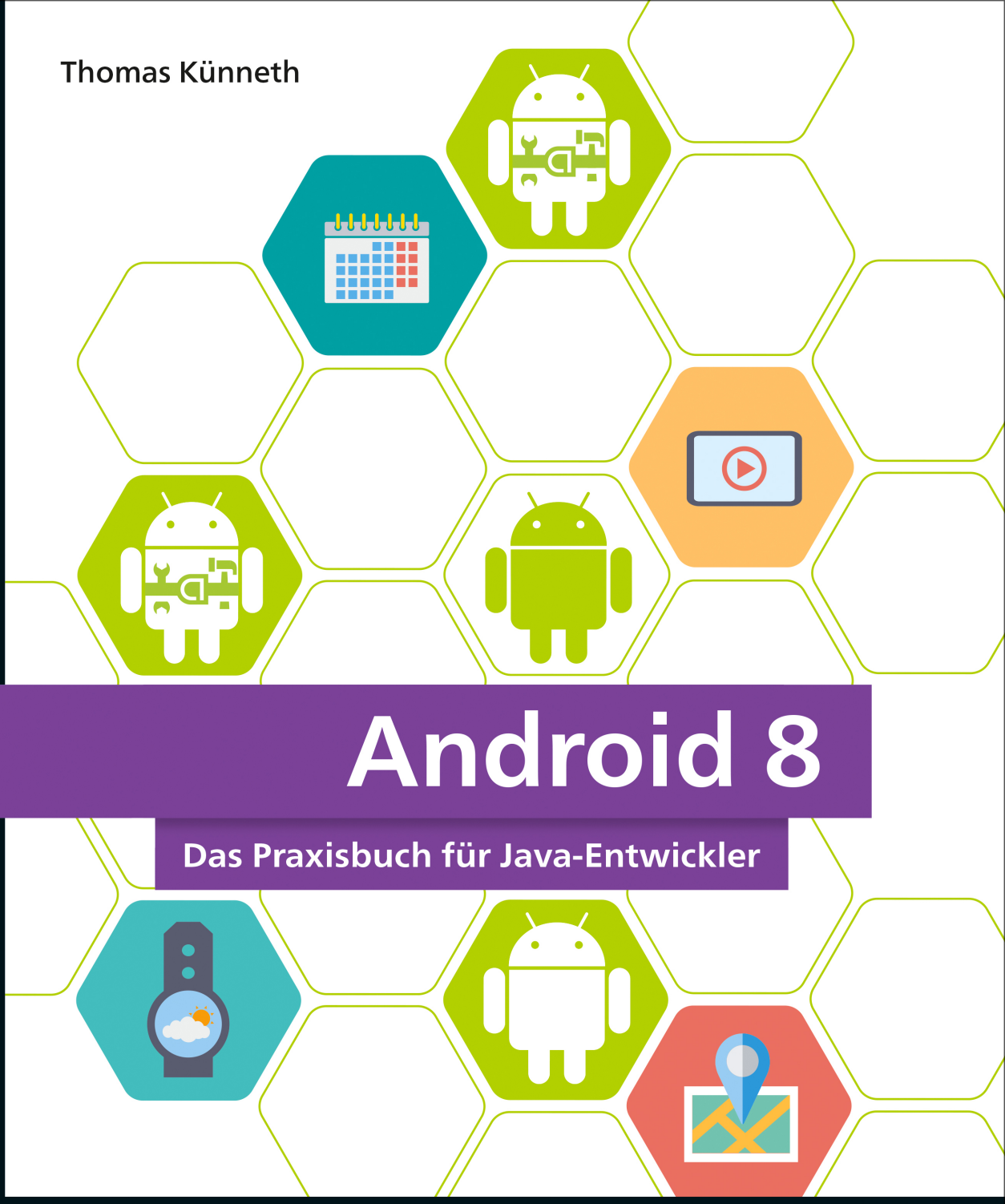
Thomas Künneth



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

Kapitel 2
Hallo Android!

Die erste eigene App ist schneller fertig, als Sie vielleicht glauben.
Dieses Kapitel führt Sie in leicht nachvollziehbaren Schritten zum Ziel.

Seit vielen Jahrzehnten ist es schöne Tradition, anhand des Beispiels »Hello World!« in eine neue Programmiersprache oder Technologie einzuführen. Dahinter steht die Idee, erste Konzepte und Vorgehensweisen in einem kleinen, überschaubaren Rahmen zu demonstrieren. Google bleibt dieser Tradition treu: Wenn Sie in Android Studio ein neues Projekt anlegen, entsteht eine minimale, aber lauffähige Anwendung, die den Text »Hello World« ausgibt. Im Verlauf dieses Kapitels erweitern Sie diese Anwendung um die Möglichkeit, einen Nutzer namentlich zu begrüßen. Ein Klick auf FERTIG schließt die App.

Hinweis

Sie finden die vollständige Version des Projekts *Hallo Android* in den Begleitmaterialien zum Buch, die Sie unter *www.rheinwerk-verlag.de/4564* herunterladen können. Um mit den Entwicklungswerkzeugen vertraut zu werden, rate ich Ihnen aber, sich diese Fassung erst nach der Lektüre dieses Kapitels und nur bei Bedarf zu kopieren.

2.1 Android-Projekte

Alle Projekte fassen Artefakte einer Android-Anwendung zusammen. Dazu gehören unter anderem Quelltexte, Konfigurationsdateien, Testfälle, aber auch Grafiken, Sounds und Animationen. Natürlich sind Projekte keine Erfindung von Android Studio, sondern bilden eines der Kernkonzepte praktisch aller Entwicklungsumgebungen. Grundsätzlich können Sie mit beliebig vielen Projekten gleichzeitig arbeiten. Projekte werden über die Menüleiste angelegt, (erneut) geöffnet und geschlossen. Anders als beispielsweise in Eclipse bezieht sich ein Android-Studio-Hauptfenster stets auf ein Projekt. Wenn Sie ein vorhandenes Projekt öffnen, fragt die IDE normalerweise nach, ob Sie es in einem neuen oder im aktuellen Fenster bearbeiten möchten. Im letzteren Fall wird das aktuelle Projekt geschlossen. Sie können dieses Verhal-

ten übrigens im SETTINGS-Dialog auf der Seite APPEARANCE & BEHAVIOR • SYSTEM SETTINGS unter PROJECT OPENING ändern.

Projekte können aus einem oder mehreren *Modulen* bestehen. Wie Sie in Kapitel 14, »Android Wear«, sehen werden, nutzt Google dieses Konzept beispielsweise, um Projekte für *Android Wear* zu strukturieren. Diese bestehen oft aus einem Teil für das Smartphone oder Tablet sowie einem Teil für die Smartwatch. »Klassische« Android-Apps kommen üblicherweise mit einem Modul aus. In diesem Fall nennt der Assistent das Modul *app*. Beispiel-Apps von Google verwenden als Modulnamen oft *Application*.

2.1.1 Projekte anlegen

Um ein neues Projekt anzulegen, wählen Sie in der Menüleiste des Hauptfensters FILE • NEW • NEW PROJECT. Alternativ können Sie im Willkommensbildschirm auf START A NEW ANDROID STUDIO PROJECT klicken. In beiden Fällen öffnet sich der Assistent CREATE NEW PROJECT, der Sie in wenigen Schritten zu einem neuen Android-Projekt führt. Auf der ersten Seite, CREATE ANDROID PROJECT, legen Sie einige grundlegende Eigenschaften Ihres Projekts fest.

Der APPLICATION NAME – mit ihm identifiziert der Benutzer Ihre App – wird später auf dem Gerät bzw. im Emulator angezeigt. Bitte geben Sie dort »Hallo Android« ein. COMPANY DOMAIN sollte den Namen einer Domain enthalten, die Ihnen gehört (zum Beispiel »thomaskuenneth.com«). Unter PROJECT LOCATION legen Sie den Speicherort Ihres Projekts fest. Es bietet sich an, Projekte an zentraler Stelle zu sammeln. Auf meinem Rechner ist dies *C:\Users\tkuen\Entwicklung\AndroidStudio*. Jedes Projekt entspricht dann einem Unterordner dieses Verzeichnisses, beispielsweise *HalloAndroid*. Der PACKAGE NAME wird automatisch aus APPLICATION NAME und COMPANY DOMAIN zusammengesetzt.

Gefällt Ihnen diese Vorbelegung nicht, können Sie COMPANY DOMAIN auch leer lassen, müssen aber in diesem Fall den PACKAGE NAME direkt eingeben. Klicken Sie hierzu auf das unscheinbare Wort EDIT am rechten Rand des Dialogs. In Java – und damit auch unter Android – werden Klassen und Dateien in Paketen abgelegt. Bei der Vergabe des Paketnamens müssen Sie sorgfältig vorgehen, vor allem, wenn Sie eine Anwendung in *Google Play* veröffentlichen möchten. Denn der Paketname, den Sie hier eintragen, referenziert *genau eine* App, muss also eindeutig sein. Gelegentlich wird der Package Name deshalb auch *Application ID* genannt. Idealerweise folgen Sie den Namenskonventionen für Java-Pakete und tragen in umgekehrter Reihenfolge den Namen einer Ihnen gehörenden Internetdomain ein, gefolgt von einem Punkt und dem Namen der App. Verwenden Sie nur Kleinbuchstaben, und vermeiden Sie

Sonderzeichen, insbesondere das Leerzeichen. Geben Sie für dieses Beispiel als PACKAGE NAME den Text »com.thomaskuenneth.halloandroid« ein, und beenden Sie Ihre Eingabe mit DONE.

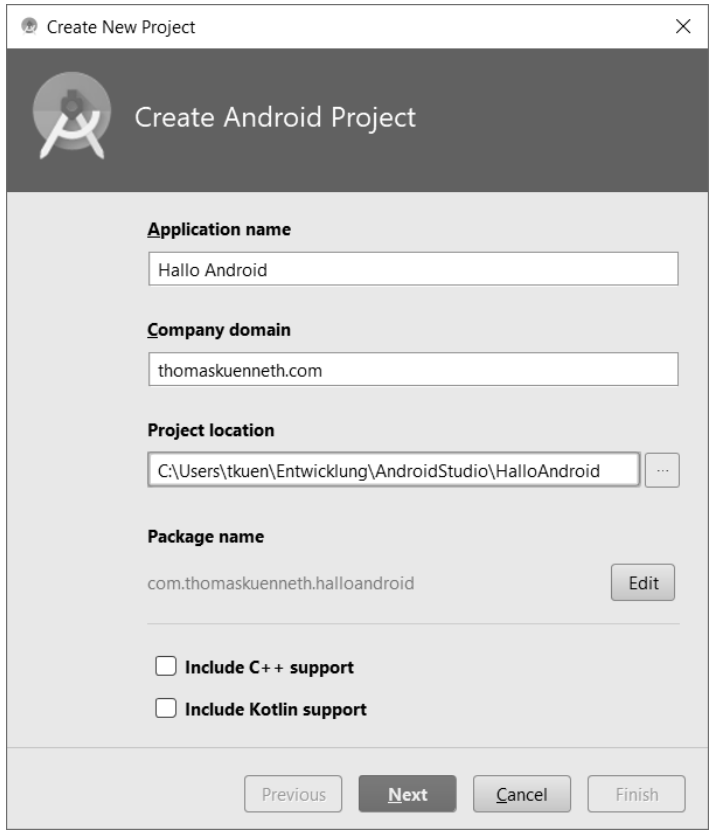


Abbildung 2.1 Der Dialog »Create New Project«

INCLUDE C++ SUPPORT müssen Sie nur dann mit einem Häkchen versehen, wenn Sie in Ihrer App nativen Code verwenden möchten. Meine Beispiele tun dies nicht. INCLUDE KOTLIN SUPPORT legt fest, dass Sie in einem Projekt mit der Programmiersprache *Kotlin* arbeiten möchten. Sie können Java- und Kotlin-Klassen beliebig mischen. Wenn Sie an dieser Stelle ein Häkchen setzen, wird die Klasse, die die Hauptaktivität repräsentiert, in Kotlin erstellt. Lassen Sie für dieses Beispiel die Kotlin-Unterstützung bitte ausgeschaltet. Der Dialog sollte nun in etwa Abbildung 2.1 entsprechen. NEXT bringt Sie auf die Seite TARGET ANDROID DEVICES des Projektassistenten, die Sie in Abbildung 2.2 sehen.

