
mitp
Eugen
Richter
Android-Apps
programmieren
Praxiseinstieg mit Android Studio

Inhaltsverzeichnis

Vorwort

Im mobilen Segment hat Android mittlerweile einen Marktanteil von über 90 %. An dieser großen Verbreitung möchten Sie sicherlich teilhaben, ob als Hobby-Programmierer oder in einem Unternehmen. Aus diesem Grund halten Sie dieses Buch in den Händen.

An wen richtet sich dieses Buch?

Dieses Buch richtet sich in erster Linie an Anfänger, die in die Android-Entwicklung einsteigen oder an Programmierer, die alte Kenntnisse auffrischen möchten.

Voraussetzungen

Die wichtigste Voraussetzung für den Leser ist ein grundlegendes Verständnis für objektorientierte Programmierung, im Idealfall in der Programmiersprache Java. Aber auch ohne Java-Kenntnisse (zum Beispiel 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.

Technischer Stand

Dieses Buch wurde mit der Android Studio Version 2.3.3 geschrieben. Wenn Sie eine neuere Version haben, werden eventuell einige Screenshots anders aussehen, als im Buch abgebildet. Das grundlegende Vorgehen sollte sich aber nicht ändern.

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:

- Das Buch basiert auf Android Studio 2.3. Android Studio 3.0 (oder 2.4), das Kotlin besser unterstützen soll, war beim Schreiben des Buches noch in der Alpha/Beta-Phase.
- Es läuft noch nicht alles reibungslos mit Kotlin, da das Framework über Jahre hinweg mit dem Schwerpunkt auf Java entwickelt wurde.
- Die meisten Beispiele im Netz (aber auch die offiziellen von Google) für Android zeigen den Java-Code. Aus diesem Grund ist es für einen Android-Entwickler besser, mit Java zu starten. Ich hoffe, dass sich Kotlin mit der Zeit mehr durchsetzt, da die Sprache wirklich deutlichen Mehrwert im Vergleich zu Java bietet.

Konventionen

XML-Dateien

- Dateiname: Kleinschreibung mit Worttrennung durch »_«.

  Beispiel: `activity_main.xml`
- Ressource-Namen (für IDs, Strings, Abstände usw.): »Camel Case«-Schreibweise.

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

Hinweis

Lange Code-Zeilen passen leider nicht immer in eine Zeile im Buch. Den Teil, der eigentlich noch mit in die Zeile gehört, erkennen Sie im Buch daran, dass er rechtsbündig eingerückt ist. In Android Studio schreiben Sie den Code dann bitte zusammenhängend in eine Zeile.

Beispiel im Buch:

```
/**
* Basis URI zu dem Content Provider
*/
public static final Uri AUTHORITY_URI = Uri.parse("content://"
                                                        + AUTHORITY);
```

So sieht es in Android Studio aus:

```
1  /**
2   * Basis URI zu dem Content Provider
3   */
4  public static final Uri AUTHORITY_URI = Uri.parse("content://"+ AUTHORITY);
```

