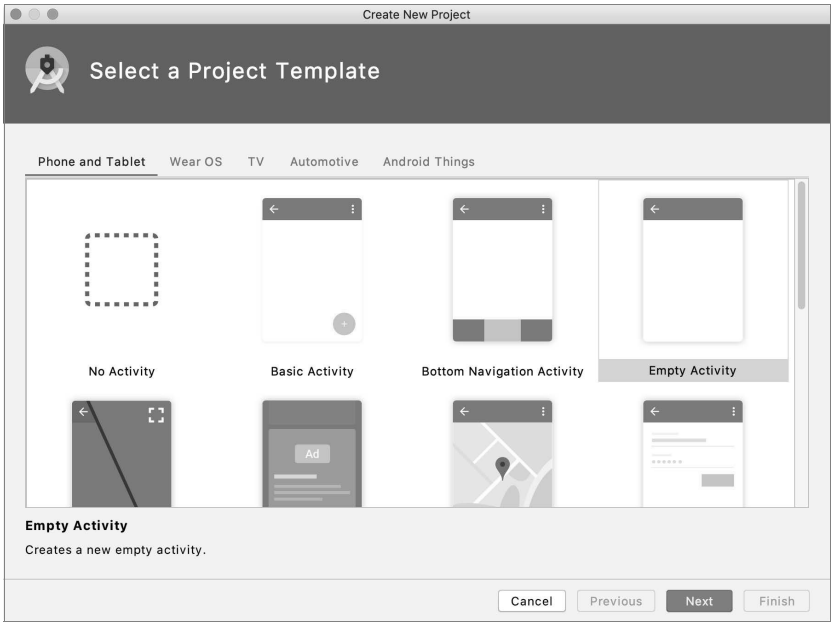

Abb. 3.2: Assistent für die Anlage neuer Projekte: Vorlage

Nach dem Klick auf NEXT gelangen Sie zum nächsten Bildschirm des Assistenten, in dem die Basisdaten der App festgelegt werden.

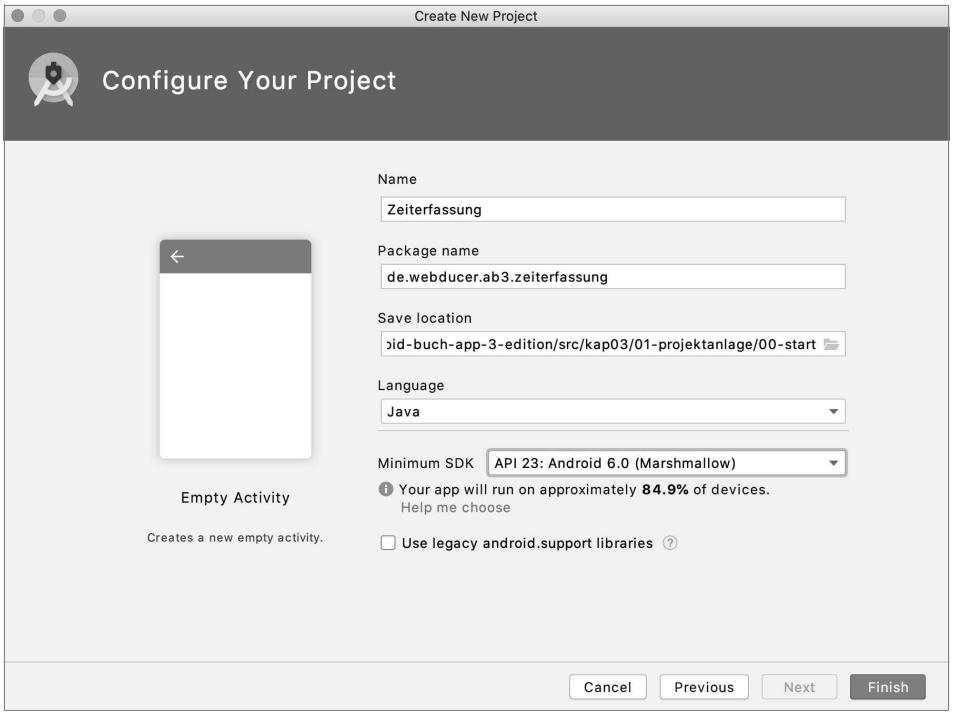

Abb. 3.3: Assistent: App Basisdaten

Unter NAME ist der Name der App einzutragen. Dieser Name erscheint im Android-Launcher (App-Übersicht). Der Wert wird als Ressource verwaltet (zu den Ressourcen kommen wir später noch) und ist übersetzbar. Tragen Sie hier den Namen der App in der Hauptsprache ein. In unserem Fall ist es `Zeiterfassung`. Die Hauptsprache ist bei uns momentan Deutsch.

Unter PACKAGE NAME wird im Normalfall der eigene Internet-Domain-Name (in der umgekehrten Schreibweise) + Name der App eingetragen. In unserem Fall geben Sie für den Package-Namen `de.webducer.ab3.zeiterfassung` ein.

Hintergrundwissen zu Paketnamen

Jede Android-App sollte einen weltweit eindeutigen Namen haben, damit es zu keinen Konflikten in den App Stores kommt. Es wird empfohlen, als Namen die

umgekehrte Schreibweise der eigenen Homepage zu nutzen, plus den Namen der App.