Auf dieser Seite legen Sie die Gerätekategorien fest, für die Ihre App zur Verfügung stehen soll. Möchten Sie beispielsweise ein eigenes Zifferblatt für *Android-Wear*-Smartwatches programmieren, setzen Sie ein Häkchen vor WEAR. Die App *Hallo*

Android soll ausschließlich auf Telefonen und Tablets laufen. Deshalb sollte nur PHONE AND TABLET angekreuzt sein. Wählen Sie als Zielplattform den Wert API 27. Lassen Sie bitte alle anderen Optionen ausgeschaltet. Ein Klick auf NEXT zeigt die Seite ADD AN ACTIVITY TO MOBILE an. Sie ist in Abbildung 2.3 dargestellt.

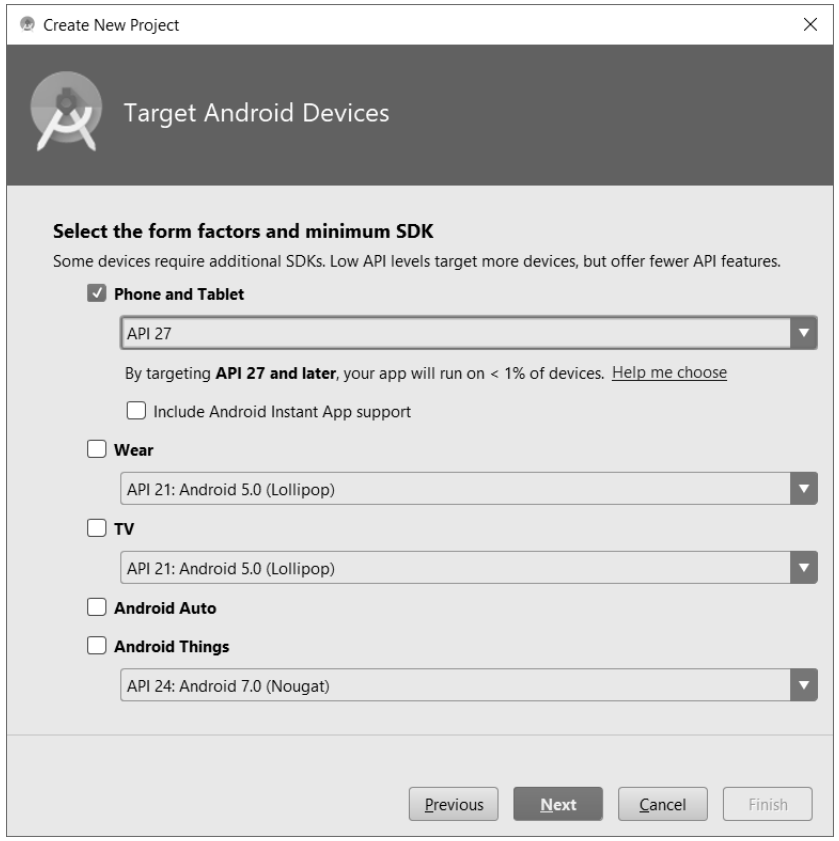


Abbildung 2.2 Geräteklassen auswählen

Hinweis

Stellt Android Studio fest, dass für das erfolgreiche Anlegen eines Projekts noch Komponenten heruntergeladen werden müssen, wird eine »Zwischenseite« eingeschoben. Sollte dies bei Ihnen passieren, können Sie diesen Schritt einfach »durchwinken«.

Activities gehören zu den Grundbausteinen einer Android-Anwendung. Deshalb möchten wir gleich zu Beginn eine anlegen. Markieren Sie EMPTY ACTIVITY, und klicken Sie danach auf NEXT.

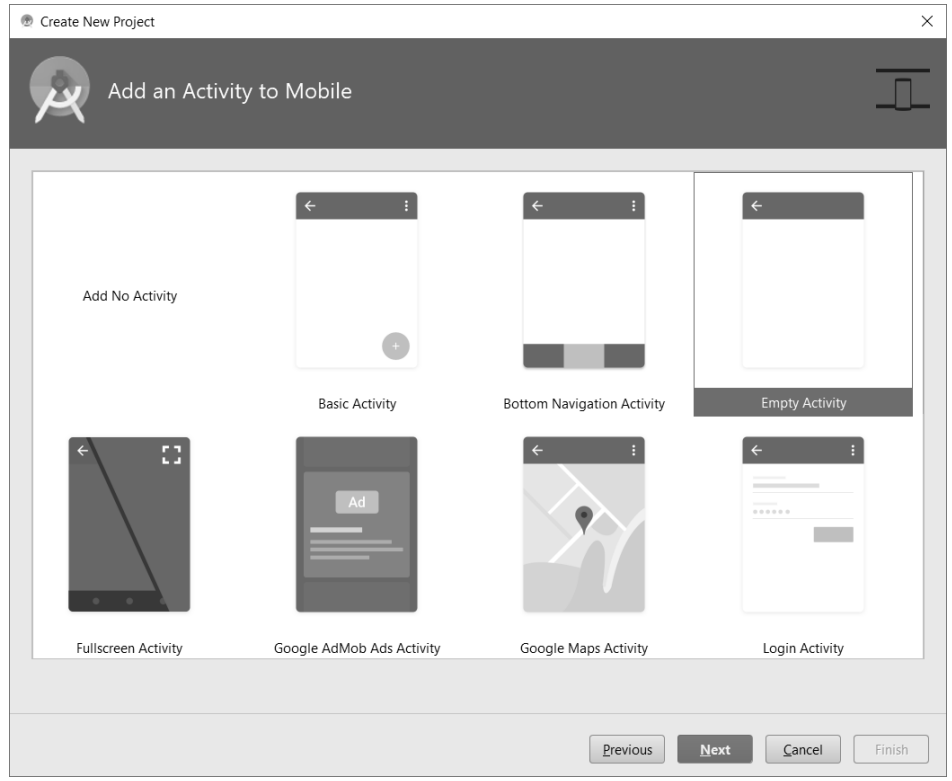


Abbildung 2.3 Die Seite »Add an Activity to Mobile«

Sie haben es fast geschafft: Um den Assistenten abschließen zu können, müssen Sie nur noch auf der Seite CONFIGURE ACTIVITY die soeben ausgewählte leere Activity konfigurieren. Hierbei vergeben Sie einen Klassennamen und den Namen einer Layoutdatei. Was es mit diesen Werten auf sich hat, zeige ich Ihnen im weiteren Verlauf dieses Kapitels. Fürs Erste sollten Sie nur sicherstellen, dass die vorgeschlagenen Werte mit denen in Abbildung 2.4 übereinstimmen. Bitte überprüfen Sie dies, und übernehmen Sie bei Abweichungen die Daten aus dem Screenshot.

Die Checkbox BACKWARDS COMPATIBILITY (APPCOMPAT) steuert, ob der Projektassistent die Bibliothek *AppCompat* einbindet. Damit hat es folgende Bewandtnis: Wann immer Android neue Funktionen erhält, bleiben diese Funktionen Geräten mit entsprechend aktuellen Plattformversionen vorbehalten. Einige davon, zum Beispiel *Fragmente* sowie die *App Bar*, sind aber so essenziell für die Bedienungsphilosophie, dass Google sie auch auf älteren Modellen verwendet sehen möchte. Deshalb werden sie durch AppCompat zur Verfügung gestellt. Allerdings, und das ist der große Haken, nicht aufrufkompatibel zu den »echten« Klassen. Meine eigenen Beispiele setzen, wenn möglich, *nicht* auf AppCompat, sondern nutzen die Originale.

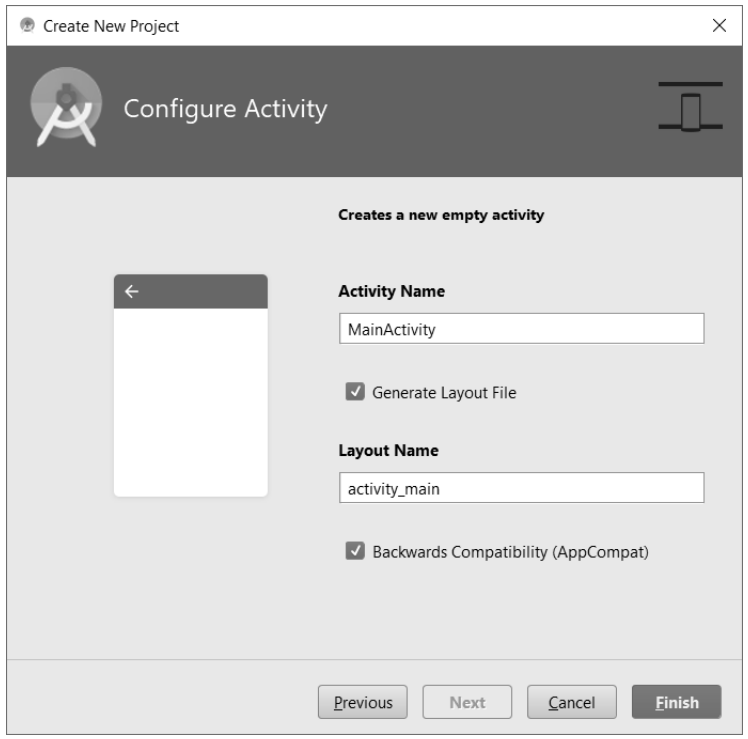


Abbildung 2.4 Eine leere Activity konfigurieren

Mit FINISH schließen Sie den Assistenten. Android Studio wird nun eine Reihe von Dateien anlegen und das neue Projekt einrichten.

Kurzer Rundgang durch Android Studio

Danach sollte das Hauptfenster der IDE in etwa Abbildung 2.5 entsprechen. Es enthält unter anderem eine Menüleiste, eine Toolbar, mehrere Editorfenster für die Eingabe von Java- bzw. Kotlin-Quelltexten und anderen Dateiformaten, einen Designer für die Gestaltung der Benutzeroberfläche, eine Statuszeile sowie mehrere Werkzeugfenster. Beginnt der Name eines solchen Fensters mit einer Ziffer, können Sie es über die Tastatur anzeigen und verbergen. Drücken Sie hierzu die angegebene Zahl zusammen mit der Alt-Taste. Auf dem Mac verwenden Sie cmd.

Werkzeugfenster erscheinen im unteren linken oder rechten Bereich des Hauptfensters. Ihre Position lässt sich über ein Kontextmenü steuern, das Sie durch Anklicken des Fenstertitels mit der rechten Maustaste aufrufen können. Ein Beispiel ist in Abbildung 2.6 zu sehen. Situationsabhängig kann eine ganze Reihe von zusätzlichen Menüpunkten enthalten sein. Die Aufteilung des Android-Studio-Hauptfensters lässt sich praktisch nach Belieben den eigenen Bedürfnissen anpassen. Beispielsweise können Sie Werkzeugfenster als schwebende Panels anzeigen lassen oder bei

Nichtgebrauch automatisch ausblenden. Über das WINDOW-Menü übernehmen Sie Ihre Anpassungen als Standard. RESTORE DEFAULT LAYOUT kehrt zu den zuletzt gespeicherten Standardeinstellungen zurück.