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(start-
Date.getTime());
  ...
}
```

Beispieldateien

Auf der Bitbucket-Projektseite `wdurl.de/AB-App` finden Sie alle Beispieldateien sowie viele weitere Informationen zu den Themen im Buch.

Kapitel 1

Grundlagen

In diesem Kapitel befassen wir uns mit den Grundlagen für die Android-Programmierung. Dazu werden wir eine Entwicklungsumgebung aufsetzen und die Grundbausteine der Android-Programmierung kennenlernen. Tieferes Verständnis wird in den folgenden Kapiteln in Form eines Workshops vermittelt. Während der praktischen Umsetzung ist es so einfacher, das Gelernte zu festigen, als durch reine Theorie-Vermittlung.

1.1 Entwicklungsumgebung

Bei der Programmierung ist es wichtig, nicht nur die Programmiersprache zu kennen, sondern auch die Werkzeuge. In diesem Abschnitt werden wir daher die Entwicklungsumgebung installieren und einrichten.

Android Studio wurde auf der Google I/O 2013 zum ersten Mal präsentiert und als Alpha-Version veröffentlicht. Ein Jahr später, auf der Google I/O 2014, wurde es zum Beta-Status erhoben. Bereits im Dezember 2014 erschien dann die erste finale Version 1.0.

Seit Dezember 2014 ist Android Studio die offizielle Entwicklungsumgebung für die Android-Entwicklung. Es hat Eclipse mit den Android Developer Tools abgelöst. Neuere Funktionen erscheinen seitdem nur noch für Android Studio. Android Developer Tools werden zwar durch die Eclipse Foundation weiterentwickelt, aber nicht mehr von Google selbst.

Die neue Entwicklungsumgebung basiert auf der nicht nur unter Java-Entwicklern sehr beliebten IDE (englische Abkürzung für **I**ntegrated **D**evelopment **E**nvironment) von JetBrains Intelli J IDEA. Diese Entwicklungsumgebung für »Java Virtual Machine« (kurz JVM) basierte Sprachen unterstützt den Entwickler beim Refactoring (Anpassung des Codes ohne Änderung der Logik).

Neben IntelliJ IDEA liefert die Firma JetBrains noch weitere Entwicklungsumgebungen für andere Sprachen, wie WebStorm für die Web-Entwicklung, PHP-Storm für PHP-Entwicklung, RubyMine für Ruby on Rails, ReSharper für .Net-Entwickler und weitere.

Zusätzlich zu der Umstellung der Entwicklungsumgebung von Eclipse auf Android Studio wurde auch das Buildsystem von Ant bei Eclipse auf Gradle unter Android Studio gewechselt.

Eine kurze Einführung in das Gradle-Buildsystem finden Sie weiter unten im Abschnitt »Gradle«.

Android Studio – Open Source

Android Studio ist ein Open Source-Projekt. Der Quellcode kann unter `tools.android.com/build/studio` heruntergeladen und eigene Android Studio Versionen damit gebaut werden.

1.1.1 Installation

Nun geht es ans Werk. Die aktuelle Version von Android Studio Bundle, Android Studio mit »Software Development Kit« (kurz SDK), erhält man von der offiziellen Android-Entwicklerseite (`developer.android.com/studio/`).

Im Normalfall erkennt der Browser das aktuelle Betriebssystem und schlägt die passende Datei für den Download vor. Wenn dies nicht der Fall ist oder man die Installationsdatei für einen anderen Rechner benötigt, findet man diese auf der oben genannten Seite unter dem Reiter »Preview« und dort dann unter »More Download Options«.

Android Studio wird für die gängigen Betriebssysteme Windows, OS X und Linux angeboten.

Für Windows stehen mehrere Versionen für den Download zur Verfügung.

- Android Studio mit der letzten Version des SDK (Standard)
- Android Studio ohne SDK (wird bei Bedarf nachgeladen)
- Android Studio ohne SDK als Zip-Archiv

Unter Linux und OS X wird das SDK erst beim ersten Start von Android Studio nachgeladen.

Als Hardware-Voraussetzungen sind ein moderner Rechner mit mind. 3 GB RAM, 2GB freiem Speicherplatz und 1280 x 800 Auflösung notwendig (siehe aktuelle Anforderungen unter `developer.android.com/studio/index.html#Requirements`). Mit dieser Minimalanforderung wird die App-Entwicklung aber nicht wirklich Spaß machen. Unabhängig von der Android-Entwicklung empfehle ich einen modernen Rechner mit mindestens folgender Ausstattung:

- Core i5 / i7 (ab 3. Generation) mit aktivierter Virtualisierungstechnik VT-x, EM64T und XD
- 8 GB RAM (für Android Virtual Devices-AVDs)
- SSD statt Festplatte
- 24“ Bildschirm mit Full-HD Auflösung (1920 x 1080)

Die Virtualisierungstechnik ist notwendig, um schnellere Emulatoren installieren zu können (Intel HAXM, Genymotion oder Xamarin Emulator). Die Standard-Emulatoren (ohne Virtualisierungstechnik) sind leider sehr langsam und machen bei der Entwicklung keinen Spaß. Genaueres können Sie im Anhang im Abschnitt »Installation von HAXM« nachlesen.

Vor der Installation von Android Studio stellen Sie bitte sicher, dass die aktuelle Version des Java Software Development Kits (kurz JDK) installiert ist. Android Studio verlangt hier mindestens Version 7. Wenn keine Gründe dagegen sprechen, empfehle ich die Installation von JDK 8 (in der aktuellsten Version).

Installation unter Windows

Abb. 1.1: Assistent für die Android Studio-Installation

Nach dem Download, der über 1 GB groß ist (Android Studio mit SDK), kann die Installation starten. Wenn die Java-Installation nicht gefunden wurde, kann man im Dialog den Pfad dazu angeben. Wenn noch kein JDK installiert wurde, schlägt die Android Studio-Installation einen entsprechenden Download vor (in der Version 7).