Im Namen dürfen keine Leer- oder Sonderzeichen, außer ».«, vorkommen, und es sollen nur Kleinbuchstaben verwendet werden. Der Grund dafür liegt in den Java-Konventionen. Der Package-Name wird im Java-Code auch als Package-Root verwendet. Die Java-Konventionen schreiben Kleinschreibung ohne Sonderzeichen vor. Die Trennung der Ebenen erfolgt nur durch einen Punkt.

Unter SAVE LOCATION können Sie den Speicherort des Projekts angeben, falls Sie es nicht im Standardverzeichnis speichern wollen.

Unter LANGUAGE können Sie die Programmiersprache (Java oder Kotlin) auswählen, in der die Logik geschrieben wird. Diese Auswahl ist nicht endgültig und kann später angepasst werden. Für unser Projekt wählen Sie hier Java.

Die alternative Sprache »Kotlin« ist eine Java-kompatible Sprache, die seit 2017 offiziell von Google für Android-Entwicklung anerkannt wurde. Mittlerweile wird diese von vielen Android-Entwicklern, und auch Google selbst, präferiert. Kotlin kann auch parallel zu Java genutzt werden, wenn man eine App, zum Beispiel von Java, migriert.

Unter MINIMUM SDK wählen Sie die kleinste Android-Version, mit der die App noch kompatibel sein soll. Android Studio zeigt zur besseren Orientierung auch gleich die Statistik, auf wie viel Prozent der weltweit auf den Google Play Store zugreifenden Geräten die neue App funktionieren würde, wenn Sie die ausgewählte als minimale Version nutzen würden. API 23 (Android 6.0) ist jetzt eine sehr gute Wahl, da man mit dieser Version zum Zeitpunkt, als dieses Buch geschrieben wurde, ca. 85% der potenziellen Benutzer erreichen würde (wenn Sie dieses Buch lesen, wahrscheinlich mehr).

Mehr zu Statistik der Verbreitung

Um ein besseres Gespür für die Verbreitung der Versionen zu entwickeln, klicken Sie auf den Link HELP ME CHOOSE, der die Verbreitung der einzelnen Versionen grafisch aufzeigt. Die Angaben sind nicht ganz aktuell, aber nicht weit von der Wahrheit entfernt. Unter dem Link finden Sie noch weitere Statistiken zu Android, wie Statistiken zu Bildschirmgrößen, OpenGL Versionen usw.

wdurl.de/ab3-and-stats

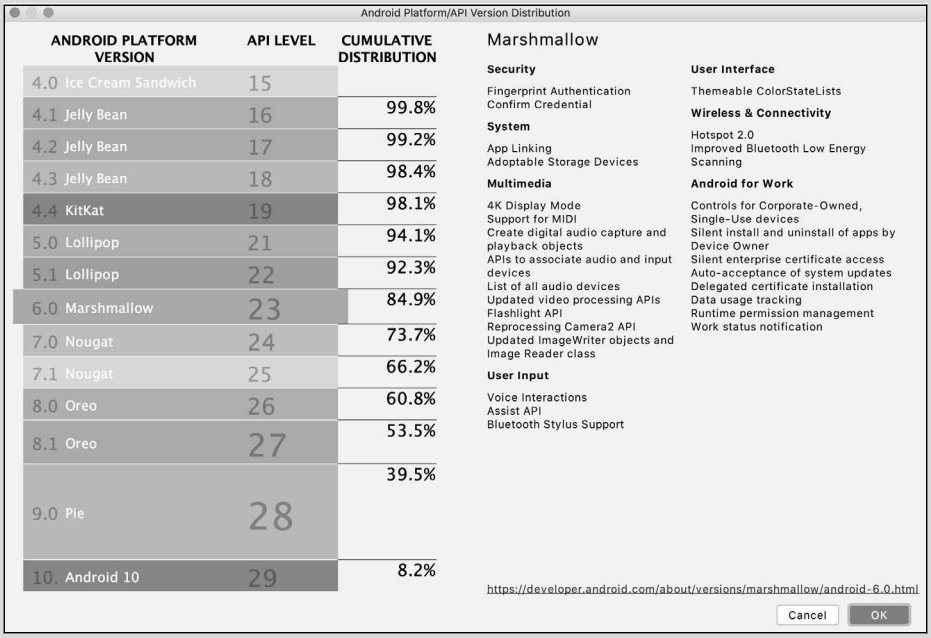

Abb. 3.4: Statistik zur Verbreitung der Android-Versionen

Ein Klick auf die einzelnen Versionen in dieser Übersicht zeigt auch gleichzeitig die Funktionen, die mit dieser Version eingeführt wurden. Zum Beispiel macht API 16 (Android 4.1) gar keinen Sinn, wenn Sie eine App schreiben wollen, die auf Bluetooth Smart (Bluetooth Low Energy) angewiesen ist. Diese Funktionalität wurde erst mit API 18 (Android 4.3) eingeführt.

Nach einem Klick auf FINISH wird das Projekt mit dem Grundgerüst erstellt und geöffnet. Dieser Prozess kann, abhängig von der Geschwindigkeit des Rechners (des Massenspeichers und der Internetverbindung), einige Zeit in Anspruch nehmen.

Quellcode (Ausgangsbasis)

`src/kap03-layouts/00-start`

3.2 Ausführen der App im Emulator

Nachdem Android Studio Ihnen ein Grundgerüst für die App erstellt hat, wollen Sie bestimmt wissen, wie dieses Grundgerüst auf dem Smartphone oder im Emu-

lator aussieht. Zuerst sehen wir uns die App auf einem »Android Virtual Device« (kurz AVD) an.