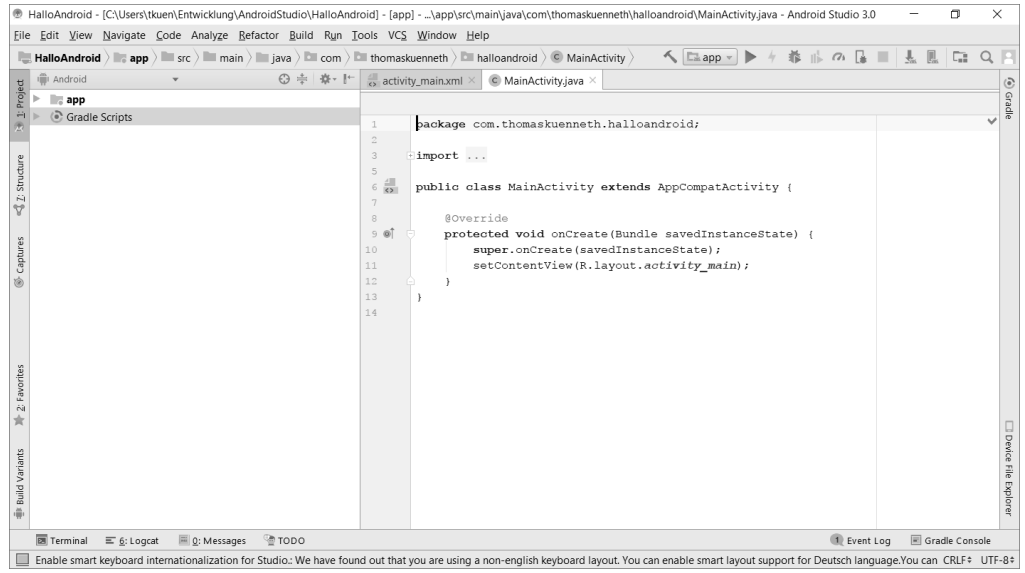


Abbildung 2.5 Das Hauptfenster nach dem Anlegen eines Projekts

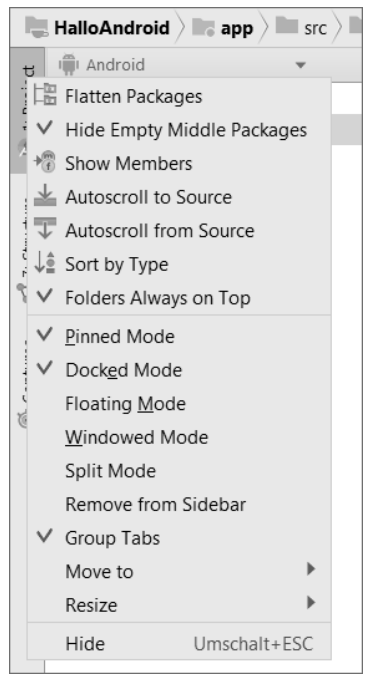


Abbildung 2.6 Kontextmenü eines Werkzeugfensters

Eine Statuszeile am unteren Rand des IDE-Hauptfensters zeigt situationsabhängige Informationen an, beispielsweise die aktuelle Cursorposition oder den Fortschritt eines Build-Vorgangs. Die Statuszeile ist in Abbildung 2.7 zu sehen. Ganz links befindet sich ein Symbol, mit dem Sie drei *Werkzeugfensterbereiche* ein- und ausblenden können. Klicken Sie es mehrere Male an, und achten Sie darauf, wie sich das Android-Studio-Fenster verändert. Lassen Sie sich dabei nicht irritieren, denn sobald Sie mit der Maus über das Symbol fahren, erscheint ein Pop-up-Menü mit allen verfügbaren Werkzeugfenstern. Das ist praktisch, wenn die Werkzeugfensterbereiche nicht sichtbar sind. Innerhalb eines Bereichs können Sie die Reihenfolge der Fenster übrigens per Drag & Drop nach Belieben ändern. Auch das Verschieben in einen anderen Werkzeugfensterbereich ist möglich.

Abbildung 2.7 Die Statuszeile von Android Studio

Der Dialog SETTINGS enthält zahlreiche Optionen, um das Aussehen und Verhalten der IDE Ihren Vorstellungen anzupassen. Sie erreichen ihn über FILE • SETTINGS. Unter macOS finden Sie den Menüpunkt unter ANDROID STUDIO. Öffnen Sie den Knoten APPEARANCE & BEHAVIOR, und klicken Sie dann auf APPEARANCE. Unter den in Abbildung 2.8 gezeigten UI OPTIONS können Sie ein THEME einstellen. DARCULA beispielsweise färbt Android Studio – das Wortspiel lässt es bereits vermuten – dunkel ein. Falls Sie möchten, können Sie die Standardschriften gegen von Ihnen gewählte Fonts austauschen. Setzen Sie hierzu ein Häkchen vor OVERRIDE DEFAULT FONTS BY, und wählen Sie in der Klappliste darunter die gewünschte Schrift und Größe aus.

Klicken Sie im Abschnitt APPEARANCE & BEHAVIOR bitte auf SYSTEM SETTINGS. Unter STARTUP/SHUTDOWN können Sie einstellen, ob beim Start das zuletzt bearbeitete Projekt automatisch geöffnet werden soll. Ist das Häkchen bei REOPEN LAST PROJECT ON STARTUP nicht gesetzt, erscheint der Willkommensbildschirm. Er enthält eine Liste der kürzlich verwendeten Projekte. Das ist praktisch, wenn Sie mit mehreren Projekten im Wechsel arbeiten. Klicken Sie das gewünschte Projekt einfach im Willkommensbildschirm an. CONFIRM APPLICATION EXIT legt fest, ob eine Rückfrage erscheint, wenn Sie Android Studio durch Anklicken des Fensterschließsymbols oder über die Menüleiste verlassen.

Unter PROJECT OPENING können Sie konfigurieren, ob Projekte in einem neuen Android-Studio-Hauptfenster geöffnet werden. Wenn Sie OPEN PROJECT IN THE SAME WINDOW auswählen, schließt die IDE das aktuelle Projekt und öffnet danach das neue. CONFIRM WINDOW TO OPEN PROJECT IN lässt Ihnen in einem entsprechenden Dialog die Wahl.

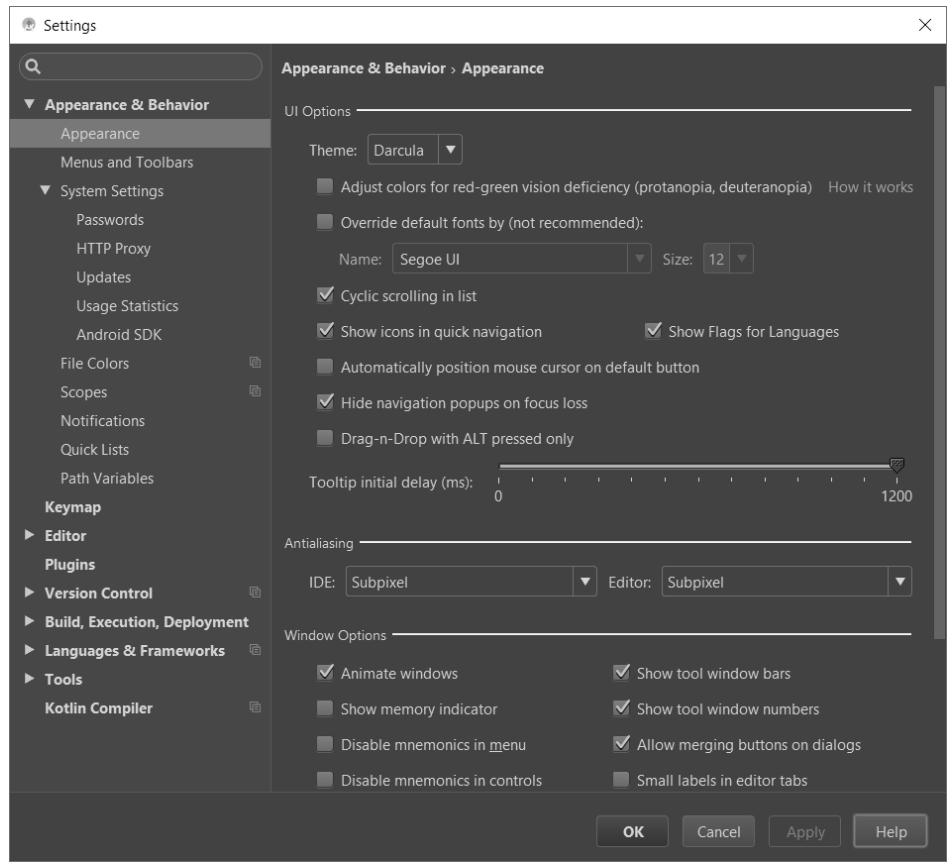


Abbildung 2.8 Der Dialog »Settings«

Da Android Studio kontinuierlich weiterentwickelt und von Fehlern befreit wird, empfehle ich Ihnen die gelegentliche Suche nach Aktualisierungen. Sie können dies zwar mit HELP • CHECK FOR UPDATE selbst auslösen, es ist allerdings bequemer, dies der IDE zu überlassen. Öffnen Sie in den Settings den Knoten APPEARANCE & BEHAVIOR • SYSTEM SETTINGS, und klicken Sie dann auf UPDATES. Sofern dies nicht bereits der Fall ist, aktivieren Sie die Option AUTOMATICALLY CHECK UPDATES FOR. In der Klappliste rechts daneben sollten Sie STABLE CHANNEL auswählen. Kanäle legen fest, welche Aktualisierungen eingespielt werden. Der *Stable Channel* enthält nur ausreichend erprobte Änderungen. Die anderen Kanäle liefern Updates schneller aus, allerdings sind diese oftmals noch fehlerbehaftet oder experimentell.

Damit möchte ich unseren kleinen Rundgang durch Android Studio beenden. Im folgenden Abschnitt stelle ich Ihnen die Struktur von Android-Projekten vor.

2.1.2 Projektstruktur

Android-Apps bestehen aus einer ganzen Reihe von Artefakten, die als baumartige Struktur dargestellt werden können. Das Android-Studio-Werkzeugfenster PROJECT bietet hierfür mehrere Sichten an, unter anderem PROJECT, PACKAGES und ANDROID. Sichten wirken als Filter, d. h., nicht jedes Artefakt (eine Datei oder ein Verzeichnis) ist unbedingt in allen Sichten zu sehen. Die Sicht PROJECT entspricht weitestgehend der Repräsentation auf Ebene des Dateisystems. Sie visualisiert die hierarchische Struktur eines Projekts. PACKAGES gruppiert Dateien analog zu Java-Paketen, soweit dies sinnvoll ist. Diese Sicht werden Sie möglicherweise eher selten verwenden. Am praktischsten für die Entwicklung ist wahrscheinlich die Sicht ANDROID, die in Abbildung 2.9 zu sehen ist.

Die Sicht ANDROID zeigt eine vereinfachte, in Teilen flachgeklopfte Struktur eines Projekts. Sie gestattet den schnellen Zugriff auf wichtige Dateien und Verzeichnisse. Thematisch zusammengehörende Artefakte werden auch dann gemeinsam dargestellt, wenn sie physikalisch in unterschiedlichen Verzeichnissen liegen. Das Werkzeugfenster PROJECT stellt Sichten entweder als Registerkarten oder als Klappliste dar, was Sie mit dem Kommando GROUP TABS im Kontextmenü des Fensters einstellen können. Um es zu öffnen, klicken Sie den Fenstertitel mit der rechten Maustaste an.

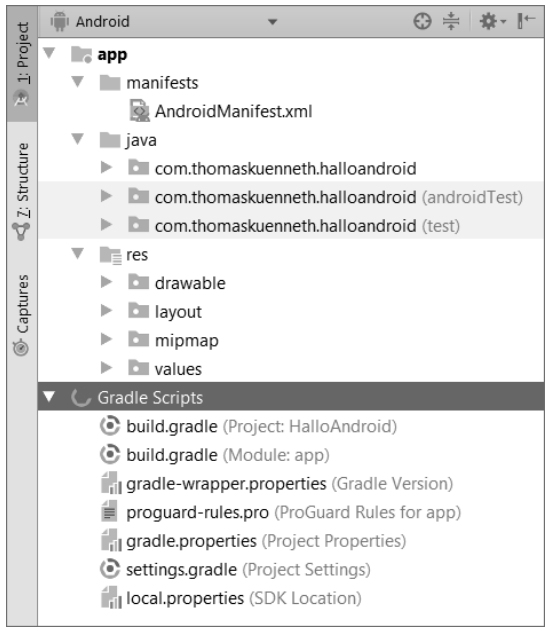


Abbildung 2.9 Die Struktur einer Android-App

Lassen Sie uns nun einen ersten Blick auf wichtige Dateien und Verzeichnisse werfen; aktivieren Sie hierzu die Sicht ANDROID. Sie sehen zwei Knoten, APP und GRADLE

SCRIPTS, von denen Sie bitte den letzteren aufklappen. Die Datei *build.gradle* kommt zweimal vor, die Dateien *gradle.properties*, *settings.gradle* und *local.properties* jeweils einmal. Unter Umständen sehen Sie noch weitere Dateien, zum Beispiel *proguard-rules.pro*. Diese Dateien berühren fortgeschrittene Themen und können fürs Erste außen vor bleiben.

local.properties wird automatisch von Android Studio generiert und sollte nicht von Hand bearbeitet werden. Sie enthält einen Eintrag, der auf das für das Projekt verwendete *Android SDK* verweist. *settings.gradle* listet alle *Module* eines Projekts auf. Unser Hallo-Android-Projekt besteht aus einem Modul: *app*. Die Datei *settings.gradle* wird aktualisiert, sobald ein Modul hinzugefügt oder gelöscht wird. In welchem Zusammenhang Module verwendet werden, zeige ich Ihnen später. Viele Apps benötigen nur ein Modul. Mit *gradle.properties* können Sie Einfluss auf den Build-Vorgang nehmen, zum Beispiel indem Sie Variablen setzen.

Die Datei *build.gradle* ist mehrfach vorhanden. Eine Version bezieht sich auf das Projekt, und zu jedem Modul gehört eine weitere Ausprägung. Da *Hallo Android* aus einem Modul (*app*) besteht, gibt es *build.gradle* also zweimal. Lassen Sie uns einen Blick auf die Version für das Modul *app* werfen: Ein Doppelklick auf BUILD.GRADLE (MODULE: APP) öffnet die Datei in einem Texteditor. Wie das aussehen kann, sehen Sie in Abbildung 2.10.