Wenn man die Standardeinstellungen bei der Installation beibehält, installiert das Setup neben Android Studio auch das SDK und ein Android Virtual Device (kurz AVD). Beim AVD handelt es sich um die x86-Version, die auf Intel-Systemen mit einen entsprechenden Prozessor (Intel Virtualization Technology – VT-Erweiterung), deutlich schneller läuft, als die emulierten AVDs in ARM-Version.

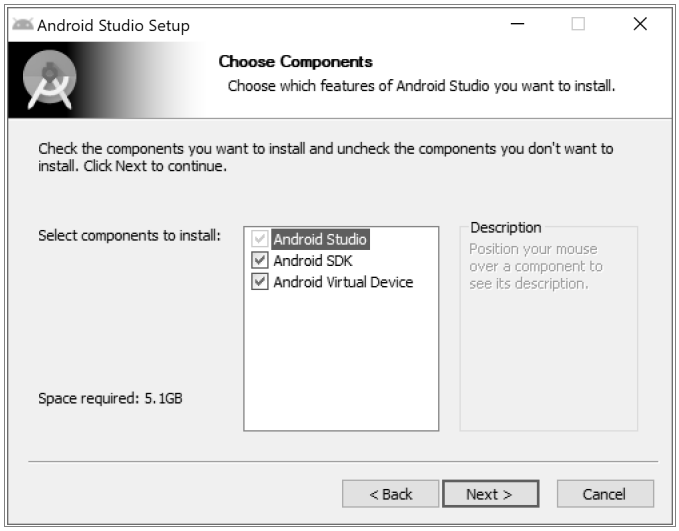

Abb. 1.2: Auswahl der Komponenten bei der Installation

Der erste Start sollte nun laufen, und man sollte den Setup-Assistenten seine Arbeit tun lassen (Assistent-Typ Standard). Bestätigen Sie die Lizenzen, der Assistent lädt danach eventuell noch fehlende Komponenten und Aktualisierungen nach.

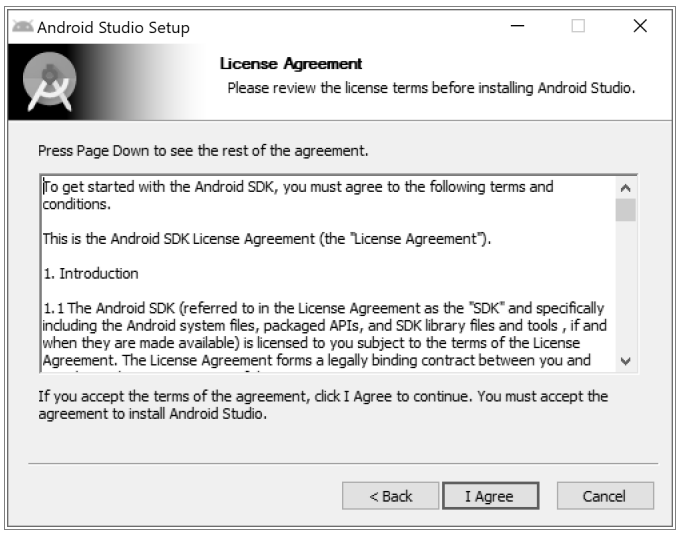

Abb. 1.3: Bestätigung der Lizenzbestimmungen

Im weiteren Verlauf können die Installationsordner für Android Studio (Standard ist `C:\Programme\Android Studio`) und für Android SDK (Standard ist `C:\Benutzer\<Benutzer>\AppData\Local\Android\sdk`) festgelegt werden.

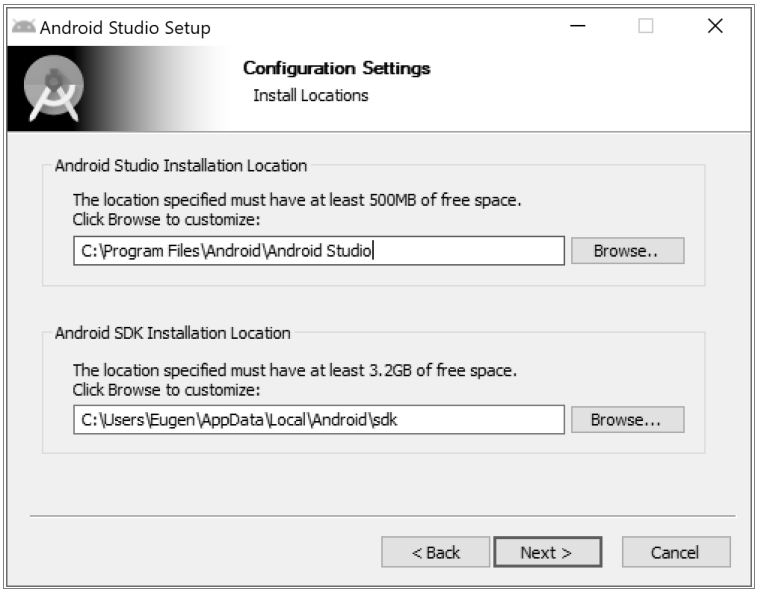

Abb. 1.4: Installationsordner für IDE und SDK

Nach Abschluss der Installation und beim Start erscheint eine Meldung mit der Möglichkeit, die Einstellungen von einer früheren Version zu übernehmen.

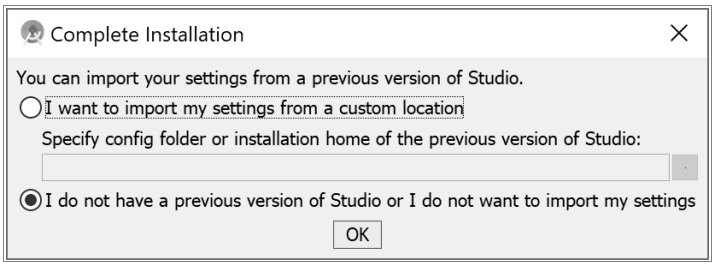

Abb. 1.5: Übernahme der Einstellungen beim ersten Start

Abhängig von der Windows-Version und der Einstellung der Firewall, kann es beim Start zur einer Anfrage kommen, ob Android Studio sich mit dem Internet verbinden darf (für eigene Updates und für Updates von Android SDK). Um auf dem aktuellen Stand zu bleiben, sollte man natürlich den Zugriff gewähren.

Kurz nach dem Start, wenn man mit dem Internet verbunden ist, kann eine weitere Meldung erscheinen, die über neue Versionen informiert (zu Android Studio, Android SDK oder Plugins). Diese sollte natürlich beachtet werden. Sofortige Installation ist zwar nicht notwendig, ist aber zu empfehlen.

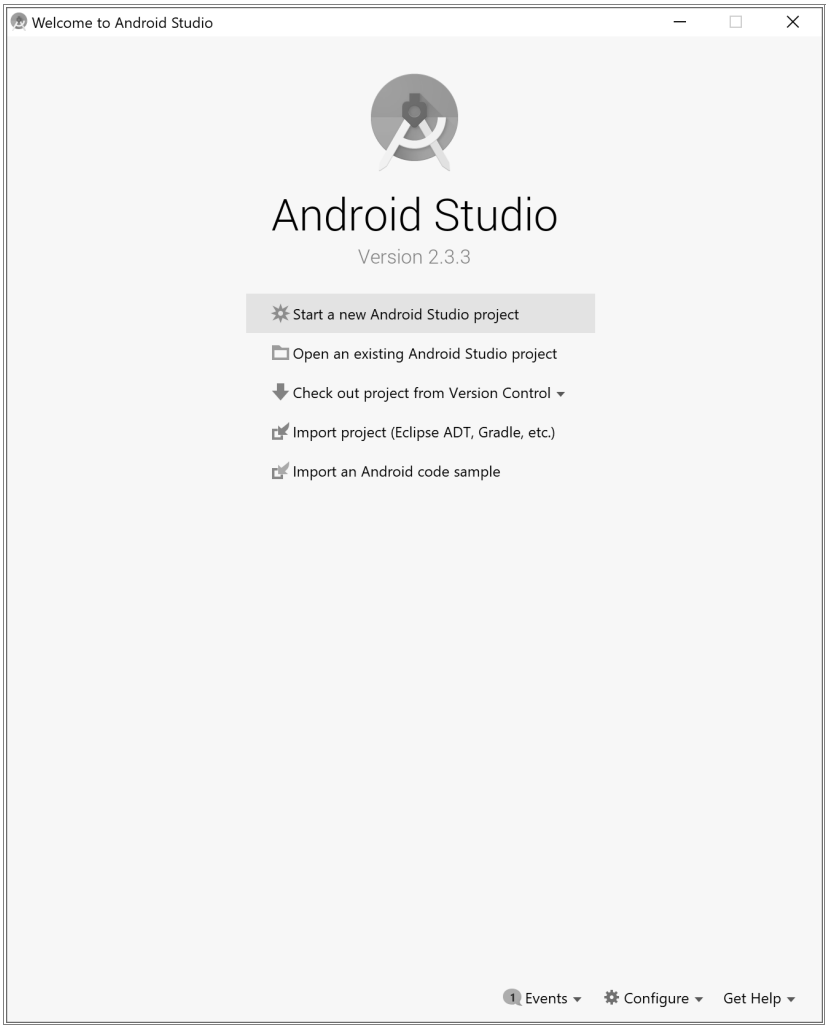

Abb. 1.6: Neue Updates nach dem Start (Events unten)

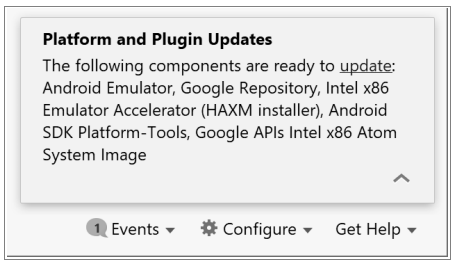

Abb. 1.7: Anzeige des Update-Events auf dem Startbildschirm

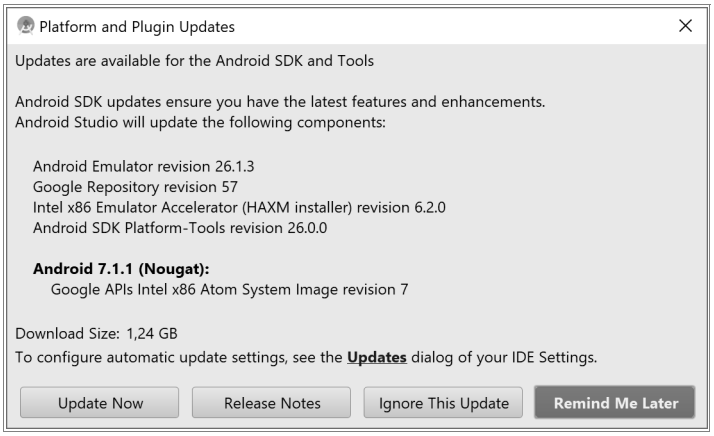

Abb. 1.8: Update-Dialog für Android-Komponenten

Installation unter OS X und Linux

Unter OS X und Linux braucht man nur das Paket an einem beliebigen Ort zu entpacken. Im Paket befindet sich nur Android Studio, kein SDK, wie bei der Standard-Windows-Installation.

Abb. 1.9: Assistent zum Nachladen der Komponenten nach dem ersten Start unter OS X

Nach dem ersten Start von Android Studio startet ein Assistent, mit dem das Android SDK nachgeladen und das erste AVD eingerichtet werden (inkl. HAXM-Erweiterung).

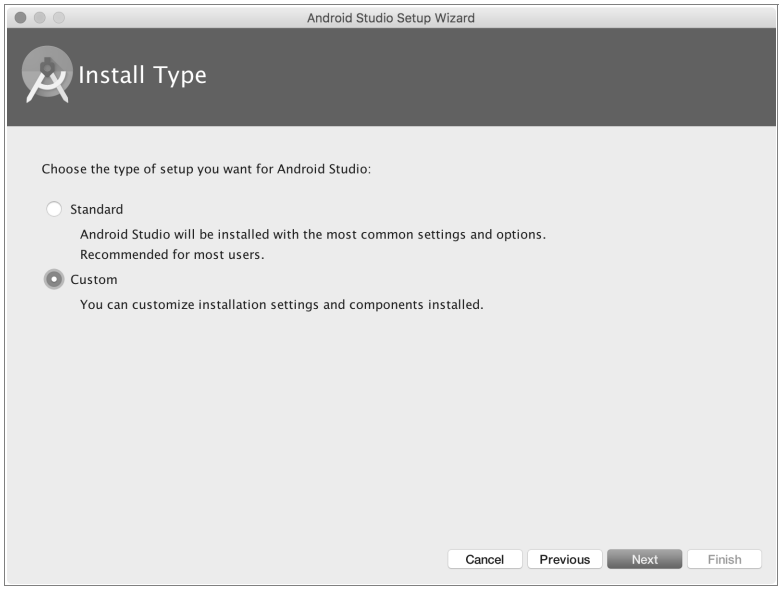

Abb. 1.10: Installationstyp für den Assistenten

Wenn man die Installationsmethode »Custom« auswählt, kann die Installation an einigen Stellen angepasst werden (unter anderem die RAM-Größe für HAXM, wohin SDK installiert wird usw.).

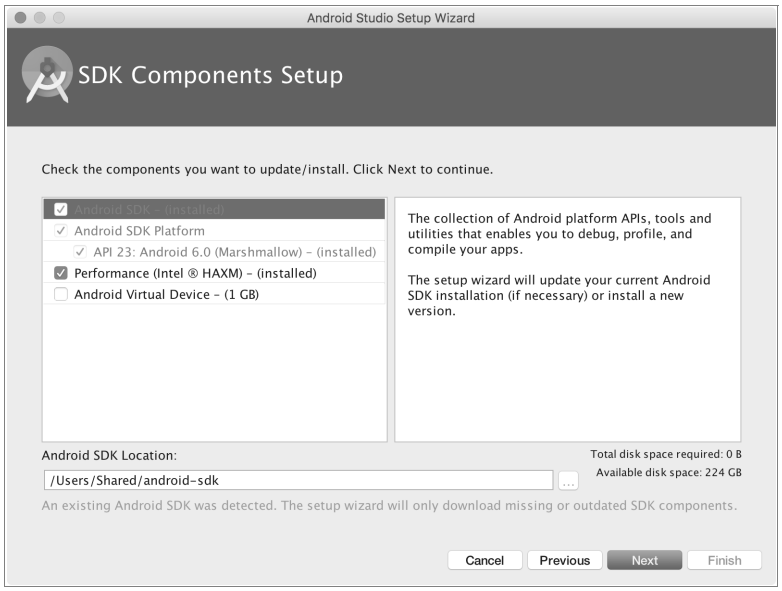

Abb. 1.11: Auswahl der zu installierenden Komponenten

Android SDK-Installationsordner

Das SDK wird unter Windows in den Ordner `c:\Users\<Benutzer>\AppData\Local\Android\Sdk\`, unter OS X in den Ordner `/Users/<Benutzer>/Library/Android/sdk/` installiert. Diesen Ordner zu kennen ist wichtig, um zum Beispiel über die Kommandozeile mit Android Debug Bridge (kurz ADB) zu arbeiten, Intel Hardware Accelerated Execution Manager (kurz HAXM) manuell zu installieren oder mitgelieferte Ressourcen anzusehen.

1.1.2 Struktur der Entwicklungsumgebung

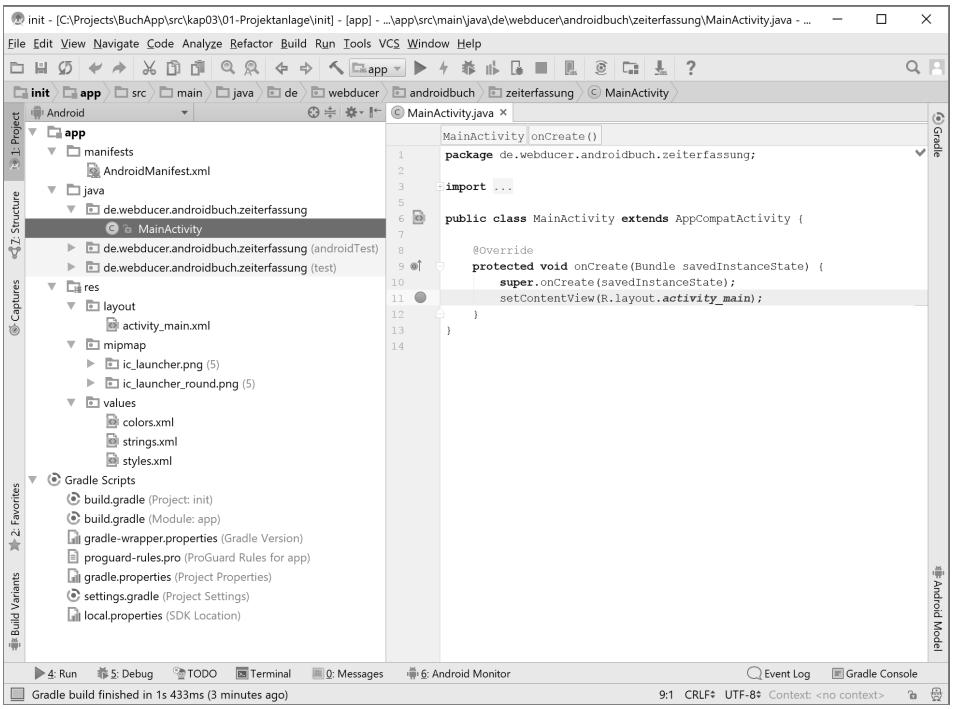

Abb. 1.12: Android Studio mit einen geöffneten Projekt

Android Studio sieht wie die meisten modernen Entwicklungsumgebungen aus. Abhängig vom Betriebssystem befindet sich ganz oben die Menüleiste für den Zugriff auf die Funktionen der Entwicklungsumgebung (unter OS X ist die Menüleiste, wie üblich, nicht direkt im aktiven Fenster).

Gleich darunter kommt die Toolbar mit den sehr oft genutzten Funktionen, die vom Entwickler jederzeit angepasst werden kann. Diese beinhaltet neben den üblichen Buttons zum Speichern, Ausführen und Debuggen der App auch spezielle, die nur die Android-Entwicklung betreffen.

Abb. 1.13: Android-Toolbar

Diese Toolbar erleichtert den Zugriff auf den Android SDK-Manager und Android Virtual Device Manager. Dazu später mehr.

Gleich unterhalb der Toolbar finden Sie die Pfadnavigation zu dem aktuell aktiven Dokument. In dieser sieht man sehr schnell, wo das Dokument physikalisch auf der Festplatte liegt. Weiterhin kann man mit dieser Pfadnavigation auch schnell auf die Dateien/Ordner in derselben Ebene zugreifen, ohne die Projektansicht dazu bemühen zu müssen.

Ganz unten befindet sich die Statusleiste mit den Informationen zum aktuellen Zustand des Projekts und der geöffneten Datei. Während des Build-Vorgangs wird hier auch der Fortschritt angezeigt.

Text-Kodierung