3.2.1 Neues AVD anlegen

Bei der Installation von Android Studio wird ein AVD nur installiert, wenn Sie es in der »Customer«-Installation ausgewählt haben. Wir legen jetzt (noch) eins an, da Sie diesen Vorgang in der Praxis häufig durchführen müssen, um ein Gerät mit bestimmten Eigenschaften zu simulieren (wie spezielle Sprache, Auflösung, Android-Version usw.).

Für die Anlage eines neuen AVDs müssen Sie den AVD Manager starten. Dazu haben Sie, wie unter Android Studio üblich, mehrere Möglichkeiten:

1. Über das Menü: Tools | AVD Manager
2. Über die Toolbar: Icon (AVD Manager)

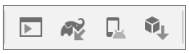

Abb. 3.5: Android-Studio-Toolbar

Nun öffnet sich der AVD Manager, der alle bis jetzt eingerichteten Emulationen anzeigt und kurze Informationen zu diesen auflistet. Wenn noch kein AVD vorhanden ist, erscheint ein Dialog, der zur Anlage eines neuen AVDs einlädt.

Abb. 3.6: AVD Manager ohne AVDs

Durch einen Klick auf CREATE VIRTUAL DEVICE ... gelangen Sie zu einem Assistenten, der die Anlage eines neuen Emulators erleichtert. Im ersten Bildschirm wählen Sie in der linken Spalte (CATEGORY ❶, Abbildung 3.7), was für ein Gerät Sie emulieren möchten (wie Smartphone, Tablet, Android Wear oder TV). Für unseren ersten Test wählen Sie PHONE. Als Modell ❷ eignet sich aktuell `Pixel 2 XL` sehr gut.

Einige der Vorlagen bieten auch einen integrierten Play Store. Damit können für Tests auch Apps installiert werden, mit denen die eigene App eventuell interagiert. Diese Integration erkauft man sich mit dem Nachteil, dass diese Vorlagen, im Gegensatz zu den übrigen, keinen »root«-Zugriff auf den Emulator erlauben. Solche Vorlagen sind an dem »Play Store«-Symbol in der Auflistung zu erkennen.

Ein »root«-Zugriff ist dann notwendig, wenn Sie später auf die Daten zugreifen möchten, die von Ihrer App erstellt wurden (zum Beispiel die Datenbank-Datei).

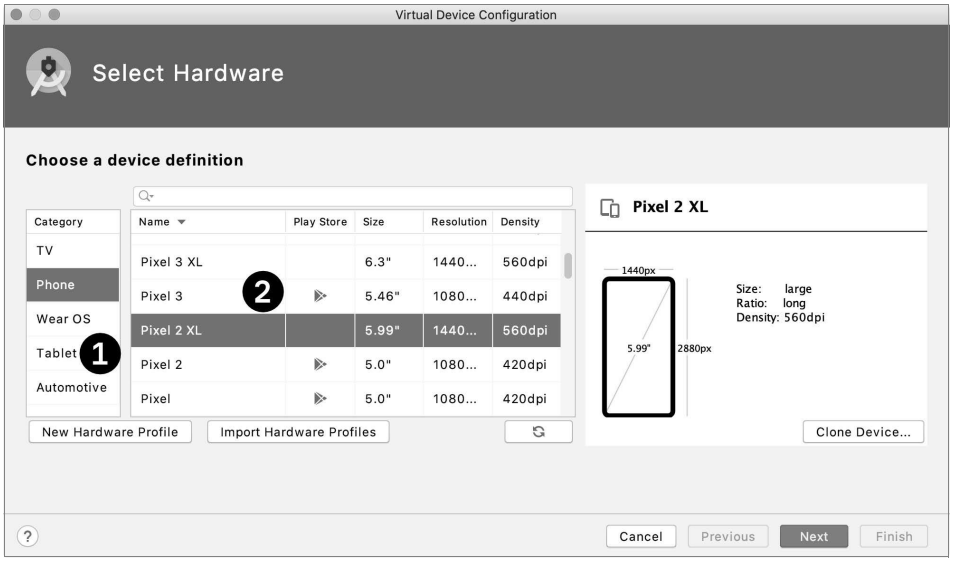

Abb. 3.7: AVD-Assistent – Basis für ein neues AVD

Nach dem Bestätigen mit NEXT können Sie auf dem nächsten Bildschirm die Android-Version auswählen, die auf dem emulierten Gerät laufen soll. Dabei gibt es Folgendes zu beachten:

1. CPU-Architektur: Aktuell werden Images für ARM, ARM x64, x86 und x86_x64 angeboten.
   - ARM-Images werden komplett emuliert. Das heißt, alle CPU-Befehle müssen durch Ihre reale CPU interpretiert werden. Deswegen sind die ARM-

Images sehr langsam, auch auf den aktuell schnellsten Rechnern. Es dauert auch mehrere Minuten, bis der Emulator gestartet wird (siehe dazu auch die Vergleichstabelle zu den Startzeiten im Anhang). Google arbeitet aktuell an der Optimierung der ARM Emulatoren.