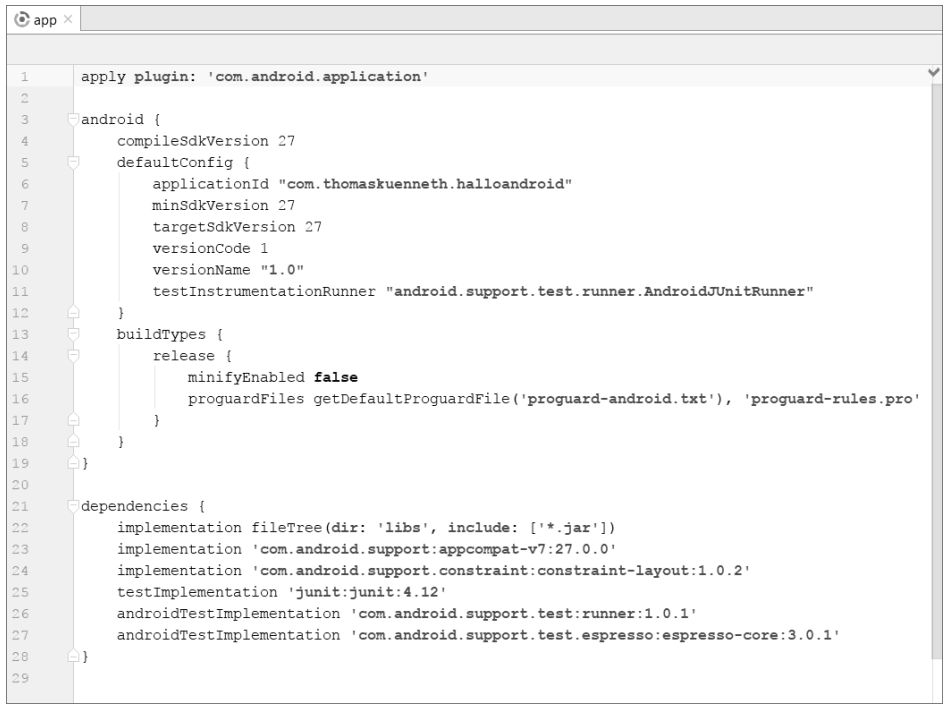


Abbildung 2.10 Die Datei »build.gradle« im Editor von Android Studio

Bitte nehmen Sie zunächst keine Änderungen vor. Sie können das Editorfenster jederzeit durch Anklicken des Kreuzes auf der Registerkarte oder durch Drücken der Tastenkombination Strg+F4 schließen. Auf dem Mac ist es cmd+W.

Der Block `android { … }` enthält Informationen, die Sie beim Anlegen des Projekts eingegeben haben, beispielsweise entspricht `applicationId` dem PACKAGE NAME. `minSdkVersion` gibt an, welche Android-Version auf einem Gerät mindestens vorhanden sein muss, damit man die App nutzen kann. Ist diese Voraussetzung nicht erfüllt, wird die Installation abgebrochen, und *Google Play* zeigt das Programm in so einem Fall gar nicht erst an. Beispielsweise ist erst ab Android 4.x ein Zugriff auf Kalenderdaten über offizielle Schnittstellen möglich. Eine App, die diese nutzt, ist auf sehr alten Geräten mit *Gingerbread* oder gar *Cupcake* nicht lauffähig.

Die `targetSdkVersion` legt fest, gegen welche Plattformversion eine App entwickelt wurde, man könnte auch sagen, unter der sich die App am wohlsten fühlt. Plattformen können mit dem *SDK Manager* installiert und gelöscht werden. Dieses Buch beschreibt die Anwendungsentwicklung mit Android 8.1. Aus diesem Grund basieren die meisten Beispiele auf *API-Level* 27. `versionCode` und `versionName` repräsentieren die Versionsnummer Ihrer App. Wie Sie diese beiden Variablen verwenden, zeige ich Ihnen etwas später. `compileSdkVersion` und `buildToolsVersion` entsprechen dem Versionsstand Ihres Android SDKs.

Tipp

Sie können für die Android-Programmierung eine ganze Reihe von Java-8-Sprachfeatures verwenden, zum Beispiel die codesparenden *Lambda*-Ausdrücke. Die Funktionen werden über *build.gradle* aktiviert. Die Datei im Modul APP muss hierzu die folgenden zusätzlichen Einträge enthalten:

```
android {
  ...
  compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
  }
}
```

Lassen Sie uns nun einen Blick auf das Modul APP werfen; es enthält die Zweige MANIFESTS, JAVA und RES. Quelltexte werden unter JAVA abgelegt. Sie sehen dreimal das Paket `com.thomaskuenneth.halloandroid`. Das mag irritieren, wenn Sie schon mit anderen Entwicklungsumgebungen gearbeitet haben, aber bitte denken Sie daran, dass die Sicht ANDROID eine optimierte und, wenn Sie so möchten, künstliche Sicht auf ein Projekt darstellt. Ein Paket enthält die Klasse `ApplicationTest` oder `ExampleInstrumentedTest`, ein zweites enthält die Klasse `ExampleUnitTest` und das dritte schließlich die Klasse `MainActivity`. Sie haben diesen Namen im Projektassistenten eingetragen. Um die Testklassen müssen Sie sich zunächst nicht kümmern. Übrigens können Sie bequem neue Klassen anlegen, indem Sie ein Paket mit der rechten Maustaste anklicken und NEW • JAVA CLASS wählen.

Der Zweig RES besteht aus mehreren Unterknoten, beispielsweise enthält VALUES die Datei *strings.xml*. Sie nimmt Texte auf, die später im Quelltext oder in Beschreibungsdateien für die Benutzeroberfläche referenziert werden. Hierzu wird von den Werkzeugen des Android SDK eine Klasse mit Namen `R` generiert, die Sie allerdings nicht von Hand bearbeiten dürfen. Deshalb ist sie in der Sicht ANDROID auch nicht vorhanden. Im Unterknoten LAYOUT wird die Benutzeroberfläche einer App definiert. Haben Sie noch ein klein wenig Geduld, wir kommen in diesem Kapitel noch dazu.

Die Knoten DRAWABLE und MIPMAP enthalten Grafiken, die von einer App verwendet werden. Ein Spezialfall eines solchen `Drawable` – so heißt die Klasse, die eine Grafik im Quelltext repräsentiert – ist das Icon für den App-Starter. Google trennt es in seinen Beispielen von den übrigen Grafiken der Anwendung, und der Projektassistent legt die Verzeichnisse *mipmap-…* an. Das heißt, das Programm-Icon liegt in MIPMAP, alle anderen Grafiken liegen in DRAWABLE. Bitmaps können in unterschiedlichen Auflösungen abgelegt werden. Sie landen in Unterverzeichnissen, die einem bestimmten Namensmuster folgen, das ich etwas später erläutern werde. Für Vektorgrafiken – Android kennt auch solche Drawables – ist das Bereitstellen in unterschiedlichen Größen natürlich nicht nötig. Sie landen in DRAWABLE.

Der Unterknoten MANIFESTS enthält die Datei *AndroidManifest.xml*; sie ist die zentrale Beschreibungsdatei einer Anwendung, und in ihr werden unter anderem die Bestandteile des Programms aufgeführt. Wie Sie später noch sehen werden, sind dies sogenannte *Activities*, *Services*, *Broadcast Receiver* und *Content Provider*. Die Datei enthält aber auch Informationen darüber, welche Rechte eine App benötigt und welche Hardware sie erwartet.

Bitte öffnen Sie mit einem Doppelklick die Datei *AndroidManifest.xml*, um sich einen ersten Eindruck von ihrer Struktur zu verschaffen. Es gibt ein Wurzelelement `<manifest>` mit einem Kind `<application>`. Android-Apps bestehen, neben den weiter oben bereits genannten anderen Bausteinen, aus mindestens einer *Activity*. Hierbei handelt es sich, stark vereinfacht ausgedrückt, um eine Bildschirmseite. Verschiedene Aspekte einer Anwendung, wie Listen, Übersichten, Such- und Eingabemasken, werden als eigene Activities realisiert und als Unterelemente von `<application>` in *AndroidManifest.xml* eingetragen.

Hinweis

Wenn Sie einen Blick auf Googles Entwicklerdokumentation zur Manifestdatei werfen, stellen Sie fest, dass es neben `<application>` eine ganze Reihe Kinder von `<manifest>` gibt. Das Tag `<uses-sdk>` gibt beispielsweise die Zielplattform an. Seit dem Wechsel von Eclipse auf Android Studio werden diese Angaben aber nicht mehr direkt in das Manifest eingetragen, sondern in *build.gradle* gepflegt und beim Bauen der Anwendung werden sie dann automatisch in das Manifest übernommen.

Im nächsten Abschnitt werden Sie erste Erweiterungen an *Hallo Android* vornehmen. Zunächst werde ich Ihnen zeigen, wie in Android Texte gespeichert werden und wie man in einer App auf diese zugreift.

2.2 Benutzeroberfläche

Die Benutzeroberfläche ist das Aushängeschild einer Anwendung. Gerade auf mobilen Geräten mit vergleichsweise kleinen Bildschirmen sollte jede Funktion leicht zugänglich und intuitiv erfassbar sein. Android unterstützt Sie bei der Gestaltung durch eine große Auswahl an Bedienelementen.

2.2.1 Texte

Bilder und Symbole sind ein wichtiges Gestaltungsmittel. Sinnvoll eingesetzt, helfen sie dem Anwender nicht nur beim Bedienen des Programms, sondern sorgen zudem für ein angenehmes und schönes Äußeres. Dennoch spielen auch Texte eine sehr wichtige Rolle. Sie werden in den unterschiedlichsten Bereichen einer Anwendung eingesetzt:

- als Beschriftungen von Bedienelementen
- für erläuternde Texte, die durch einen Screenreader vorgelesen werden
- für Hinweis- und Statusmeldungen

Die fertige Version von *Hallo Android* soll den Benutzer zunächst begrüßen und ihn nach seinem Namen fragen. Im Anschluss wird ein persönlicher Gruß angezeigt. Nach dem Anklicken einer Schaltfläche beendet sich die App.

Aus dieser Beschreibung ergeben sich die folgenden Texte. Die Bezeichner vor dem jeweiligen Text werden Sie später im Programm wiederfinden:

- `willkommen` – *Guten Tag. Schön, dass Sie mich gestartet haben. Bitte verraten Sie mir Ihren Namen.*
- `weiter` – *Weiter*

- `hallo` – *Hallo <Platzhalter>. Ich freue mich, Sie kennenzulernen.*
- `fertig` – *Fertig*

Ein Großteil der Texte wird zur Laufzeit so ausgegeben, wie sie schon während der Programmierung erfasst wurden. Eine kleine Ausnahme bildet die Grußformel, denn sie besteht aus einem konstanten und einem variablen Teil. Letzterer ergibt sich erst, nachdem der Anwender seinen Namen eingetippt hat. Wie Sie gleich sehen werden, ist es in Android sehr einfach, dies zu realisieren.

Da Sie Apps in Java schreiben, könnten Sie die auszugebenden Meldungen einfach im Quelltext ablegen. Das sähe folgendermaßen aus (ich habe die `String`-Konstante aus Gründen der besseren Lesbarkeit in drei Teile zerlegt):

```
nachricht.setText("Guten Tag. Schön, dass" +
        " Sie mich gestartet haben." +
        " Bitte verraten Sie mir Ihren Namen.");
```

Das hätte allerdings eine ganze Reihe von Nachteilen: Da jede Klasse in einer eigenen Datei abgelegt wird, merkt man oft nicht, wenn man gleiche Texte mehrfach definiert, was die Installationsdatei der App vergrößert und unnötig Speicher kostet. Außerdem wird es auf diese Weise sehr schwer, mehrsprachige Anwendungen zu bauen. Wenn Sie aber eine App über Google Play vertreiben möchten, sollten Sie neben den deutschsprachigen Texten mindestens eine englische Lokalisierung ausliefern.