Wenn Sie ein vorhandenes Projekt öffnen und dessen Dateien unverständliche Zeichen enthalten, kann dies an einer falschen Kodierung der Dateien liegen. In der heutigen Programmierung sollten die Dateien als »UTF-8« kodiert vorliegen, womit auch deutsche Umlaute in den Kommentaren und Texten keine Probleme verursachen. Sie sehen die Kodierung der aktuell aktiven Datei immer in der Statusleiste. Auch das Format für Zeilenende (Windows mit `\r\n – CRLF oder` Linux `\n – LF`) ist neben der Textkodierung zu sehen.

Der größte Bereich wird vom Bearbeitungsfenster belegt. Links, rechts und unten finden Sie die Tool-Fenster, die spezielle Aufgaben/Darstellungen übernehmen. Durch einen Klick auf die Überschrift des Fensters wird dieses geöffnet und bleibt offen, bis man ein anderes Tool-Fenster auf derselben Seite anklickt, oder noch einmal auf dieselbe Tool-Fenster-Überschrift klickt. In Abbildung 1.12 ist das Projekt-Tool-Fenster offen.

Tool-Fenster zuklappen

Wenn Sie mehr Platz für das Bearbeitungsfenster haben möchten, weil Sie gerade mit dem Kodieren beschäftigt sind, können Sie alle Tool-Fenster auf einmal zuklappen. Dazu klicken Sie doppelt auf den Tab des aktuell aktiven Dokuments. Ein zweiter Doppelklick auf den Tab stellt alle Tool-Fenster wieder her.

Die wichtigsten Tool-Fenster für die Android-Entwicklung sind:

- **Project**

  Das »Project«-Tool-Fenster stellt die Dateien und Ordner des Projekts in einer Baumansicht dar. Dabei gibt es mehrere Ansichten auf die Projektstruktur.

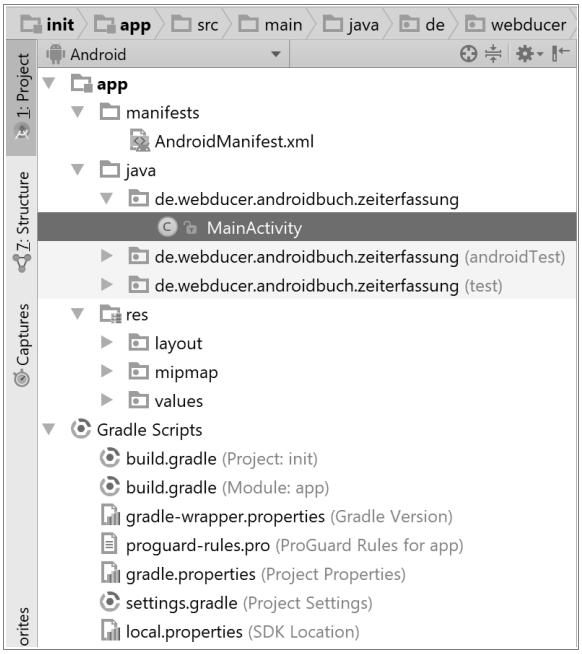

Abb. 1.14: »Project«-Tool-Fenster

- **Structure**

  Das »Structure«-Tool-Fenster zeigt den Aufbau des aktuell aktiven Dokuments. Bei Java-Dateien hat man so sehr schnellen Zugriff auf die Methoden und Klassenvariablen. Bei XML-Dateien sieht man hier die Hierarchie der Elemente.

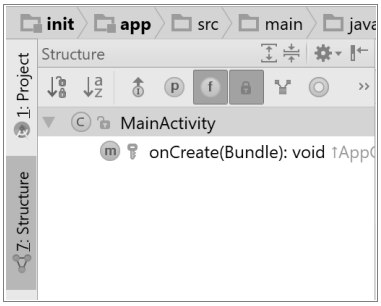

Abb. 1.15: »Structure«-Tool-Fenster

■ Android Monitor

Das »Monitor«-Tool-Fenster erlaubt den Zugriff auf den Log aus dem angeschlossenen Gerät beim Debuggen und die Übersicht über die Prozessor- und Speicher-Nutzung der laufenden App.

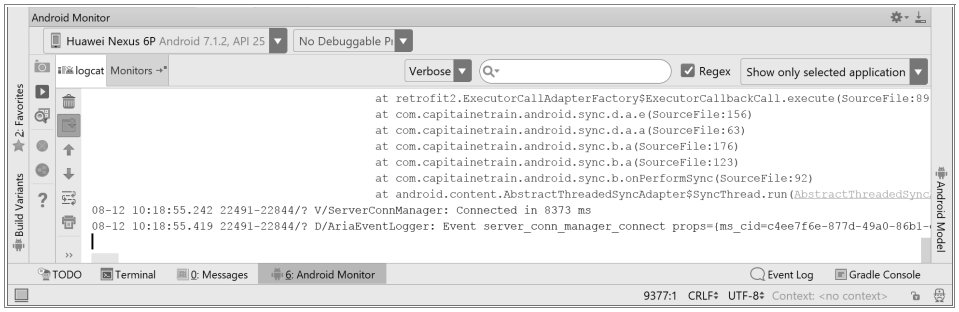

Abb. 1.16: »Android Monitor«-Tool-Fenster mit »logcat«

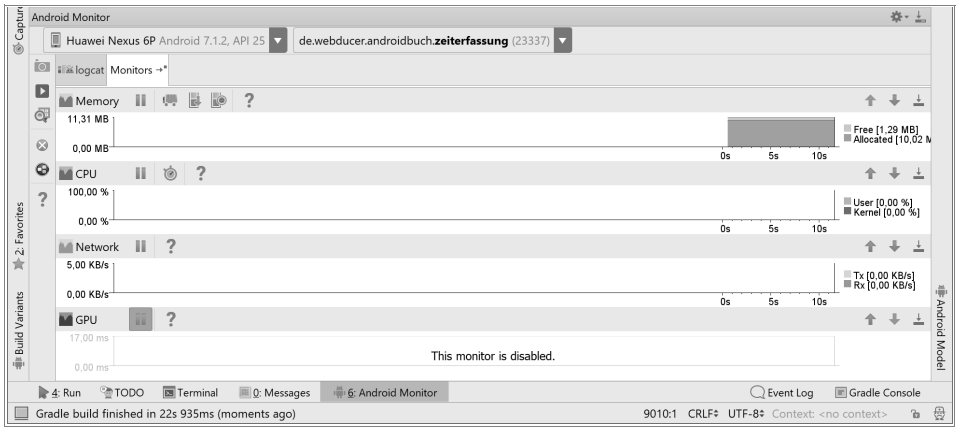

Abb. 1.17: »Android Monitor«-Tool-Fenster mit CPU und Speicher Monitoring

■ Messages

Das »Messages«-Tool-Fenster zeigt die Meldungen der Entwicklungsumgebung. Hier werden auch Fehler angezeigt, falls es bereits bei der Kompilierung der App zu Fehlern gekommen ist.

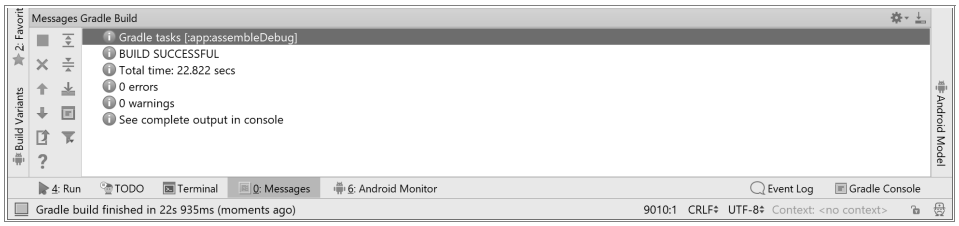

Abb. 1.18: »Messages«-Tool-Fenster

- **Terminal**

  Das »Terminal«-Tool-Fenster erlaubt den Zugriff auf die Kommandozeile, ohne die Entwicklungsumgebung verlassen zu müssen. Es wird zum Beispiel gerne verwendet, um die git-Versionsverwaltung zu nutzen, Android Debug Bridge zu steuern oder einfach ein bestimmtes Skript auszuführen.

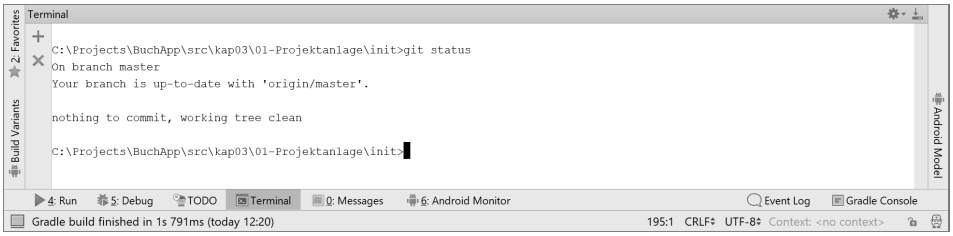

Abb. 1.19: »Terminal«-Tool-Fenster

- **Debug**

  Das »Debug«-Tool-Fenster zeigt die Informationen während einer Debug-Ausführung der App. Hier werden auch die Daten der Variablen angezeigt, die bei einem aktiven »Breakpoint« gültig sind. Es leistet sehr gute Dienste bei der Fehleranalyse.

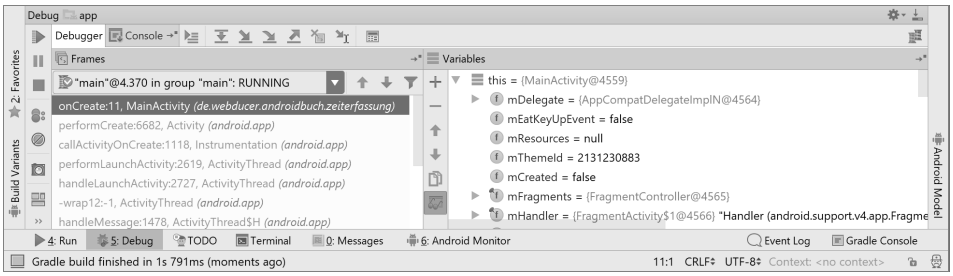

Abb. 1.20: »Debug«-Tool-Fenster in Aktion

1.1.3 Gradle

Die Erstellung von Android-Apps ist sehr kompliziert. Es sind viele Tools beteiligt, um aus allen App-Bestandteilen (Java-Klasse, XML-Dateien, Bilder usw.) eine fertige App zu erstellen. Hier ist ein sehr verkürzter Ablauf:

1. Herunterladen der externen Abhängigkeiten
2. Kompilieren der Java-Dateien in DEX
3. Kompilieren der Ressourcen
4. Optional: Obfuscation mit ProGuard

5. Verpacken als APK-Datei (DEX + Ressourcen)
6. Signieren der APK-Datei
7. Optimierung der fertigen Datei auf Speicherverbrauch

Damit man all diese Schritte nicht manuell durchführen muss, verwendet Android Studio »Gradle« als Build-System.

Gradle ist ein Build-Management-System für Java. Es nutzt eine auf Groovy aufbauende »Domain Specific Language« (kurz DSL), mit der die einzelnen Build-Schritte beschrieben werden. Android liefert mit dem Android-Plugin für Gradle bereits alle Schritte, die für die normale Erstellung notwendig sind, mit. Man kann aber sehr einfach mit eigenen Schritten in den Build-Prozess eingreifen und ihn personalisieren.

Gradle übernimmt neben dem Build selbst auch die Verwaltung der Abhängigkeiten zu externen Bibliotheken. Dazu wird das von der Java-Entwicklung bekannte Maven-Repository eingebunden. Auch Android-Bibliotheken werden als Maven-Artifacts abgebildet.

Wir werden im Laufe des Buchs an einigen Stellen in das Build-Script unserer App eingreifen und dieses anpassen (neue Abhängigkeiten hinzufügen, Version anpassen usw.).

Gradle und Android

Wenn Sie mehr über Gradle wissen möchten, schauen Sie auf die offizielle Homepage des Build-Tools unter `gradle.org`. Weitere Möglichkeiten, das eigene Android-Build anzupassen, finden Sie auf den Entwickler-Seiten für Android unter `developer.android.com/studio/build/`.

1.2 Android Grundbausteine

1.2.1 Projektstruktur

Android-Ansicht

In der Android-Ansicht (Standardansicht nach der Installation von Android Studio) werden Android-spezifische Ressourcen zusammengefasst und als eine einzige »Datei« angezeigt, um die Übersicht zu verbessern.