- x86-Images werden durch den Intel-Treiber (HAXM – **H**ardware **A**ccelerated e**X**ecution **M**anager) direkt in der CPU Ihres Rechners ausgeführt, ohne Befehle zuerst interpretieren zu müssen (seit Android 3.2 werden auch AVDs auf AMD-Prozessoren beschleunigt, mit Hyper-V unter Windows). Auf den aktuellen Rechnern laufen deswegen die Apps in solchen Emulatoren deutlich schneller als auf den realen Geräten (ein Core i5 oder i7 ist immer noch deutlich schneller als jede mobile CPU). Der Zusatz x64 steht dafür, dass das Abbild (Image) ein 64Bit-Android enthält und somit nur auf einer 64-bitigen CPU laufen kann (sollte mittlerweile Standard sein). Auch das Entwickler-Betriebssystem muss 64-bitig sein. Relevant wird diese Unterscheidung erst dann, wenn Sie einen C/C++-Code ausführen möchten.

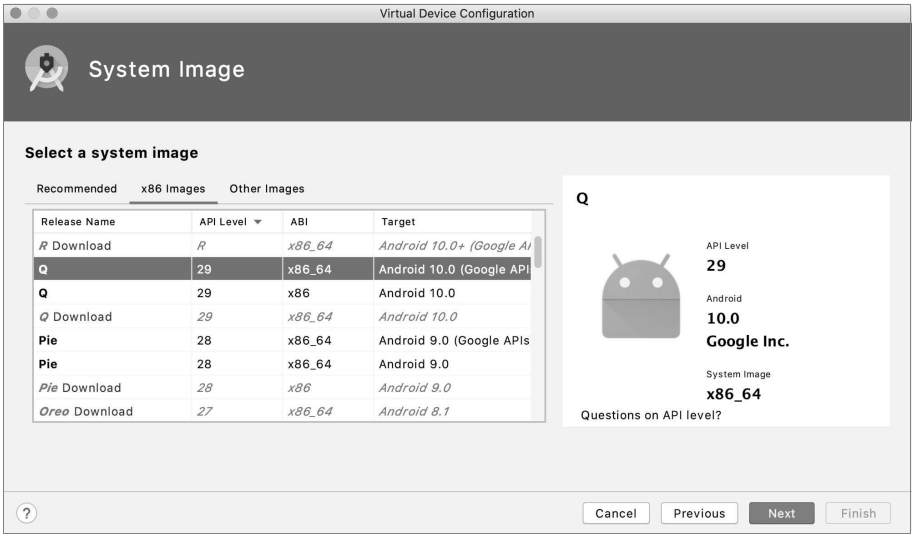

Abb. 3.8: AVD-Assistent – Auswahl des Android SDKs

2. Google API

   So gekennzeichnete Images sind ein wenig größer, da sie neben den reinen Android-Services auch die Services für Google-Dienste beinhalten, wie Maps, Play-Store-Service, Firebase usw. Diese Images benötigen Sie immer, wenn die App diese Services konsumieren soll.

3. API-Version

   Dies ist die eigentliche Version von Android. Abhängig von den Testkriterien sollten Sie eine für Sie passende auswählen.

Sollte im RECOMMENDED Tab nicht die gewünschte API Version vorhanden sein, wechseln Sie zum X86 IMAGES-Tab. Hier können Sie die eventuell noch fehlende Images für eine Version direkt herunterladen. Wählen Sie hier `Android Q (API 29)` als `x86` oder `x86_64` für unseren Emulator.

Der nächste Bildschirm zeigt die Details des neuen virtuellen Geräts.

Abb. 3.9: AVD-Assistent – Details des neuen AVDs

In der Standardansicht für die Details können folgende Daten angepasst werden:

- »AVD Name«: Name des AVDs

  Es ist empfehlenswert, an dieser Stelle eine Konvention zu nutzen, um anhand des Namens die Eigenschaften des AVDs schnell zu überblicken, wenn Sie viel mit AVDs arbeiten. Zum Beispiel `A21_x86_de` für Android AVD mit API *21,* basierend auf dem *x86*-Abbild und in *deutscher* Sprache.

- Basis-Gerät, auf dem dieses AVD basiert, das wir auf dem ersten Bildschirm ausgewählt haben.
- SDK-Version für das AVD aus dem zweiten Bildschirm.
- »Startup Orientation«: Ausrichtung des AVDs

  Hoch- oder Querformat, die Umschaltung ist auch während der Laufzeit des AVDs möglich.
- »Emulated Performance Graphics«: Nutzung der Grafikkarte des Host-Rechners beschleunigt bei guter Grafikkarte die Animationen und Reaktionszeit des AVDs.
- »Device Frame«: Zeigt an, ob bei dem Emulator auch ein Geräterahmen angezeigt werden soll. Praktisch: Wenn Sie einen kleinen Bildschirm haben, können Sie den Rahmen weglassen und so mehr von der App sehen.

Weitere Einstellungen können vorgenommen werden, wenn Sie auf SHOW ADVANCED SETTINGS klicken.