Unter Android werden Texte daher zentral in der Datei *strings.xml* abgelegt, die sich im Verzeichnis *values* befindet. Ändern Sie die durch den Projektassistenten angelegte Fassung folgendermaßen ab:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!-- Name der App -->
    <string name="app_name">Hallo Android!</string>

    <!-- Willkommensmeldung -->
    <string name="willkommen">
Guten Tag. Schön, dass Sie mich gestartet haben.
Bitte verraten Sie mir Ihren Namen.
    </string>

    <!-- persönlicher Gruß -->
    <string name="hallo">
Hallo %1$s. Ich freue mich, Sie kennenzulernen.
    </string>
```

```
    <!-- Beschriftungen für Schaltflächen -->
    <string name="weiter">Weiter</string>
    <string name="fertig">Fertig</string>

</resources>
```

Listing 2.1 »strings.xml«

Das Attribut `name` des Elements `<string>` wird später im Quelltext als Bezeichner verwendet. Der Name muss also projektweit eindeutig sein. Ist Ihnen im Listing die fett gesetzte Zeichenfolge `%1$s` aufgefallen? Android wird an dieser Stelle den vom Benutzer eingegebenen Namen einfügen. Wie dies funktioniert, zeige ich Ihnen später.

Hinweis

Die Zeilen `Hallo %1$s...` und `Guten Tag.` sind nicht eingerückt, weil die führenden Leerzeichen sonst in die App übernommen werden, was in der Regel nicht gewünscht ist.

Vielleicht fragen Sie sich, wie Sie Ihr Programm mehrsprachig ausliefern können, wenn es genau eine zentrale Datei *strings.xml* gibt. Neben dem Verzeichnis *values* kann es lokalisierte Ausprägungen geben, die auf das Minuszeichen und auf ein Sprachkürzel aus zwei Buchstaben enden, zum Beispiel *values-en* oder *values-fr*. Die Datei *strings.xml* in diesen Ordnern enthält Texte in den korrespondierenden Sprachen, also auf Englisch oder Französisch. Muss Android auf eine Zeichenkette zugreifen, geht das System vom Speziellen zum Allgemeinen. Ist die Standardsprache also beispielsweise Englisch, wird zuerst versucht, den Text in *values-en/strings.xml* zu finden. Gelingt dies nicht, wird *values/strings.xml* verwendet. In dieser Datei müssen also alle Strings definiert werden, Lokalisierungen hingegen können unvollständig sein.

Im folgenden Abschnitt stelle ich Ihnen sogenannte *Views* vor. Bei ihnen handelt es sich um die Grundbausteine, aus denen die Benutzeroberfläche einer App zusammengesetzt wird.

2.2.2 Views

Hallo Android besteht auch nach vollständiger Realisierung aus sehr wenigen Bedienelementen, und zwar aus

- einem nicht editierbaren Textfeld, das den Gruß unmittelbar nach dem Programmstart sowie nach Eingabe des Namens darstellt,
- einer Schaltfläche, die je nach Situation mit WEITER oder FERTIG beschriftet ist, und aus

- einem Eingabefeld, das nach dem Anklicken der Schaltfläche WEITER ausgeblendet wird.

Wie die Komponenten auf dem Bildschirm platziert werden sollen, zeigt ein sogenannter *Wireframe*, den Sie in Abbildung 2.11 sehen. Man verwendet solche abstrakten Darstellungen gern, um die logische Struktur einer Bedienoberfläche in das Zentrum des Interesses zu rücken.

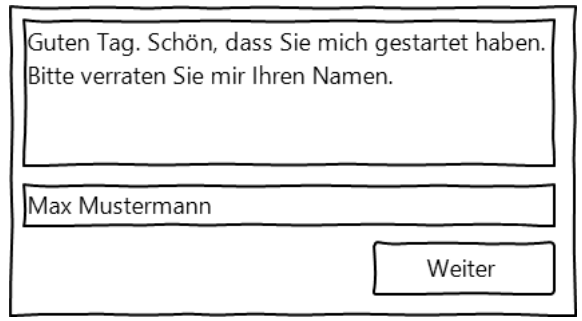


Abbildung 2.11 Prototyp der Benutzeroberfläche von »Hallo Android«

Unter Android sind alle Bedienelemente direkte oder indirekte Unterklassen der Klasse `android.view.View`. Jede View belegt einen rechteckigen Bereich des Bildschirms. Seine Position und Größe wird durch Layouts bestimmt, die wiederum von `android.view.ViewGroup` erben, die ebenfalls ein Kind von `View` ist. Sie haben üblicherweise keine eigene grafische Repräsentation, sondern sind Container für weitere Views und ViewGroups.

Die Text- und Eingabefelder sowie die Schaltflächen, die in *Hallo Android* verwendet werden, sind also Views. Konkret verwenden wir die Klassen `Button`, `TextView` und `EditText`. Wo sie auf dem Bildschirm positioniert werden und wie groß sie sind, wird hingegen durch die ViewGroup `LinearLayout` festgelegt.

Zur Laufzeit einer Anwendung manifestiert sich ihre Benutzeroberfläche demnach als Objektbaum. Aber nach welcher Regel wird er erzeugt? Wie definieren Sie als Entwickler den Zusammenhang zwischen einem Layout, einem Textfeld und einer Schaltfläche? Java-Programmierer sind gewohnt, die Oberfläche programmatisch zusammenzusetzen. Nicht nur im Swing-Umfeld finden sich unzählige Ausdrücke im Stil von

```
JPanel p = new JPanel();
JButton b = new JButton();
p.add(b);
```

Auch unter Android könnten Sie die Bedienelemente auf diese Weise zusammenfügen:

```
ScrollView v = new ScrollView(context);
LinearLayout layout = new LinearLayout(context);
layout.setOrientation(LinearLayout.VERTICAL);
v.addView(layout);
layout.addView(getCheckbox(context, Locale.GERMANY));
layout.addView(getCheckbox(context, Locale.US));
layout.addView(getCheckbox(context, Locale.FRANCE));
```

Listing 2.2 Beispiel für den programmgesteuerten Bau einer Oberfläche

Allerdings ist dies nicht die typische Vorgehensweise; diese lernen Sie im folgenden Abschnitt kennen.

2.2.3 Oberflächenbeschreibungen

Eine Android-Anwendung beschreibt ihre Benutzeroberflächen normalerweise mittels XML-basierter Layoutdateien. Diese werden zur Laufzeit der App zu Objektbäumen »aufgeblasen«. Alle Bedienelemente von *Hallo Android* werden in einen Container des Typs `LinearLayout` gepackt. Seine Kinder erscheinen entweder neben- oder untereinander auf dem Bildschirm. Wie Sie gleich sehen werden, steuert das Attribut `android:orientation` die Laufrichtung. Für die Größe der Views und ViewGroups gibt es `android:layout_width` und `android:layout_height`.

Oberflächenbeschreibungen werden in *layout*, einem Unterverzeichnis von *res*, gespeichert. Beim Anlegen des Projekts hat der Android-Studio-Projektassistent dort die Datei *activity_main.xml* abgelegt. Öffnen Sie diese mittels Doppelklick, und ändern Sie sie entsprechend Listing 2.3 ab. Damit Sie den Quelltext eingeben können, klicken Sie auf die Registerkarte Text am unteren Rand des Editors.

```
<?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:orientation="vertical">

  <TextView
    android:id="@+id/nachricht"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

  <EditText
    android:id="@+id/eingabe"
    android:layout_width="match_parent"
```



```
    android:layout_height="wrap_content" />

  <Button
    android:id="@+id/weiter_fertig"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end" />

</LinearLayout>
```

Listing 2.3 »activity_main.xml«

Die XML-Datei bildet die Hierarchie der Benutzeroberfläche ab. Demzufolge ist `<LinearLayout>` das Wurzelelement. Mein Beispiel enthält die drei Kinder `<TextView>`, `<EditText>` und `<Button>`. Jedes Element hat die bereits kurz angesprochenen Attribute `android:layout_width` und `android:layout_height`. Deren Wert `match_parent` besagt, dass die Komponente die Breite oder Höhe des Elternobjekts erben soll. Der Wert `wrap_content` hingegen bedeutet, dass sich die Größe aus dem Inhalt der View ergibt, beispielsweise aus der Beschriftung einer Schaltfläche. Die Zeile `android:layout_gravity="end"` sorgt dafür, dass die Schaltfläche rechtsbündig angeordnet wird.

Tipp

Anstelle von `match_parent` finden Sie im Internet oft noch die ältere Notation `fill_parent`. Diese wurde schon in Android 2.2 (API-Level 8) von `match_parent` abgelöst. Für welche Variante Sie sich entscheiden, ist nur von Belang, wenn Sie für sehr alte Plattformversionen entwickeln. Denn abgesehen vom Namen sind beide identisch. Ich rate Ihnen, stets `match_parent` zu verwenden.

Ist Ihnen aufgefallen, dass keinem Bedienelement ein Text oder eine Beschriftung zugewiesen wird? Und was bedeuten Zeilen, die mit `android:id="@+id/` beginnen? Wie Sie bereits wissen, erzeugt Android zur Laufzeit einer Anwendung aus den Oberflächenbeschreibungen entsprechende Objektbäume. Zu der in der XML-Datei spezifizierten Schaltfläche gibt es also eine Instanz der Klasse `Button`.

Um auf diese Instanz eine Referenz ermitteln zu können, wird ein Name definiert, beispielsweise `weiter_fertig`. Wie auch bei *strings.xml* sorgen die Android-Entwicklungswerkzeuge dafür, dass nach Änderungen an Layoutdateien korrespondierende Einträge in der generierten Klasse `R` vorgenommen werden. Wie Sie diese nutzen, sehen Sie gleich.

Speichern Sie Ihre Eingaben, und wechseln Sie zurück zum grafischen Editor, indem Sie auf die Registerkarte Design klicken. Er sollte in etwa so wie in Abbildung 2.12

aussehen. Machen Sie sich über die angezeigte Warnung keine Gedanken, wir kümmern uns etwas später darum.

Hinweis

In XML-Dateien nutzt Google gern den Underscore als verbindendes Element, zum Beispiel in `layout_width`, `layout_height` oder `match_parent`. Sie sollten diesem Stil folgen. Aus diesem Grund habe ich die ID der Schaltfläche zum Weiterklicken und Beenden der App `weiter_fertig` genannt. In Java-Quelltexten ist aber die sogenannte *CamelCase*-Schreibweise gebräuchlich, deshalb heißt die Variable der Schaltfläche `weiterFertig`.

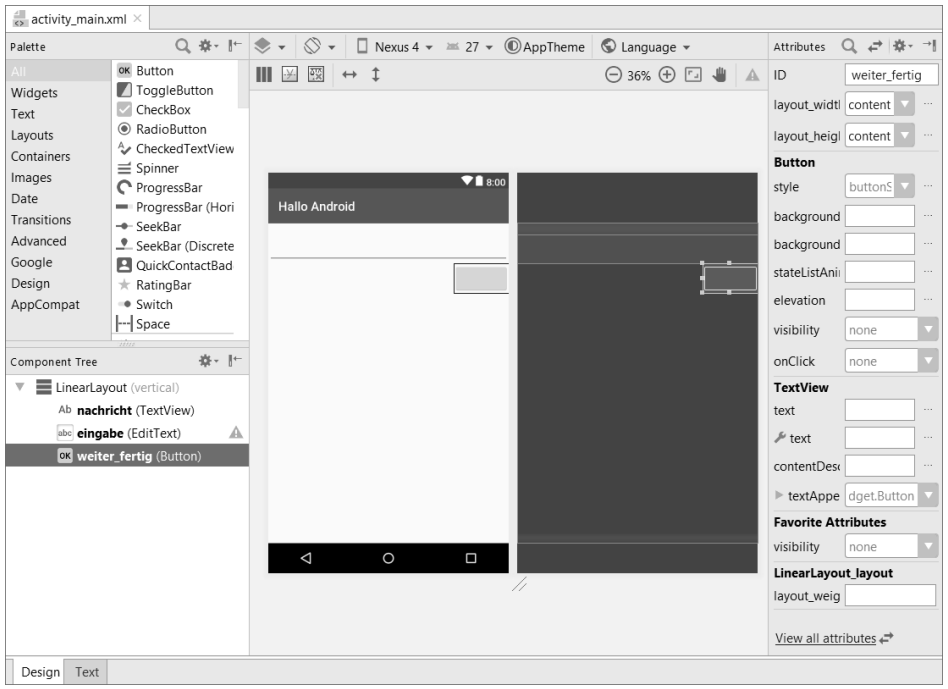


Abbildung 2.12 Vorschau der Benutzeroberfläche im grafischen Editor

2.3 Programmlogik und -ablauf

Viele Desktop-Anwendungen sind datei- oder dokumentenzentriert. Egal, ob Textverarbeitung, Tabellenkalkulation oder Layoutprogramm – ihr Aufbau ist stets gleich. Den überwiegenden Teil des Bildschirms oder Fensters belegt ein Arbeitsbereich, der ein Dokument oder einen Teil davon darstellt. Um diesen Bereich herum gruppieren sich Symbolleisten und Paletten, mit deren Werkzeugen die Elemente des Dokuments bearbeitet werden.

Das gleichzeitige Darstellen von Werkzeugen und Inhalt ist auf den vergleichsweise kleinen Bildschirmen mobiler Geräte nur bedingt sinnvoll, denn der Benutzer würde dabei kaum etwas erkennen. Als Entwickler müssen Sie Ihre Anwendung deshalb in Funktionsblöcke oder Bereiche unterteilen, die genau einen Aspekt Ihres Programms abbilden.

Ein anderes Beispiel: E-Mail-Clients zeigen die wichtigsten Informationen zu eingegangenen Nachrichten häufig in einer Liste an. Neben oder unter der Liste befindet sich ein Lesebereich, der das aktuell ausgewählte Element vollständig anzeigt. Auch dies lässt sich aufgrund des geringen Platzes auf Smartphones nicht sinnvoll realisieren. Stattdessen zeigen entsprechende Anwendungen dem Nutzer zunächst eine Übersicht, nämlich die Liste der eingegangenen Nachrichten, und verzweigen erst in eine Detailansicht, wenn eine Zeile der Liste angeklickt wird.

Hinweis

Manche Tablets bieten riesige Bildschirme, auf denen deutlich mehr Informationen dargestellt werden können als auf Smartphones. Um Benutzeroberflächen für beide Welten entwickeln zu können, hat Google mit Android 3 die sogenannten *Fragmente* eingeführt. Im nächsten Kapitel zeige ich Ihnen, wie Sie Benutzeroberflächen mithilfe von Fragmenten für unterschiedliche Bildschirmgrößen anbieten.

2.3.1 Activities

Unter Android ist das Zerlegen einer App in aufgabenorientierte Teile bzw. Funktionsblöcke ein grundlegendes Architekturmuster. Die gerade eben skizzierten Aufgaben bzw. »Aktivitäten« *E-Mail auswählen* und *E-Mail anzeigen* werden zu Bausteinen, die die Plattform *Activities* nennt. Eine Anwendung besteht aus mindestens einer solchen Activity, je nach Funktionsumfang können es aber auch viele mehr sein. Normalerweise ist jeder Activity eine Benutzeroberfläche, also ein Baum, zugeordnet, der aus Views und ViewGroups besteht.

Activities bilden demnach die vom Anwender wahrgenommenen Bausteine einer App. Sie können sich gegenseitig aufrufen; die Vorwärtsnavigation innerhalb einer Anwendung wird auf diese Weise realisiert. Da das System Activities auf einem Stapel ablegt, müssen Sie sich als Entwickler nicht darum kümmern, von wem Ihre Activity aufgerufen wird. Drückt der Benutzer die reale oder eine virtuelle ZURÜCK-Schaltfläche, wird automatisch die zuvor angezeigte Activity reaktiviert. Vielleicht fragen Sie sich, aus wie vielen Activities *Hallo Android* besteht?

Theoretisch könnten Sie die App in drei Activities unterteilen, die Sie unabhängig voneinander anlegen müssten:

1. Begrüßung anzeigen
2. Namen eingeben
3. personalisierten Gruß anzeigen

Das wäre sinnvoll, wenn die entsprechenden Aufgaben umfangreiche Benutzereingaben oder aufwendige Netzwerkkommunikation erforderten. Dies ist hier nicht der Fall. Da die gesamte Anwendung aus sehr wenigen Bedienelementen besteht, ist es in diesem Fall zielführender, alle Funktionen in einer Activity abzubilden. Bitte übernehmen Sie die Klasse `MainActivity` aus der im Folgenden dargestellten ersten Version.

In der Methode `onCreate()` wird mit `setContentView()` die Benutzeroberfläche geladen und angezeigt. Danach werden durch den Aufruf der Methode `findViewById()` zwei Referenzen auf Bedienelemente ermittelt und den Variablen `nachricht` und `weiterFertig` zugewiesen. `setText()` setzt die Beschriftung der Schaltfläche sowie des Textfeldes. Hierzu erfahren Sie gleich mehr.

Hinweis

Bitte achten Sie darauf, in Ihren Apps `findViewById()` erst nach `setContentView()` aufzurufen. Andernfalls drohen Abstürze.