In Abbildung 1.12 ist das zum Beispiel `res/mipmap/ic_launcher.png`. Hinter dieser Datei steht die Nummer »5«. Diese deutet darauf hin, dass die Datei in fünf unterschiedlichen Ausführungen vorliegt (in diesem Fall ist die Datei in fünf ver-

schiedenen Auflösungen vorhanden). Klappt man die Datei auf, sieht man, um welche Ausführungen es sich handelt. Die Android-Ansicht ist in den meisten Fällen deutlich kompakter, da pro relevanter Datei nur ein Eintrag vorhanden ist.

Nun schauen wir uns die einzelnen Ordner im Projekt und deren Bedeutung an.

app

Das `app`-Verzeichnis steht für das aktuelle Modul. Das erste Modul heißt standardmäßig immer »app«. Wenn Sie später eine App modularisieren, werden auch mehrere Modul-Verzeichnisse auf der obersten Ebene erscheinen.

app/manifest

Im `manifest`-Verzeichnis finden Sie immer die `AndroidManifest.xml` Datei. Diese beschreibt, was die aktuelle App enthält. Hier steht, welche Berechtigungen die App benötigt, welche Fenster diese hat und welche Services diese eventuell zur Verfügung stellt. Somit enthält die Manifest-Datei die Meta-Informationen unserer App, die auch von Play Store beim Hochladen ausgewertet werden.

app/java

Im `java`-Ordner sind die Quellcode-Java-Dateien zu finden, welche die Logik unserer App steuern. Er enthält standardmäßig drei Unterordner – einen für die Geschäftslogik und und zwei für unterschiedliche Test-Typen (mehr dazu in Kapitel 5).

app/res

Im `res`-Ordner befinden sich die von Android verwalteten Ressourcen wie Layouts, Menüs, Bilder usw.

Verwaltet bedeutet bei den Ressourcen, dass das Betriebssystem dafür zuständig ist, die richtigen Ressourcen zu laden (z. B. die richtige Sprache oder ein Bild in der richtigen Auflösung). Aus diesem Grund sind die Unterordner im `res`-Ordner nach einer bestimmten Konvention vorgegeben. Schauen wir uns einige der verwalteten Unterordner an.

app/res/drawable

In diesem Ordner werden die für die App benötigten Bilder und Icons abgelegt. Damit sind sowohl die Pixel-Grafiken, wie PNG oder JPG, als auch Vektorgrafiken gemeint. Meistens werden die Pixel-Grafiken in unterschiedlichen Auflösungen abgelegt, damit das Bild oder das Icon sowohl auf Geräten mit kleiner Auflösung (z. B. nur 160 dpi bei Nexus One) als auch auf hochauflösenden Bildschirmen (z. B. mit 520dpi bei Nexus 6P) gleich groß und scharf angezeigt werden.

app/res/layout

Im `layout`-Ordner liegen die Layout-Dateien für die Fenster (Activity), Fragmente und Oberflächen-Bestandteile (wie Layouts für eine Zeile in der Liste). Oft werden die Layout-Dateien durch die Bildschirmausrichtung (Hoch- oder Querformat) oder minimale Bildschirmbreite (min. 820 dpi) spezialisiert.

app/res/menu

Abhängig von der Vorlage wird auch der `menu`-Ordner in den Ressourcen angelegt. Dieser enthält Dateien für die Definition der Menüs in der Action-Bar oder auch die Kontext-Menüs. In unserem einfachen Beispiel ist dieser Ordner momentan noch nicht vorhanden.

app/res/mipmap

Der `mipmap`-Ordner enthält das Icon für die App, das im Startmenü angezeigt wird. Da die Icons im Startmenü ein wenig größer sind, als die Standard-Icons an anderen Stellen, liegt diese Bilddatei nicht im Standard-Bildordner `app/res/drawable`, sondern in einem eigenen Unterverzeichnis. In den früheren Android-Versionen lag das Start-Icon im `drawable`-Ordner.

app/res/values

Im `values`-Ordner liegen XML-Dateien, die Wertpaare enthalten. Dazu gehören zum Beispiel mit `strings.xml` die Text-Ressourcen, die übersetzt werden können, aber auch Farbdefinitionen mit `colors.xml`, Abmessungen mit `dimens.xml` oder auch Stile mit `styles.xml`. Abhängig vom Basistyp kommen unterschiedliche Spezialisierungen vor, wie Sprachen bei den Text-Ressourcen, oder minimale Breite bei Abmessungen.

Gradle Scripts

Unter `Gradle Scripts` liegen Skripte für die Steuerung des Build-Vorgangs der App. Unter anderem können hier die externen Abhängigkeiten und auch die Versionsnummern der App definiert werden. Genauer sehen wir uns die Skripte an, wenn wir die ersten Änderungen daran vornehmen.

Projekt-Ansicht

Die Projekt-Ansicht entspricht weitestgehend der Struktur, in der die Dateien wirklich auf der Festplatte abgelegt werden. Hier sieht man deutlich, dass es nicht nur einen `mipmap`-Ordner gibt, wie in der Android-Ansicht, sondern fünf. Jeder Ordner ist mit einem Kennzeichen versehen (dazu im nächsten Kapitel mehr). In diesen Ordnern liegt jeweils die Datei `ic_launcher.png` in unterschiedlichen Auflösungen.

Wir werden in diesem Buch in den meisten Fällen die Android-Ansicht nutzen. Sollte eine andere Ansicht erforderlich sein, wird diese explizit angegeben.

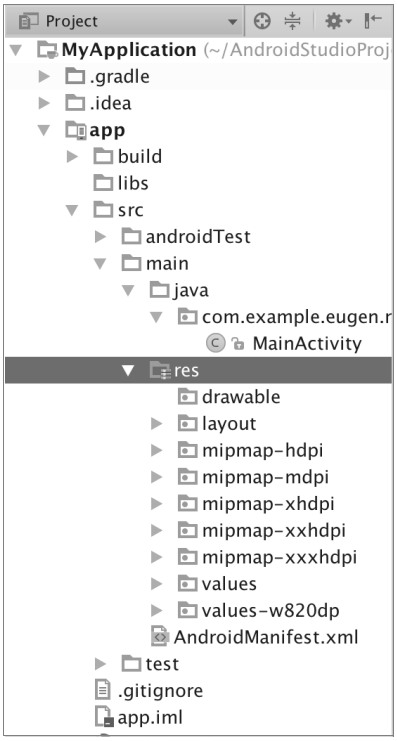

Abb. 1.21: Projekt-Ansicht auf die App

1.2.2 App-Bausteine

Im vorangegangenen Abschnitt wurde die Ansicht des Projekts besprochen, nun wollen wir beleuchten, aus welchen Bausteinen eine App bestehen könnte.

Manifest

Jedes Android-Projekt (unabhängig ob eine App, Widget oder Bibliothek) muss ein Manifest haben. Dieses beschreibt die nach Außen kommunizierende Eigenschaften des Projekts.

Das sind zum Beispiel:

- Activities: sichtbare Bildschirme für den Anwender
- Permissions: Berechtigungen der App
- Service: die »unsichtbaren« Komponenten, die das Projekt bietet
- ContentProvider: Bereitstellung der eigenen Daten an Externe