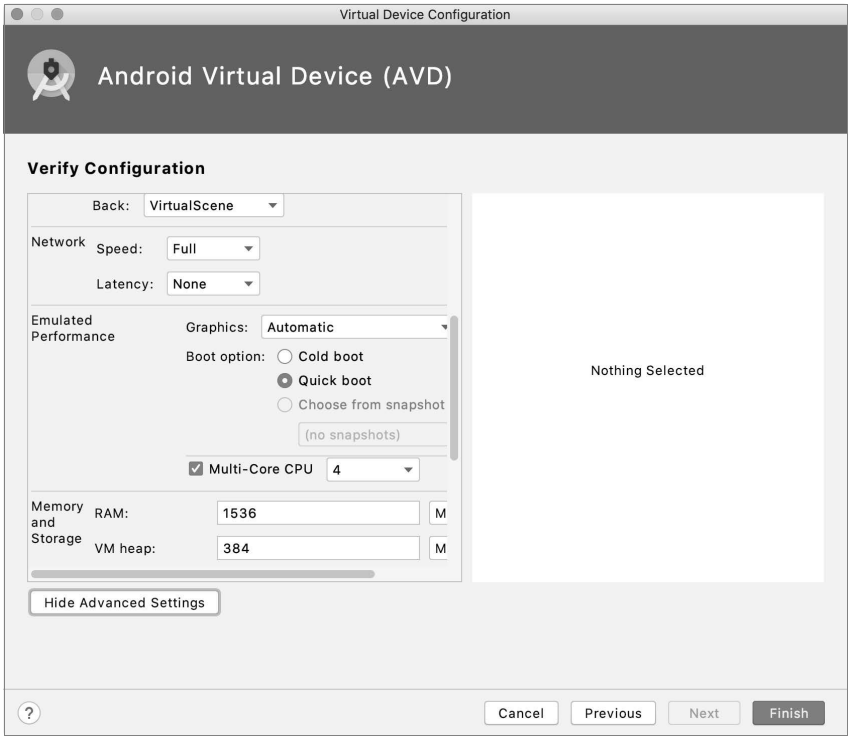

Abb. 3.10: AVD-Assistent – Erweiterte Einstellungen für neues AVD

In der erweiterten Ansicht sind folgende Einstellungen die wichtigsten:

- »Memory and Sorage«: Größe des Speichers

  Wichtig: Wenn Ihr Entwickler-Rechner wenig Arbeitsspeicher hat und Sie HAXM deshalb während der Installation nur wenig Speicher zugestanden haben, dann sollte der Arbeitsspeicher kleiner als der für HAXM zugebilligte sein, wenn Sie ein x86-Abbild nutzen.
- »SD-Card«: Speichergröße der SD-Karte
- »Keyboard«: Aktivierung der Hardware-Tastatur

  Das ist sehr nützlich, da sonst alle Eingaben über die Android-Tastatur im AVD mit der Maus gemacht werden müssen.
- »Camera«: Einstellen der beiden Kameras (emuliert – eine Animation – virtuelle 3D-Szene, keine oder Ihre Webcam)
- »Network«: Geschwindigkeit des Netzwerks (WLAN, LTE, GPRS usw.). Das kann auch später zur Laufzeit des Emulators geändert werden.
- »Device Frame«: Rahmen für das AVD, damit dieses wie ein »echtes« Gerät aussieht.

Nach der Bestätigung mit FINISH wird das neue AVD generiert. Es kann auf langsameren Rechnern einige Zeit dauern, bis es erzeugt wurde. Danach schließen Sie den AVD Manager.

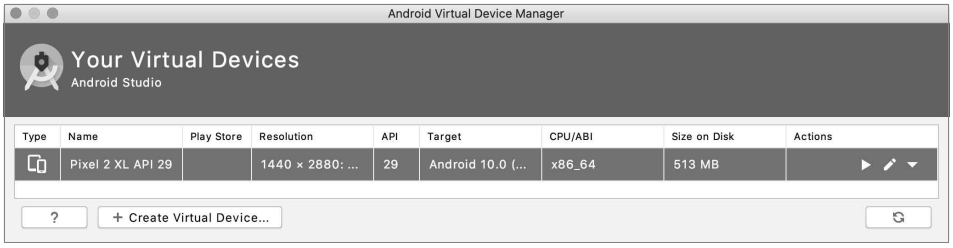

Abb. 3.11: AVD Manager mit gerade eingerichtetem AVD

3.2.2 Starten der App

Um eine App auf einem Emulator zu starten, müssen Sie zuerst in der Auswahlbox das Gerät selektieren, auf dem die App laufen soll. Im oberen Bereich werden

kompatible, bereits laufende Geräte angezeigt. Im unteren Bereich lässt sich ein vorhandenes AVD aus der Liste auswählen.

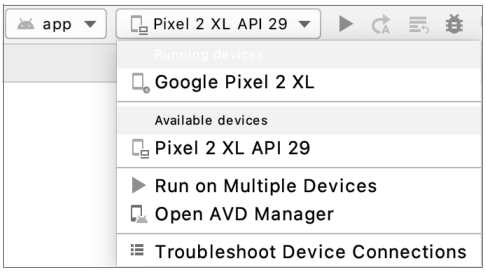

Abb. 3.12: Auswahl eines Gerätes oder AVDs

Dann haben Sie, wie bei Android Studio üblich, mehrere Möglichkeiten, die App zu starten:

1. Über das Menü: RUN | RUN 'APP'
2. Über die Toolbar mit dem ▶-Icon
3. Über die Tastenkombination Control+R unter OS X und ⇧+F10 unter Windows/Linux

Der Start des AVDs kann abhängig vom System sehr lange dauern (mitunter auch mehrere Minuten), insbesondere der erste Start (wie ein richtiger Neustart auf einem Android-Gerät nach dem Zurücksetzen auf die Werkseinstellungen).

AVD-Nutzung

Schließen Sie das AVD nach dem Start *nicht*, da der Neustart immer eine gewisse Zeit dauert. Wenn Sie Ihren Code ändern und die App neu mit RUN 'APP' ausführen, wird die im Emulator laufende App durch die neue Version ersetzt. Der Emulator muss dabei nicht neu gestartet werden.