```
package com.thomaskuenneth.halloandroid;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

  private TextView nachricht;
  private Button weiterFertig;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      nachricht = findViewById(R.id.nachricht);
      weiterFertig = findViewById(R.id.weiter_fertig);
```



```
      nachricht.setText(R.string.willkommen);
      weiterFertig.setText(R.string.weiter);
  }
}
```

Listing 2.4 Erste Version der Klasse »MainActivity«

Um die Anwendung zu starten, wählen Sie Run • Run app. Android Studio möchte von Ihnen nun wieder wissen, mit welchem Gerät Sie die Anwendung verwenden wollen. Markieren Sie im Dialog Select Deployment Target entweder den Emulator, den Sie in Kapitel 1, »Android – eine offene, mobile Plattform«, angelegt haben, oder ein über USB-Kabel verbundenes echtes Gerät, und schließen Sie den Dialog mit OK.

Wie in Abbildung 2.13 gezeigt, kann Select Deployment Target unterschiedliche Bereiche haben. Connected Devices zeigt alle gestarteten und verbundenen Geräte an. Available Virtual Devices bietet verfügbare, aber noch nicht gestartete Emulatoren an. Welche dieser beiden Bereiche angezeigt werden, hängt davon ab, welche virtuellen Geräte Sie angelegt und gestartet haben. Nach der Installation sollte das Emulatorfenster bzw. der Bildschirm des echten Geräts in etwa Abbildung 2.14 entsprechen. »In etwa«, weil es ein paar Faktoren gibt, die die Darstellung einer App beeinflussen:

- die Plattformversion des Emulators
- der API-Level in der Datei *build.gradle*

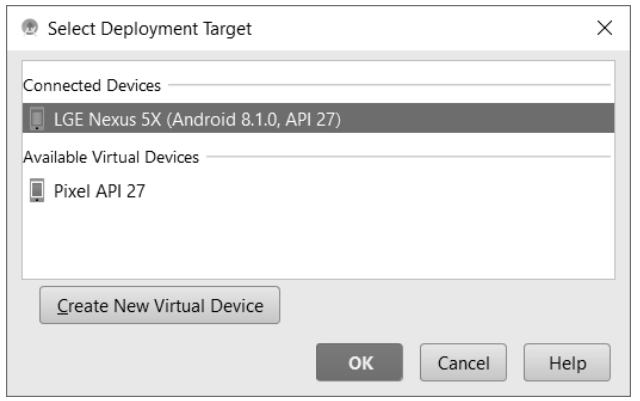


Abbildung 2.13 Der Dialog »Select Deployment Target«

Lassen Sie uns zunächst weiter auf Ihre erste eigene App konzentrieren. Das Textfeld nimmt zwar Eingaben entgegen, das Anklicken der Schaltfläche Weiter löst aber selbstverständlich noch keine Aktion aus. Diese Aktion werden wir im nächsten Ab-

schnitt implementieren. Zuvor möchte ich Sie aber mit einigen Schlüsselstellen des Quelltextes vertraut machen. Ganz wichtig: Jede Activity erbt von der Klasse `android.app.Activity` oder von spezialisierten Kindklassen. Beispielsweise kennt die Plattform die Klasse `ListActivity`[1], die das Erstellen von Auswahl- und Übersichtslisten vereinfacht.

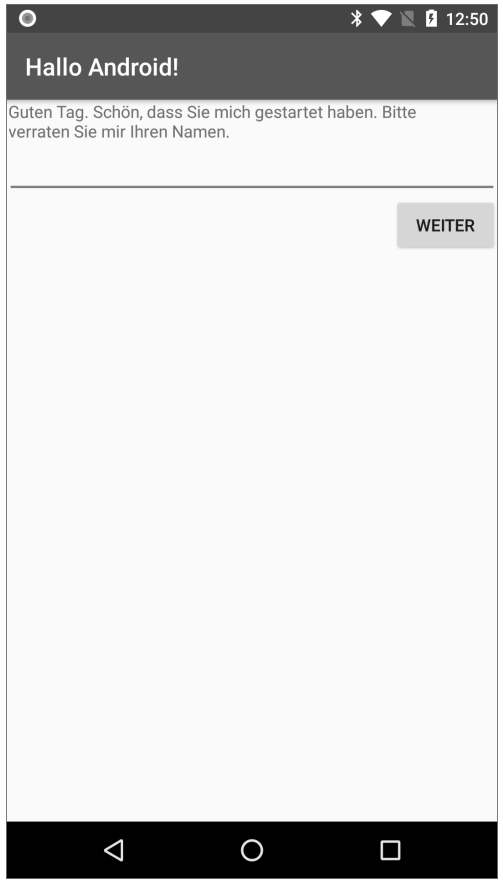


Abbildung 2.14 »Hallo Android« auf einem Nexus 5X

Haben Sie bemerkt, dass die gesamte Programmlogik in der Methode `onCreate()` liegt? Activities haben einen ausgeklügelten Lebenszyklus, den ich Ihnen in Kapitel 4, »Activities und Broadcast Receiver«, ausführlicher vorstelle. Seine einzelnen Stationen werden durch bestimmte Methoden der Klasse `Activity` realisiert, die Sie bei Bedarf überschreiben können. Beispielsweise informiert die Plattform eine Activity, kurz bevor sie beendet, unterbrochen oder zerstört wird. Die Methode `onCreate()` wird immer überschrieben. Sie ist der ideale Ort, um die Benutzeroberfläche aufzu-

1 *http://developer.android.com/reference/android/app/ListActivity.html*

bauen und Variablen zu initialisieren. Ganz wichtig ist, mit `super.onCreate()` die Implementierung der Elternklasse aufzurufen. Sonst wird zur Laufzeit die Ausnahme `SuperNotCalledException` ausgelöst.

Das Laden und Anzeigen der Bedienelemente reduziert sich auf eine Zeile Quelltext:

```
setContentView(R.layout.activity_main);
```

Sie sorgt dafür, dass alle Views und ViewGroups, die in der Datei *activity_main.xml* definiert wurden, zu einem Objektbaum entfaltet werden und dieser als Inhaltsbereich der Activity gesetzt wird. Warum ich den Begriff »entfalten« verwende, erkläre ich Ihnen in Kapitel 5, »Benutzeroberflächen«.

Möglicherweise fragen Sie sich, woher die Klasse `R` stammt. Sie wird von den *Build Tools* automatisch generiert und auf dem aktuellen Stand gehalten. Ihr Zweck ist es, Elemente aus Layout- und anderen XML-Dateien im Java-Quelltext verfügbar zu machen. `R.layout.activity_main` referenziert also die XML-Datei mit Namen *activity_main*.

Der Inhalt des Textfeldes `nachricht` und die Beschriftung der Schaltfläche `weiterFertig` wird auf sehr ähnliche Weise festgelegt: Zunächst ermitteln wir durch Aufruf der Methode `findViewById()` eine Referenz auf das gewünschte Objekt. `R.id.nachricht` und `R.id.weiter_fertig` verweisen hierbei auf Elemente, die wir ebenfalls in *activity_main.xml* definiert haben. Sehen Sie sich zur Verdeutlichung folgendes Dateifragment an:

```
<TextView
  android:id="@+id/nachricht"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
/>
…
<Button
  android:id="@+id/weiter_fertig"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_gravity="end"
/>
```

Listing 2.5 Auszug aus der Datei »activity_main.xml«

Durch den Ausdruck `android:id="@+id/xyz"` entsteht ein Bezeichner, auf den Sie mit `R.id.xyz` zugreifen können. Dies funktioniert nicht nur in Layoutdateien, sondern auch für die Definition von Texten, die in der Datei *strings.xml* abgelegt werden. Auch hierzu ein kurzer Auszug:

```
<!-- Beschriftungen für Schaltflächen -->
<string name="weiter">Weiter</string>
<string name="fertig">Fertig</string>
```

Listing 2.6 Auszug aus der Datei »strings.xml«

Die Anweisung `weiterFertig.setText(R.string.weiter);` legt den Text der einzigen Schaltfläche unserer App fest.

2.3.2 Benutzereingaben

Um *Hallo Android* zu komplettieren, müssen wir auf das Anklicken der Schaltfläche `weiterFertig` reagieren. Beim ersten Mal wird das Textfeld `eingabe` ausgelesen und als persönlicher Gruß in `nachricht` eingetragen. Anschließend wird das Textfeld ausgeblendet und die Beschriftung der Schaltfläche geändert. Wird diese ein zweites Mal angeklickt, beendet sich die App. Im Folgenden sehen Sie die entsprechend erweiterte Fassung der Klasse `MainActivity`:

```
package com.thomaskuenneth.halloandroid;

import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

  private TextView nachricht;
  private Button weiterFertig;
  private EditText eingabe;
  private boolean ersterKlick;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    nachricht = findViewById(R.id.nachricht);
    weiterFertig = findViewById(R.id.weiter_fertig);
```



```
    nachricht.setText(R.string.willkommen);
    weiterFertig.setText(R.string.weiter);
    eingabe = findViewById(R.id.eingabe);

    ersterKlick = true;
    nachricht.setText(R.string.willkommen);
    weiterFertig.setText(R.string.weiter);

    weiterFertig.setOnClickListener(v -> {
      if (ersterKlick) {
        nachricht.setText(getString(R.string.hallo,
            eingabe.getText()));
        eingabe.setVisibility(View.INVISIBLE);
        weiterFertig.setText(R.string.fertig);
        ersterKlick = false;
      } else {
        finish();
      }
    });
  }
}
```

Listing 2.7 Zweite Fassung der Klasse »MainActivity«

Um auf das Anklicken der Schaltfläche reagieren zu können, wird ein sogenannter *OnClickListener* registriert. Dieses Interface besteht aus der Methode `onClick()`. Die hier vorgestellte Implementierung unter Verwendung eines *Lambda*-Ausdrucks nutzt die `boolean`-Variable `ersterKlick`, um die durchzuführenden Aktionen zu bestimmen. `eingabe.setVisibility(View.INVISIBLE);` blendet das Eingabefeld aus. `getString(R.string.hallo, eingabe.getText())` liefert den in *strings.xml* definierten persönlichen Gruß und fügt an der Stelle `%1$s` den vom Benutzer eingetippten Namen ein. Um die App zu beenden, wird die Methode `finish()` der Klasse `Activity` aufgerufen.

2.3.3 Der letzte Schliff

In diesem Abschnitt möchte ich Ihnen zeigen, wie Sie *Hallo Android* den letzten Schliff geben. Zum Beispiel kann das System in leeren Eingabefeldern einen Hinweis anzeigen, was der Benutzer eingeben soll. Hierzu fügen Sie in der Datei *strings.xml* die folgende Zeile ein:

```
<string name="vorname_nachname">Vorname Nachname</string>
```

Anschließend erweitern Sie in *activity_main.xml* das Element `<EditText>` um das Attribut `android:hint="@string/vorname_nachname"`. Damit verschwindet übrigens auch die Warnung, die Sie etwas weiter vorne gesehen haben. Starten Sie die App, um sich das Ergebnis anzusehen. Abbildung 2.15 zeigt das entsprechend abgeänderte Programm.

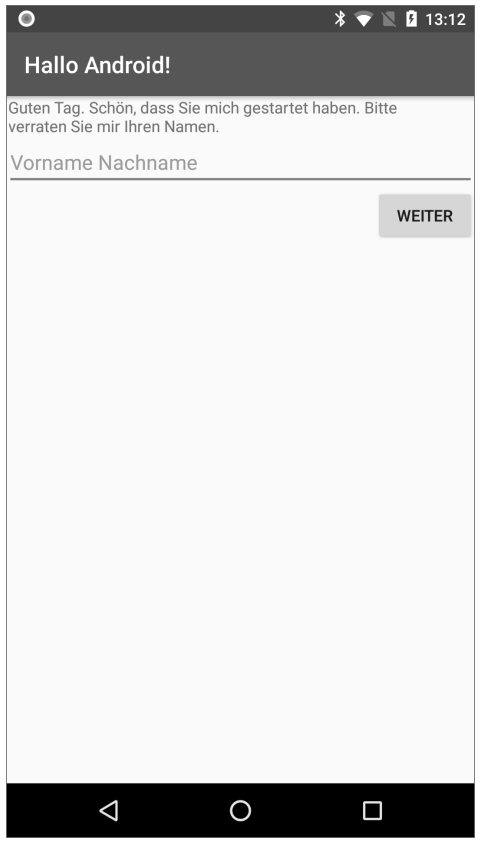


Abbildung 2.15 Leeres Eingabefeld mit Hinweis

Schon besser, wenn auch noch nicht perfekt. Drücken Sie während der Eingabe eines Namens nämlich auf ↵, wandert der Cursor in die nächste Zeile, und auch die Höhe des Eingabefeldes nimmt zu. Dieses Verhalten lässt sich zum Glück leicht korrigieren. Erweitern Sie hierzu `<EditText>` um die folgenden drei Attribute:

```
android:lines="1"
android:inputType=" textCapWords"
android:imeOptions="actionNext"
```

Damit begrenzen wir die Eingabe auf eine Zeile, und der erste Buchstabe eines Wortes wird automatisch in einen Großbuchstaben umgewandelt. Schließlich löst das Drücken von ↵ bzw. das Anklicken des korrespondierenden Symbols auf der virtuellen Gerätetastatur eine Aktion aus. Um auf diese reagieren zu können, müssen wir in der Activity ebenfalls eine Kleinigkeit hinzufügen. Unter die Zeile `eingabe = findViewById(R.id.eingabe);` gehören die folgenden Zeilen:

```
eingabe.setOnEditorActionListener((v, actionId, event) -> {
  if (weiterFertig.isEnabled()) {
    weiterFertig.performClick();
  }
  return true;
});
```

Der Aufruf der Methode `performClick()` simuliert das Antippen der Schaltfläche WEITER. Dadurch wird der Code ausgeführt, den wir in der Methode `onClick()` der Klasse `OnClickListener` implementiert haben. Alternativ hätten wir diesen Code auch in eine eigene Methode auslagern und diese an beiden Stellen aufrufen können. Aber Sie wissen nun, wie Sie das Antippen einer Komponente simulieren können. Übrigens prüft die Methode `isEnabled()`, ob die Schaltfläche aktiv oder inaktiv ist. Das werden wir gleich noch brauchen.

Schließlich wollen wir noch dafür sorgen, dass die Bedienelemente nicht mehr an den Rändern der Anzeige kleben. Eine kurze Anweisung schiebt zwischen ihnen und dem Rand einen kleinen leeren Bereich ein. Fügen Sie dem XML-Tag `<LinearLayout>` einfach das Attribut `android:padding="10dp"` hinzu. *Padding* wirkt nach innen. Das `LinearLayout` ist eine Komponente, die weitere Elemente enthält. Diese werden in horizontaler oder vertikaler Richtung angeordnet. Mit `android:padding` legen Sie fest, wie nahe die Schaltfläche, das Textfeld und die Eingabezeile der oberen, unteren, linken und rechten Begrenzung kommen können.

Im Gegensatz dazu wirkt *Margin* nach außen. Hiermit können Sie einen Bereich um die Begrenzung einer Komponente herum definieren. Auch hierzu ein Beispiel: Fügen Sie dem XML-Tag `<Button>` das Attribut `android:layout_marginTop="16dp"` hinzu, wird die Schaltfläche deutlich nach unten abgesetzt. Sie haben einen oberen Rand definiert, der gegen die untere Begrenzung der Eingabezeile wirkt. Werte, die auf `dp` enden, geben übrigens geräteunabhängige Pixelgrößen an. Sie beziehen die Auflösung der Anzeige eines Geräts mit ein.

Fällt Ihnen noch ein Defizit der gegenwärtigen Version auf? Solange der Benutzer keinen Namen eingetippt hat, sollte die Schaltfläche WEITER nicht anwählbar sein. Das lässt sich mithilfe eines sogenannten *TextWatchers* leicht realisieren. Dazu fügen Sie in der Methode `onCreate()` vor dem Ende des Methodenrumpfes, also vor ihrer schließenden geschweiften Klammer, das folgende Quelltextfragment ein:

```
eingabe.addTextChangedListener(new TextWatcher() {

   @Override
   public void onTextChanged(CharSequence s, int start,
          int before, int count) {
   }

   @Override
   public void beforeTextChanged(CharSequence s, int start,
          int count, int after) {
   }

   @Override
   public void afterTextChanged(Editable s) {
      weiterFertig.setEnabled(s.length() > 0);
   }
});
weiterFertig.setEnabled(false);
```

Listing 2.8 Auszug aus »HalloAndroidActivity.java«

Mit diesem Code scheint etwas nicht zu stimmen. Android Studio unterkringelt einige seiner Bestandteile mit roten Schlangenlinien, andere Elemente erscheinen in roter Schrift. Aber keine Bange, Sie müssen nur noch die fehlenden `import`-Anweisungen für die Klassen `TextWatcher` und `Editable` ergänzen. Das geht ganz einfach. Klicken Sie `Editable` an, und drücken Sie danach Alt+↵. Wiederholen Sie dies mit `TextWatcher`.

Jedes Mal, wenn ein Zeichen eingegeben oder gelöscht wird, ruft Android unsere Implementierung der Methode `afterTextChanged()` auf. Diese ist sehr einfach gehalten: Nur wenn der Name mindestens ein Zeichen lang ist, kann die Schaltfläche WEITER angeklickt werden. Als kleine Übung können Sie versuchen, die Prüfroutine so zu erweitern, dass Vor- und Nachname vorhanden sein müssen. Prüfen Sie der Einfachheit halber, ob der eingegebene Text ein Leerzeichen enthält, das nicht am Anfang und nicht am Ende steht.

Damit haben Sie Ihre erste eigene Anwendung fast fertiggestellt. Es gibt nur noch eine kleine Unvollkommenheit: Die Schaltfläche FERTIG befindet sich gegenüber der Schaltfläche WEITER etwas näher am oberen Bildschirmrand. Der Grund ist, dass die Grußfloskel meistens in eine Zeile passt, der Begrüßungstext aber zwei Zeilen benötigt. Beheben Sie dieses Malheur, indem Sie in der Layoutdatei innerhalb des `<TextView />`-Tags, zum Beispiel unterhalb von `android:id="@+id/nachricht"`, die Zeile `android:lines="2"` einfügen. Abbildung 2.16 zeigt die fertige App *Hallo Android*.

Abbildung 2.16 Die fertige App »Hallo Android«

2.4 Zusammenfassung

Sie haben in diesem Kapitel mithilfe des Projektassistenten ein neues Projekt angelegt und zu einer vollständigen App mit Benutzerinteraktion erweitert. Dabei haben Sie Layoutdateien kennengelernt und erste Erfahrungen mit dem quelltextunabhängigen Speichern von Texten gesammelt. Die folgenden Kapitel vertiefen dieses Wissen.

Kapitel 8

Sensoren, GPS und Bluetooth

Android-Geräte enthalten zahlreiche Sensoren und Schnittstellen, die sich mit geringem Aufwand in eigenen Apps nutzen lassen. Wie das funktioniert, zeige ich Ihnen in diesem Kapitel.

Moderne Mobiltelefone schalten ihre Anzeige ab, sobald man sie in Richtung des Kopfes bewegt. Die Darstellung auf dem Bildschirm passt sich der Ausrichtung des Geräts an. Spiele reagieren auf Bewegungsänderungen. Karten-Apps erkennen automatisch den gegenwärtigen Standort. Restaurant- oder Kneipenführer beschreiben nicht nur den kürzesten Weg zur angesagten Döner-Bude, sondern präsentieren die Meinungen anderer Kunden und bieten Alternativen an. Und mit der Funktechnologie Bluetooth lassen sich im Handumdrehen Geräte in Reichweite ansprechen und vernetzen. Dies und noch viel mehr ist möglich, weil die Android-Plattform eine beeindruckende Sensoren- und Schnittstellenphalanx beinhaltet, die von allen Apps genutzt werden kann.

8.1 Sensoren

Android stellt seine Sensoren über eine Instanz der Klasse `SensorManager` zur Verfügung. Wie Sie diese verwenden, zeige ich Ihnen anhand des Projekts *SensorDemo1*. Die gleichnamige App (sie ist in Abbildung 8.1 zu sehen) ermittelt alle zur Verfügung stehenden Sensoren und gibt unter anderem deren Namen, Hersteller und Version aus. Außerdem verbindet sich das Programm mit dem Helligkeitssensor des Geräts und zeigt die gemessenen Werte an. Sensoren lassen sich grob in drei Kategorien unterteilen:

- *Bewegungssensoren* messen Beschleunigungs- und Drehungskräfte entlang dreier Achsen. Zu dieser Kategorie gehören Accelerometer, Gyroskop und Gravitationsmesser.
- *Umweltsensoren* erfassen verschiedene Parameter der Umwelt, beispielsweise die Umgebungstemperatur, den Luftdruck, Feuchtigkeit und Helligkeit. Zu dieser Kategorie gehören Barometer, Photometer und Thermometer.
- *Positionssensoren* ermitteln die Position bzw. die Lage eines Geräts im Raum. Diese Kategorie beinhaltet Orientierungssensoren sowie Magnetometer.

Abbildung 8.1 Die App »SensorDemo1« auf einem Nexus 5X

Welche Messfühler einer App zur Verfügung stehen, hängt sowohl von der Plattformversion als auch von der Hardware ab, auf der die App ausgeführt wird. Android hat im Laufe der Zeit nämlich kontinuierlich neue Sensoren »gelernt«. Sensoren können durch Hard- oder Software realisiert werden. Je nach Typ verbrauchen sie viel oder wenig Strom. Einige liefern kontinuierlich Daten, andere nur, wenn sich seit der letzten Messung etwas geändert hat. Die Nutzung der Sensoren erfolgt primär über die Klasse `android.hardware.SensorManager`, die ich nun ausführlich vorstellen werde.

8.1.1 Die Klasse »SensorManager«

Die Methode `onCreate()` meiner Beispielklasse `SensorDemo1Activity` kümmert sich um das Laden und Anzeigen der Benutzeroberfläche. Alle sensorbezogenen Aktivitäten finden in `onResume()` und `onPause()` statt. Beim Fortsetzen der Activity wird mit `getSystemService(SensorManager.class)` die Referenz auf ein Objekt des Typs

`android.hardware.SensorManager` ermittelt. Diese Methode ist in allen der von `android.content.Context` abgeleiteten Klassen vorhanden, beispielsweise in `android.app.Activity` und `android.app.Service`. Anschließend können Sie mit `getSensorList()` herausfinden, welche Sensoren in Ihrer App zur Verfügung stehen. Die Methoden `getName()`, `getVendor()` und `getVersion()` liefern den Namen, den Hersteller und die Version des Sensors. Mit `isDynamicSensor()` können Sie ermitteln, ob ein Sensor dynamisch ist. Was es damit auf sich hat, erkläre ich Ihnen im folgenden Abschnitt.

Beim Aufruf von `getSensorList()` können Sie anstelle von `TYPE_ALL` die übrigen mit `TYPE_` beginnenden Konstanten der Klasse `Sensor` nutzen, um nach einer bestimmten Art von Sensor »Ausschau zu halten«. Beispielsweise begrenzt `TYPE_LIGHT` die Trefferliste auf Helligkeitssensoren. `TYPE_STEP_DETECTOR` liefert Schrittdetektoren; solche Sensoren melden sich, wenn der Nutzer einen Fuß mit genügend »Schwung« auf den Boden stellt, also einen Schritt macht. In Abschnitt 8.1.3, »Ein Schrittzähler«, erfahren Sie mehr darüber.

Wenn Sie die Art des gewünschten Sensors schon kennen, ist es meist einfacher, anstelle von `getSensorList()` die Methode `getDefaultSensor()` aufzurufen. Allerdings weist die Android-Dokumentation darauf hin, dass diese Methode unter Umständen einen Sensor liefert, der gefilterte oder gemittelte Werte produziert. Möchten Sie dies – zum Beispiel aus Genauigkeitsgründen – nicht, dann verwenden Sie `getSensorList()`. Neben ihren Namen und Herstellern liefern Sensoren eine ganze Menge an Informationen, beispielsweise zu ihrem Stromverbrauch (`getPower()`), ihrem Wertebereich (`getMaximumRange()`) und ihrer Genauigkeit (`getResolution()`).