```
<manifest package="de.webducer.androidbuch.zeiterfassung"
  xmlns:android="http://schemas.android.com/apk/res/android">

  <uses-permission android:name="android.permission.WRITE_EXTERNAL_
                                                          STORAGE" />
  <uses-permission android:name="android.permission.INTERNET" />

  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    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 1.1: Beispiel für Android-Manifest

Activity

Als »Activity« wird unter Android die größte visuelle Einheit bezeichnet. Unter Desktop-Systemen entspricht eine Activity eher einem Fenster. Die Activity ist eine selbständige Einheit, welche 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).

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.

Services

»Services« sind eigenständige Funktionseinheiten unter Android, die aber keine Oberfläche haben. Diese werden oft von Activities aufgerufen, um bestimmte Operation durchzuführen (zum Beispiel Herunterladen einer Datei aus dem Internet).

Content Provider

»Content Provider« sind die Schnittstellen der App nach außen. Dieses Konstrukt ermöglicht es anderen Apps, auf die Daten der eigenen App zuzugreifen. Zum Beispiel ist es über den Content Provider möglich, in einer eigenen App auf die Android-Kontaktdaten zuzugreifen (wenn man die entsprechende Berechtigung hat). Content Provider können durch Berechtigungen geschützt werden.

Berechtigungen

Vor Android 6 (Marshmallow) musste der Benutzer bereits bei der Installation allen »eventuell« notwendigen Berechtigungen zustimmen, auch denen, die er vielleicht nie bei seiner Benutzung brauchen würde.

Seit Android 6 werden die Berechtigungen erst dann angefragt, wenn diese wirklich verwendet werden sollen. Zum Beispiel wird die Berechtigung für den Zugriff auf die Speicherkarte erst, wenn der Benutzer einen Export machen möchte, erbeten. Damit ist dem Benutzer ersichtlich, weshalb die Berechtigung überhaupt benötigt wird.

Folgende Berechtigungen werden in den Apps (nicht immer sinnvoll) angefragt:

- Internet: für Werbung, Bug-Reports oder Synchronisation
- Externer Speicher: für Backups oder Lesen von bestimmten Dateitypen
- Kalender: für Eintragen oder Lesen von Terminen
- Kontakte: sehr beliebt bei Messaging Apps

Broadcast Receiver

»Broadcast Receiver« sind dazu da, um Benachrichtigungen vom Betriebssystem (oder auch von anderen Apps) zu empfangen und zu verarbeiten. Diese Benachrichtigungen werden an alle, die Interesse daran haben, vom Betriebssystem versendet. Zu den gebräuchlichsten Nachrichtentypen gehören zum Beispiel folgende:

- Boot Completed: Android ist gestartet.
- Power Connected: Gerät hängt an einem Ladegerät.
- Power Disconnected: Gerät wurde vom Ladegerät getrennt.

- Battery Low: Akku hat einen niedrigen Stand.
- Connection State Changed: Änderung der Konnektivität (WLAN, 3G, 4G usw.).
- Airplane Mode: Gerät ist im Flugzeug-Modus.

Ressourcen

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

Unter nicht verwalteten Ressourcen versteht man Ressourcen, die nicht vom Betriebssystem unterschieden werden, 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. Verwaltet 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:

- Layout: liegen im Ordner `res/layout`
- Menü: 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.

1.2.3 Layout

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). Nur in Ausnahmefällen sollte man die Oberfläche über den Java-Code aufbauen.

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

1. ViewGroup
2. View

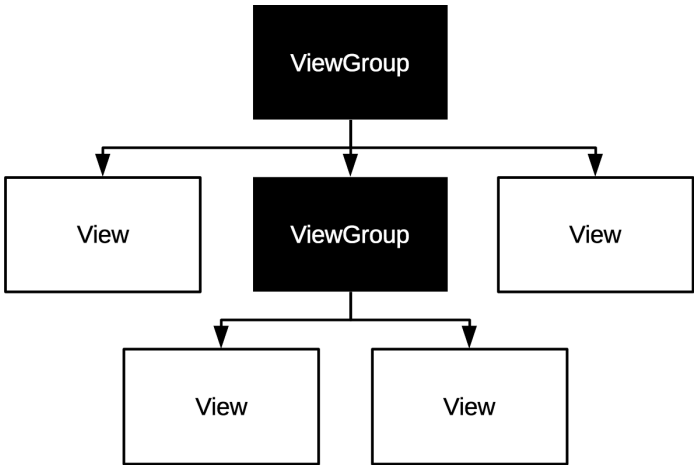

Abb. 1.22: Hierarchischer Aufbau einer Oberfläche aus »ViewGroups« und »Views«

ViewGroup

»ViewGroups« sind Container, die andere »Views« enthalten können (auch weitere ViewGroups). Diese sind für die Anordnung der einzelnen Views 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 eingesetzte ViewGroups:

LinearLayout

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 erlaubt.

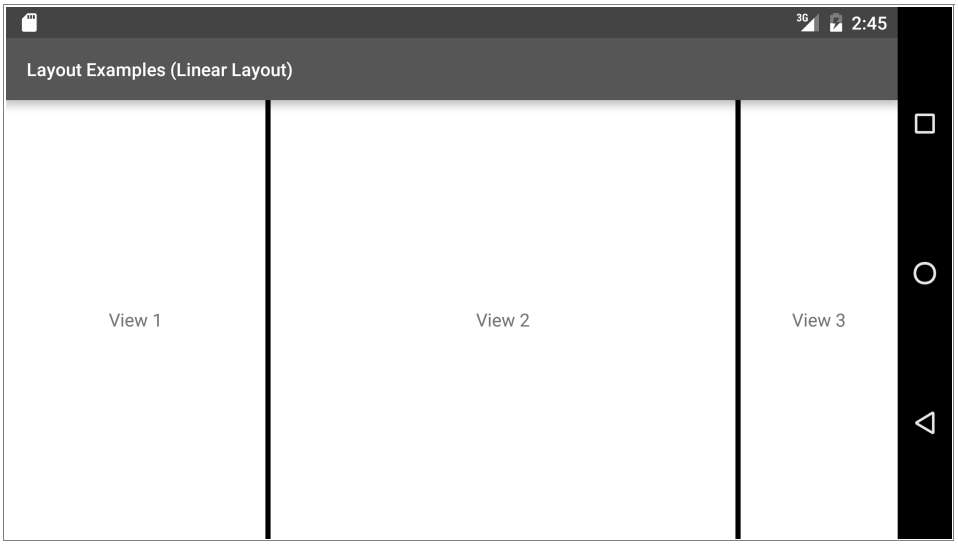

Abb. 1.23: »LinearLayout« mit horizontaler Ausrichtung

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

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal"
    android:weightSum="7">