Im neu gestarteten AVD erscheint nach dem Start von Android die Hello-World-App, die von dem Assistenten erzeugt wurde (wischen Sie, falls nötig, das Schloss-Symbol nach oben, um das AVD zu entsperren). Nun haben Sie eine lauffähige Basis, mit der Sie weiterarbeiten können.

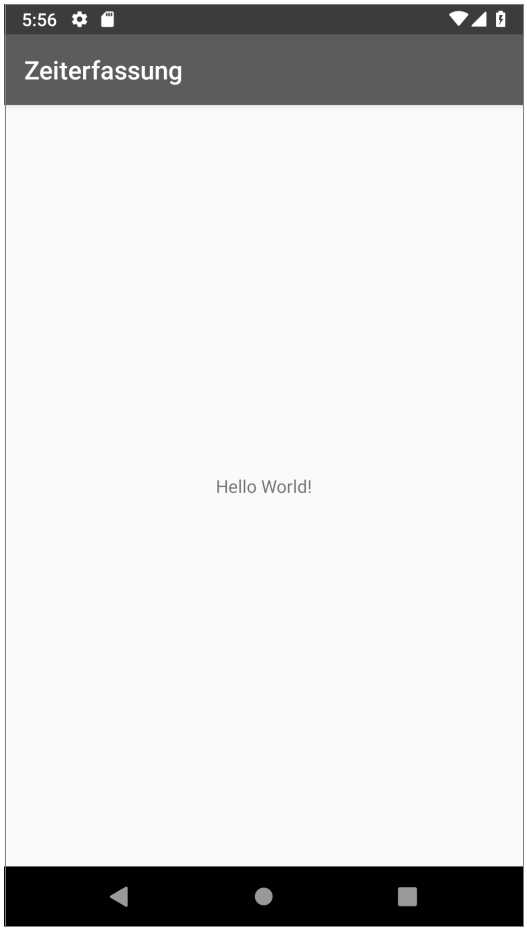

Abb. 3.13: Hello-World-App im AVD

Wenn Sie die App auf Ihrem Smartphone oder Tablet ausführen möchten, schließen Sie Ihr Gerät per USB-Kabel an. Es erscheint dann beim Start der App im Geräte-Auswahl-Dialog im oberen Bereich.

Wie Sie Ihr Smartphone oder Tablet für die Entwicklung freischalten und für den ersten Lauf einrichten, lesen Sie im Anhang A.3.

3.3 App-Bausteine

Schauen wir uns nun das generierte Projekt genauer an, um die Grundbausteine einer Android App kennenzulernen.

3.3.1 Manifest

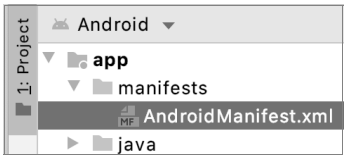

Abb. 3.14: AndroidManifest in Projektbaum-Ansicht

Jedes Android-Projekt (egal, ob App, Widget oder Bibliothek) muss ein Manifest haben. Dieses beschreibt die nach außen kommunizierenden Eigenschaften des Projekts.

Das sind zum Beispiel:

- `Package`: eindeutiger Bezeichner der App
- `Activities`: sichtbare Bildschirme für den Anwender
- `Permissions`: Berechtigungen der App
- `Services`: Die »unsichtbaren« Komponenten/Dienste, die das Projekt bietet. Ein möglicher Service wäre zum Beispiel das Herunterladen der Daten im Hintergrund, oder ein Dienst zur Synchronisation der Daten mit Internet.
- `ContentProvider`: Bereitstellung der eigenen Daten an externe Abnehmer (Apps, Widgets usw.)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="de.webducer.ab3.zeiterfassung">

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
```

```
      </intent-filter>
    </activity>
  </application>

</manifest>
```

Listing 3.1: Beispiel für eine AndroidManifest.xml Datei

3.3.2 Activity

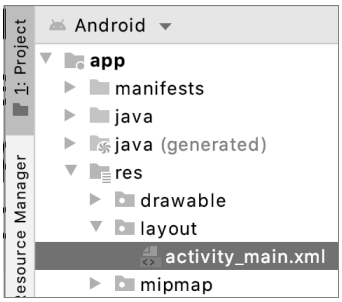

Abb. 3.15: Main-Actrivity Layout Datei im Projekt

Als »Activity« wird unter Android die größte visuelle Einheit bezeichnet. Unter Desktop-Systemen entspricht eine Activity eher einem Fenster. Die Activity ist eine selbstständige Einheit, die die komplette Logik selbst enthält und unabhängig von anderen arbeiten kann. Unter Android kann eine Activity nicht direkt von einer anderen aufgerufen werden. Die Aufrufe erfolgen immer durch das Betriebssystem über Nachrichten (Intents).

3.3.3 Fragment

»Fragmente« wurden mit Android 3.1 eingeführt, um Oberflächen flexibler gestalten zu können. Fragmente sind Teilbereiche einer Activity, haben aber die komplette Logik, um die enthaltenen Elemente zu verwalten, in sich. Damit können die Fragmente in anderen Bereichen wiederverwendet werden, zum Beispiel in einer anderen Activity oder auch mehrfach in derselben.

3.3.4 Ressourcen

Android unterscheidet zwei Typen von Ressourcen, »verwaltete« und »nicht verwaltete« Ressourcen. Die nicht verwalteten Ressourcen werden im Modul-Unterordner `assets` abgelegt.

Als Ressourcen bezeichnet man unter Android alle Daten, die über einen eindeutigen Namen angesprochen werden können.

Unter nicht verwalteten Ressourcen versteht man Ressourcen, die nicht vom Betriebssystem unterschieden, sondern direkt vom Entwickler im Code aufgerufen werden. Der `assets`-Ordner kann vom Entwickler so gestaltet werden, wie dieser es für richtig hält. Es gibt keine Android-Vorgaben zu Unterordnern, Datentypen usw. Meistens werden Videos, Musik und Bilder in Spielen als nicht verwaltete Ressourcen benutzt.

Die verwalteten Ressourcen liegen im Unterordner `res` und verteilen sich auf unterschiedliche, von Android vorgegebene Unterordner. Verwalten bedeutet an dieser Stelle, dass das Betriebssystem während der Laufzeit abhängig vom Kontext bestimmte Ressourcen lädt. Zum Beispiel wird das als verwaltete Ressource hinterlegte Layout abhängig von der Geräteausrichtung geladen. Folgende oft benutzte Ressourcen-Typen gibt es:

- Layouts: liegen im Ordner `res/layout`
- Menüs: liegen im Ordner `res/menu`
- Bilder (Pixel- und Vektorgrafiken): liegen im Ordner `res/drawable`
- App-Icons: liegen im Ordner `res/mipmap`
- Wertpaare: liegen im Ordner `res/values`
- XML-Dateien: liegen im Ordner `res/xml`
- Rohdaten: liegen im Ordner `res/raw`
- Übergänge/Animationen: liegen im Ordner `res/transition`

All diese verwalteten Ressourcen können eine oder mehrere Präzisierungen/Spezialisierungen erfahren. Dabei werden an den Basisordner ein oder mehrere Suffixe mit »-« angehängt. So steht zum Beispiel `res/values-en-land-w820dp` für die Wertpaare-Ressourcen, die nur dann vom Betriebssystem geladen werden, wenn das System in englischer Sprache (`-en`) ist, im Querformat (`-land`) läuft und eine Mindestbreite von 820 (`-w820dp`) aufweist.

3.3.5 Layout

Bevor wir ein eigenes Layout anlegen, schauen wir uns an, wie Layouts unter Android definiert werden.

In den meisten Fällen wird unter Android das Layout in XML-Dateien definiert. Damit erreicht man eine gute Trennung zwischen dem Aussehen (XML-Dateien) und der Logik (Java-Code).

Jetpack Compose

Aktuell gibt es wieder Ansätze, auch die UI mit einer Programmiersprache (wie Kotlin) zu schreiben, statt mit einer Beschreibungssprache (wie XML). Auch bei Google wird mit Jetpack Compose aktuell dieser Ansatz verfolgt. Dabei werden die sehr guten Eigenschaften von Kotlin bei der Definition einer DSL (Domain Specific Language) ausgenutzt.

wdurl.de/ab3-jpc

Alle sichtbaren Elemente in Android leiten sich von der Basisklasse `View` ab. Die Elemente werden in zwei Kategorien unterteilt:

1. ViewGroup
2. View

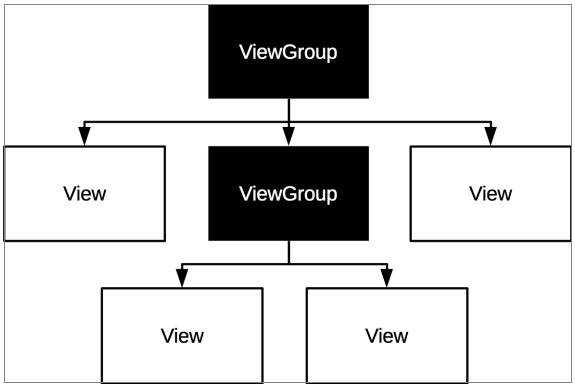

Abb. 3.16: Hierarchischer Aufbau einer Oberfläche aus ViewGroups und Views

ViewGroup

»ViewGroups« sind Container, die andere Oberflächenelemente enthalten können (auch weitere ViewGroups). Sie sind für die Anordnung der einzelnen Elemente an der Oberfläche verantwortlich. Einige der ViewGroups werden schon von Anfang an durch das Android-Betriebssystem unterstützt, andere lassen sich durch Bibliotheken einbinden. Hier ein kurzer Überblick über die am häufigsten eingesetzten ViewGroups:

ViewGroup: LinearLayout

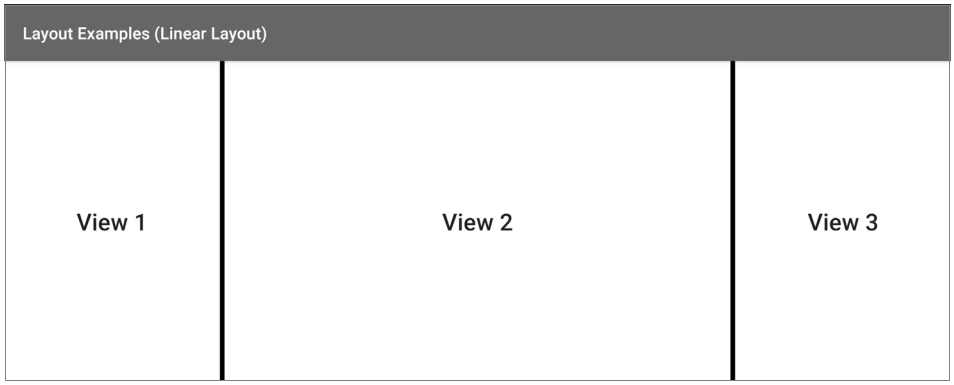

Abb. 3.17: »LinearLayout« mit horizontaler Ausrichtung

Der Container `LinearLayout` ordnet die Kind-Elemente (Views) entweder unter- oder nebeneinander. Die Anordnung hängt von der Eigenschaft `android:orientation` ab, die den Wert `horizontal` oder `vertical` annehmen kann. Standardwert für die Ausrichtung ist `horizontal`. Damit zählt dieser Container zu den einfachsten, leider zugleich aber auch zu den langsamsten (relativ gesehen). Durch das Verschachteln können auch mit diesem relativ einfachen Container sehr komplexe Layouts entworfen werden.

LinearLayout ist einer der wenigen Container, die eine prozentuale Aufteilung zwischen den Kind-Elementen erlauben.

Die Reihenfolge in dem fertigen Layout entspricht der Reihenfolge, in der die Kind-Elemente dem Container hinzugefügt (in Java) oder definiert (XML) werden.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:background="@color/colorPrimaryDark"
  android:orientation="horizontal"
  android:weightSum="10">

  <TextView
    android:layout_width="wrap_content"
```

```
    android:layout_height="match_parent"
    android:layout_weight="2"
    android:gravity="center"
    android:text="View 1" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_weight="6"
    android:gravity="center"
    android:text="View 2" />

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:layout_weight="2"
    android:gravity="center"
    android:text="View 3" />

</LinearLayout>
```