    <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="4"
        android:gravity="center"
        android:text="View 2" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:gravity="center"
        android:text="View 3" />

</LinearLayout>
```

Listing 1.2: XML-Beispiel für »LinearLayout« mit drei Text-Elementen

Relative Layout

Der Container »RelativeLayout« ordnet die Kind-Elemente relativ zu sich selbst oder zu den anderen Elementen an. Der Container erlaubt sehr komplexe Layouts mit nur einer einzigen Hierarchie-Ebene und ist sehr performant.

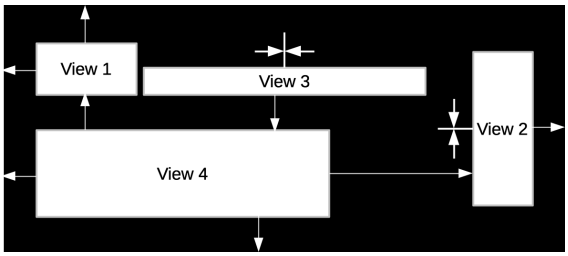

Abb. 1.24: Beziehungen der »Views« zueinander in einen relativen Layout

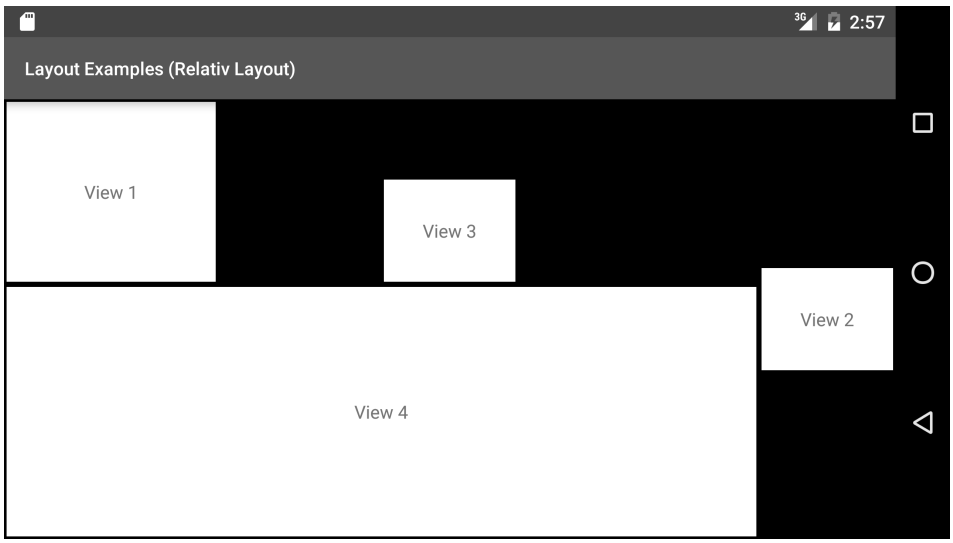

Abb. 1.25: Vorschau des Relative Layouts in Android

Das Verstehen vom Aufbau eines mit Relative Layout erstellten Layouts fällt aber deutlich schwerer als mit Linear Layout. Der Umstand, dass die Reihenfolge der Definition der Kind-Elemente keine Rolle bei der Anordnung der Elemente im Container spielt, führt zur 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 zum Beispiel folgende:

- Element 1:
  - Am linken Rand des Containers
  - Am oberen Rand des Containers
- Element 2:
  - Am rechten Rand des Containers
  - Rechts von »Element 1«
  - Vertikal zentriert im Container
- Element 3:
  - Unterhalb von »Element 2«
  - Vertikal zentriert im Container

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/view1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:gravity="center"
        android:padding="60dp"
        android:text="View 1" />

    <TextView
        android:id="@+id/view2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
```

```
        android:layout_centerVertical="true"
        android:gravity="center"
        android:padding="30dp"
        android:text="View 2" />

    <TextView
        android:id="@+id/view3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/view4"
        android:layout_centerHorizontal="true"
        android:gravity="center"
        android:padding="30dp"
        android:text="View 3" />

    <TextView
        android:id="@+id/view4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/view1"
        android:layout_toLeftOf="@+id/view2"
        android:layout_toStartOf="@+id/view2"
        android:gravity="center"
        android:padding="30dp"
        android:text="View 4" />

</RelativeLayout>
```

Listing 1.3: XML-Beispiel für »RelativeLayout« mit vier Text-Elementen

TableLayout

Der Container »TableLayout« ordnet die Kind-Elemente in Zeilen und Spalten. Die einzelnen Zeilen müssen als ein eigener Container definiert werden. Leider lässt sich im TableLayout keine prozentuale Angabe der einzelnen Spalten definieren. Der längste Wert in einer Zelle einer Spalte bestimmt die Breite. Eine der Spalten kann so definiert werden, dass sie den übrigbleibenden Platz einnimmt (meistens die letzte Spalte).

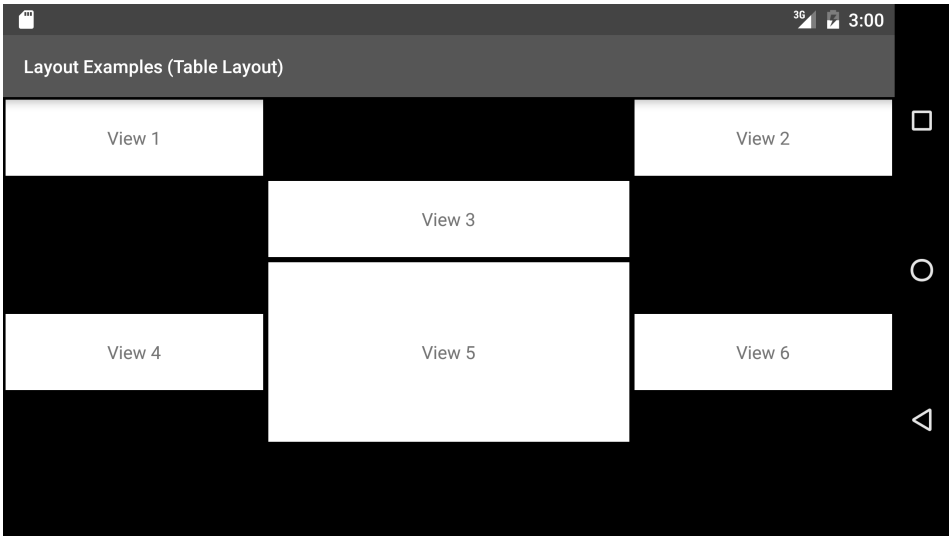

Abb. 1.26: Zeilen und Spalten in einem »TableLayout«

Dieser Container wird oft für Formulare benutzt, um Feld-Beschriftungen bündig (unabhängig von der Übersetzung) auf der linken Seite zu platzieren. Der Container wird mittlerweile nur selten benutzt, da dafür ein flexiblerer Nachfolger, »GridLayout«, vorhanden ist.

Nicht in allen Spalten muss ein Element stehen. Wenn es für das Layout notwendig ist, können auch einige Zellen leer bleiben. Pro Zelle darf nur ein einziges Element platziert werden (aber auch eine ViewGroup).

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:measureWithLargestChild="false"
    android:stretchColumns="1,2,3">

    <TableRow>

        <TextView
            android:layout_column="1"
            android:gravity="center"
            android:padding="20dp"
            android:text="View 1" />
```

```
        <TextView
            android:layout_column="3"
            android:gravity="center"
            android:padding="20dp"
            android:text="View 2" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_column="2"
            android:gravity="center"
            android:padding="20dp"
            android:text="View 3" />
    </TableRow>

    <TableRow>

        <TextView
            android:layout_column="1"
            android:gravity="center"
            android:padding="20dp"
            android:text="View 4" />

        <TextView
            android:layout_column="2"
            android:gravity="center"
            android:padding="60dp"
            android:text="View 5" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_column="3"
            android:gravity="center"
            android:padding="20dp"
            android:text="View 6" />
    </TableRow>