Listing 3.2: XML-Beispiel für `LinearLayout` mit drei Text-Elementen

ViewGroup: RelativLayout

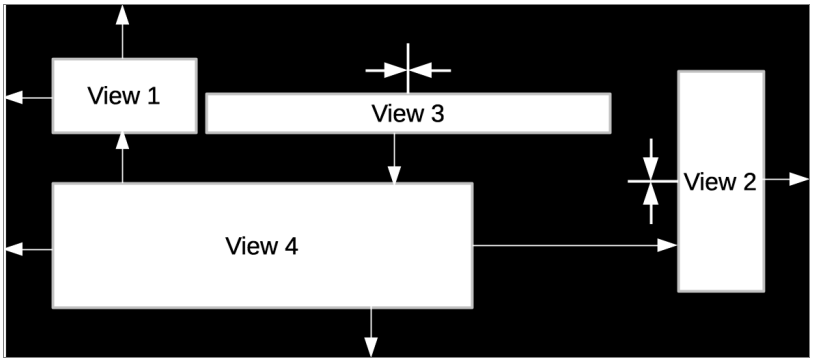

Abb. 3.18: Beziehungen der »Views« zueinander in einem RelativeLayout

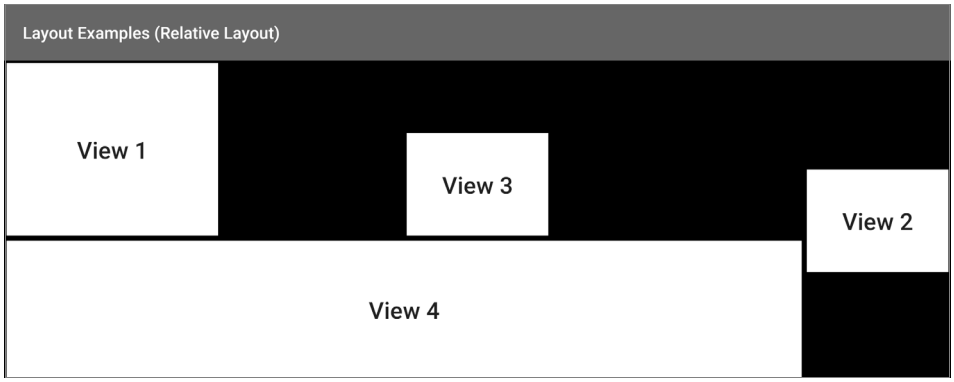

Abb. 3.19: Vorschau des RelativeLayouts in Android

Der Container `RelativeLayout` ordnet die Kind-Elemente relativ zu sich selbst oder zu den anderen Elementen an. Er erlaubt sehr komplexe Layouts mit nur einer einzigen Hierarchie-Ebene und ist sehr performant. Den Aufbau eines mit RelativeLayout erstellten Layouts zu verstehen, ist aber deutlich schwerer als bei LinearLayout. Der Umstand, dass die Reihenfolge der Definition der Kind-Elemente keine Rolle bei der Anordnung der Elemente im Container spielt, führt häufig zu Verwirrung. Das erste definierte Element kann auf dem Bildschirm im Container ganz unten, in der Mitte oder rechts stehen.

Typische Angaben für die Platzierung eines Elements sind in Abbildung 3.18 und 3.19 zu sehen:

- View 1:
  - Am linken Rand des Containers
  - Am oberen Rand des Containers
- View 2:
  - Am rechten Rand des Containers
  - Rechts von »View 1«
  - Vertikal zentriert im Container
- View 3:
  - Oberhalb von »View 4«
  - Horizontal zentriert im Container

Stichwortverzeichnis