</TableLayout>
```

Listing 1.4: XML-Beispiel für »TableLayout« mit zwei Zeilen und zwei Spalten (vier Text-Elemente)

GridLayout

Der Container »GridLayout« verhält sich ähnlich wie TableLayout, hat aber gegenüber dem TableLayout viele Vorteile.

- Es ist kein Container für die Zeilen notwendig, wie `TableRow` beim TableLayout.
- In einer Zelle können mehrere Views ohne einen weiteren Container platziert werden (z. B. eins oben rechts in der Zelle und eins unten links).
- Für die Zellen kann eine Gewichtung vergeben werden (prozentuale Verteilung).
- Wenn Spalten und Zeilen nicht direkt angegeben werden, können die hinzugefügten Views horizontal oder vertikal das Grid füllen (also von oben nach unten und dann in die nächste Spalte oder von links nach rechts und dann in die nächste Zeile).
- Für Abstände zwischen den Zeilen/Spalten existiert ein eigenes View `Space`, sodass man die Abstände nicht durch die Angabe der Außenabstände (margin) einstellen muss.
- Aufspannen eines Views über mehrere Zeilen/Spalten ist möglich.

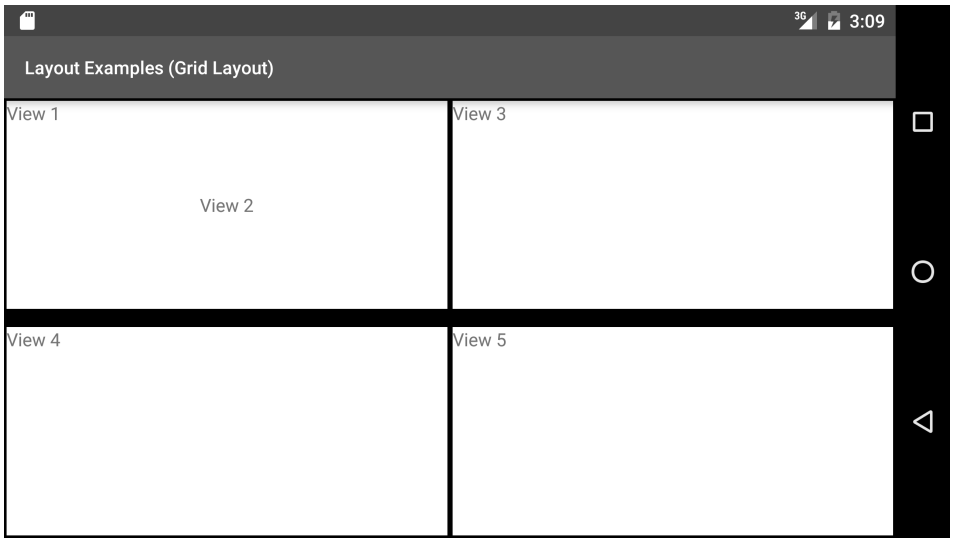

Abb. 1.27: Überlappende Views in einem GridLayout

Somit ist es mit »GridLayout« möglich, sehr komplexe Layouts zu realisieren, ohne viele verschachtelte ViewGroups nutzen zu müssen.

```
<GridLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
```

```
  android:layout_height="match_parent"
  android:columnCount="2"
  android:rowCount="2">

  <TextView
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_column="0"
    android:layout_columnWeight="1"
    android:layout_gravity="fill_horizontal"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:text="View 1"/>

  <TextView
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_column="0"
    android:layout_columnWeight="1"
    android:layout_gravity="fill_horizontal"
    android:layout_row="0"
    android:layout_rowWeight="1"
    android:gravity="center"
    android:text="View 2"/>

  <TextView
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_columnWeight="1"
    android:layout_gravity="fill_horizontal"
    android:layout_rowWeight="1"
    android:text="View 3"/>

  <android.support.v4.widget.Space
    android:layout_width="match_parent"
    android:layout_height="10dp"
    android:layout_columnSpan="2"/>

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

```
    android:layout_height="0dp"
    android:layout_gravity="fill_horizontal"
    android:layout_rowWeight="1"
    android:text="View 4"/>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="0dp"
    android:layout_gravity="fill_horizontal"
    android:layout_rowWeight="1"
    android:text="View 5"/>

</GridLayout>
```

Listing 1.5: XML-Beispiel für »GridLayout« mit zwei Spalten und zwei Zeilen (fünf Text-Elemente)

ListView

»ListView« gehört zu einer besonderen Art der ViewGroups, den Adapter-Views. Diese werden im Normalfall nicht manuell gefüllt. Das Füllen übernehmen Adapter, die mit einer Quelle mit Daten verbunden sind. Aus diesen Daten erzeugen die Adapter dann eine Liste, die diese Daten graphisch repräsentiert. Meistens kommen die Daten aus den Ressourcen (einer festen Liste), Datenbanken und Content Providern.

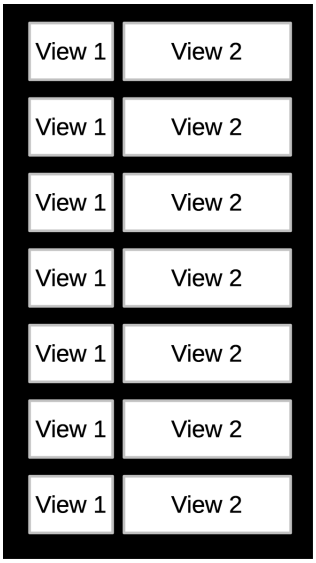

Abb. 1.28: Anordnung der Zeilen in einem »ListView«

```
<ListView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">

</ListView>
```

Listing 1.6: XML-Beispiel für »ListView«

GridView

»GridView« gehört wie »ListView« zu den Adapter-Views. Statt wie bei der Liste die Datensätze untereinander darzustellen, werden diese in einem Gitter angeordnet. Sind alle Spalten der ersten Zeile gefüllt, werden weitere Daten in die nächste Zeile gegeben. Auch hier kommen die Daten meistens von den Ressourcen, Datenbanken und Content Providern.

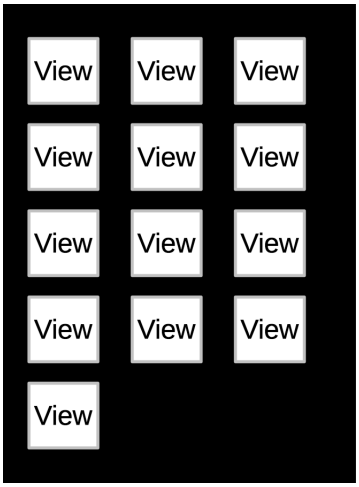

Abb. 1.29: Anordnung der »Views« in einem »GridView«

```
<GridView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:numColumns="3">

</GridView>
```

Listing 1.7: XML-Beispiel für »GridView«

View

Die »Views« sind die einfachen Elemente, die der Benutzer wirklich sieht und mit denen er interagieren kann. Dazu gehören zum Beispiel die nicht bearbeitbaren Texte (unter Android `TextView` genannt), bearbeitbaren Felder (`EditText`), Buttons, aufklappbare Auswahllisten (`Spinner`), einfache und mehrfache Auswahlboxen usw.

Das Standard-Aussehen der Views hängt stark von der Android-Version und dem Hersteller ab. Im Folgenden finden Sie einen kurzen Überblick über oft benutze Views.

TextView

»TextView« stellt ein Element dar, in dem ein einfacher, nicht bearbeitbarer Text angezeigt werden kann. Auch Texte über mehrere Zeilen können mit diesem Element abgebildet werden.

EditText

»EditText« stellt den Text nicht nur dar, sondern kann auch Eingaben vom Benutzer entgegennehmen. Im Normalfall kann beliebiger Text in ein EditText eingegeben werden. Durch den Einsatz des Eingabefilters `android:inputType` kann die Eingabemöglichkeit des Benutzers eingeschränkt werden. Die Filter validieren zum Teil auch die Eingaben und veranlassen das Betriebssystem dazu, passende Tastaturlayouts zu verwenden.

Bei der Eingabe einer E-Mail-Adresse wird zum Beispiel das »@«-Zeichen direkt in der Tastatur angezeigt und nicht erst unter den Sonderzeichen. Bei der Eingabe von Zahlen wird nur die Zahlentastatur eingeblendet.

Folgende (und noch mehr) Typen stellt Android direkt zur Verfügung:

- Text
- Uri
- E-Mail
- Zahlen
- Vorzeichenbehaftete Zahlen
- Passwort
- Dezimalzahlen
- Datum
- Uhrzeit
- Telefonnummer

Button

»Button« ist ein einfaches Element, das für die Interaktion mit dem Benutzer eingesetzt wird. Im Button wird der Text der Aktion abgebildet und die Klick-Aktion wird, wenn alles richtig gemacht wurde, im Java-Code behandelt.

Spinner

Ein »Spinner« ist eine Auswahlbox. In vielen Programmiersprachen wird diese auch »Combobox« genannt. Nach dem Klick auf den Spinner öffnet sich eine Liste mit möglichen Einträgen, von denen nur einer ausgewählt werden kann. Dieser wird im geschlossenen Zustand im Spinner direkt angezeigt.

ImageView

»ImageView« dient der Anzeige von Bildern. Die Bilder können sowohl direkt vom internen Speicher geladen werden als auch aus den verwalteten Ressourcen.

Switch

Ein »Switch« entspricht einem Schalter, der zwischen zwei Positionen hin und her wechselt. Das gleiche Verhalten kann man auch mit einer Checkbox erreichen.

WebView

Mit »WebView« können Sie HTML-Seiten (lokale oder aus dem Internet) in Ihrer App darstellen. Oft wird dies in der App genutzt, um lokal gespeicherte Hilfeseiten (als HTML-Seiten abgelegt) anzuzeigen oder die eigene Homepage innerhalb der App zu präsentieren.

1.3 Zusammenfassung

In diesem Kapitel haben wir die Grundlagen der Android-Programmierung dargestellt.

Die Standard-Entwicklungsumgebung für die Android-Programmierung haben wir installiert und die wichtigsten Komponenten angeschaut.

Wir haben die Grundstruktur einer Android-App angesehen und die Bestandteile von dieser kurz kennengelernt.

Stichwortverzeichnis

N

O

P

Q

R

S