```
package com.thomaskuenneth.sensordemo1;

import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.widget.TextView;
import java.util.HashMap;
import java.util.List;

public class SensorDemo1Activity extends Activity {

  private static final String TAG =
      SensorDemo1Activity.class.getSimpleName();
```

```
private TextView textview;
private SensorManager manager;
private Sensor sensor;
private SensorEventListener listener;
private SensorManager.DynamicSensorCallback callback;
private HashMap<String, Boolean> hm;

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  hm = new HashMap<>();
  setContentView(R.layout.main);
  textview = findViewById(R.id.textview);
}

@Override
protected void onResume() {
  super.onResume();
  textview.setText("");
  // Liste der vorhandenen Sensoren ausgeben
  manager = getSystemService(SensorManager.class);
  if (manager == null) {
    finish();
  }
  List<Sensor> sensors = manager.getSensorList(Sensor.TYPE_ALL);
  for (Sensor s : sensors) {
    textview.append(getString(R.string.template,
        s.getName(),
        s.getVendor(),
        s.getVersion(),
        Boolean.toString(s.isDynamicSensor())));
  }
  // Helligkeitssensor ermitteln
  sensor = manager.getDefaultSensor(Sensor.TYPE_LIGHT);
  if (sensor != null) {
    listener = new SensorEventListener() {

      @Override
      public void onAccuracyChanged(Sensor sensor,
                    int accuracy) {
        Log.d(TAG, "onAccuracyChanged(): " + accuracy);
      }
```



```
      @Override
      public void onSensorChanged(SensorEvent event) {
        if (event.values.length > 0) {
          float light = event.values[0];
          String text = Float.toString(light);
          if ((SensorManager.LIGHT_SUNLIGHT <= light)
              && (light <=
              SensorManager.LIGHT_SUNLIGHT_MAX)) {
            text = getString(R.string.sunny);
          }
          // jeden Wert nur einmal ausgeben
          if (!hm.containsKey(text)) {
            hm.put(text, Boolean.TRUE);
            text += "\n";
            textview.append(text);
          }
        }
      }
    };
    // Listener registrieren
    manager.registerListener(listener, sensor,
        SensorManager.SENSOR_DELAY_NORMAL);
  } else {
    textview.append(getString(R.string.no_seonsor));
  }

  // Callback für dynamische Sensoren
  callback = null;
  if (manager.isDynamicSensorDiscoverySupported()) {
    callback = new SensorManager.DynamicSensorCallback() {
      @Override
      public void onDynamicSensorConnected(Sensor sensor) {
        textview.append(getString(R.string.connected,
            sensor.getName()));
      }

      @Override
      public void onDynamicSensorDisconnected(Sensor sensor) {
        textview.append(getString(R.string.disconnected,
            sensor.getName()));
      }
    };
```

```
      manager.registerDynamicSensorCallback(callback,
          new Handler());
    }
  }

  @Override
  protected void onPause() {
    super.onPause();
    if (sensor != null) {
      manager.unregisterListener(listener);
    }
    if (callback != null) {
      manager.unregisterDynamicSensorCallback(callback);
    }
  }
}
```

Listing 8.1 Die Klasse »SensorDemo1Activity«

Seit *API-Level 21* kennt die Methode `getDefaultSensor()` einen optionalen zweiten Parameter, der steuert, ob das System sogenannte *Wake-up-* oder *Non-Wake-up*-Sensoren liefert. Der Unterschied besteht darin, ob Sensoren für das Melden von Daten den SoC (*System on a Chip*) aus dem Ruhezustand aufwecken und das Wechseln in diesen Modus verhindern (wake-up) oder nicht (non-wake-up). Sofern Sensordaten nur während der Ausführung einer Activity erhoben und angezeigt werden, ist die Unterscheidung irrelevant. Für eine möglichst unterbrechungsfreie Aufzeichnung im Hintergrund sind Wake-up-Sensoren die bessere Wahl. Andernfalls muss die App selbstständig dafür sorgen, dass der SoC nicht in den Ruhezustand wechselt. Weitere Informationen finden Sie unter *https://source.android.com/devices/sensors/suspend-mode.html*.

SensorEventListener

Mit den Methoden `registerListener()` und `unregisterListener()` der Klasse `SensorManager` können Sie sich über Sensorereignisse informieren lassen sowie entsprechende Benachrichtigungen wieder deaktivieren. `registerListener()` erwartet ein Objekt des Typs `android.hardware.SensorEventListener`, den Sensor sowie eine Angabe zur Häufigkeit, mit der Wertänderungen übermittelt werden sollen. Sie können einen vordefinierten Wert, zum Beispiel `SensorManager.SENSOR_DELAY_NORMAL`, oder eine Zeitspanne in Mikrosekunden übergeben. Android garantiert die Einhaltung dieses Wertes allerdings nicht. Sensorereignisse können also häufiger oder seltener zugestellt werden.

Das Activity-Methodenpaar `onResume()` und `onPause()`bietet sich an, um `SensorEventListener` zu registrieren bzw. zu entfernen. Prüfen Sie genau, ob das Sammeln von Sensordaten auch dann erforderlich ist, wenn Ihre Activity nicht ausgeführt wird. Je nach Sensor kann das Messen nämlich in erheblichem Maße Strom verbrauchen.

`SensorEventListener`-Objekte implementieren die Methoden `onAccuracyChanged()` und `onSensorChanged()`. Erstere wird aufgerufen, wenn sich die Genauigkeit eines Sensors geändert hat. Wie wichtig diese Information ist, hängt von der Art des verwendeten Messfühlers ab. Sollte es beispielsweise Probleme beim Ermitteln der Herzfrequenz geben, weil der Sensor kalibriert werden muss (`SENSOR_STATUS_UNRELIABLE`) oder weil er keinen Körperkontakt hat (`SENSOR_STATUS_NO_CONTACT`), dann sollte Ihre App auf jeden Fall einen entsprechenden Hinweis anzeigen. Ist hingegen die Genauigkeit des Barometers nicht mehr hoch (`SENSOR_STATUS_ACCURACY_HIGH`), sondern nur noch durchschnittlich (`SENSOR_STATUS_ACCURACY_MEDIUM`), ist vielleicht keine diesbezügliche Aktion erforderlich.

Die Methode `onSensorChanged()` wird aufgerufen, wenn neue Sensordaten vorliegen. Die App *SensorDemo1* nutzt den Helligkeitssensor eines Geräts und gibt je nach Helligkeit den gemessenen Wert oder den Text »sonnig« aus. Die Klasse `SensorManager` enthält zahlreiche Konstanten, die sich auf die vorhandenen Ereignistypen beziehen. Auf diese Weise können Sie, wie im Beispiel zu sehen ist, das Ergebnis der Helligkeitsmessung auswerten, ohne selbst in entsprechenden Tabellen nachschlagen zu müssen.

> **Tipp**
>
> Liefert die mit Android Nougat eingeführte `Sensor`-Methode `isAdditionalInfoSupported()` den Wert `true`, kann ein Sensor über einen neuen Mechanismus weitere, zusätzliche Informationen preisgeben. Sie sind in der Klasse `SensorAdditionalInfo` enthalten. Um solche Objekte zu empfangen, registrieren Sie mit `registerListener()` anstelle von `SensorEventListener` ein Objekt des Typs `SensorEventCallback` und überschreiben zusätzlich die Methode `onSensorAdditionalInfo()`.

Welche Werte in dem `SensorEvent`-Objekt übermittelt werden und wie Sie diese interpretieren, hängt vom verwendeten Sensor ab. Beispielsweise liefert der Umgebungstemperatursensor (`TYPE_AMBIENT_TEMPERATURE`) in `values[0]` die Raumtemperatur in Grad Celsius. Luftdruckmesser (`Sensor.TYPE_PRESSURE`) tragen dort hingegen den atmosphärischen Druck in Millibar ein.

Die von einem Android-Gerät oder dem Emulator zur Verfügung gestellten Sensoren können Sie erst zur Laufzeit Ihrer App ermitteln. Selbstverständlich sollten Sie nicht einfach Ihre Activity beenden, wenn ein benötigter Sensor nicht zur Verfügung steht, sondern einen entsprechenden Hinweis ausgeben. Mithilfe des Elements `<uses-feature>` der Manifestdatei können Sie die Sichtbarkeit in *Google Play* auf geeignete Geräte einschränken. Hierzu ein Beispiel:

```
<uses-feature android:name="android.hardware.sensor.barometer"
   android:required="true" />
```

Apps, deren Manifest ein solches Element enthält, werden in Google Play nur auf Geräten angezeigt, in die ein Barometer eingebaut ist. Beachten Sie hierbei aber, dass diese Filterung eine Installation nicht verhindert, falls die App auf anderem Wege auf das Gerät gelangt ist. Deshalb ist es wichtig, vor der Nutzung eines Sensors seine Verfügbarkeit, wie weiter oben gezeigt, zu prüfen.

8.1.2 Dynamische Sensoren und Trigger

Mit Android 7 hat Google sogenannte *dynamische Sensoren* eingeführt. Bisher war es so, dass ein Sensor entweder immer »da ist« (weil er in ein Smartphone oder Tablet eingebaut wurde) oder eben nicht. Was aber wäre, wenn man ein Gerät durch Module erweitern und je nach Bedarf Sensoren andocken oder abklemmen könnte? Google hatte mit dem Projekt *Ara* die Vision eines voll modularen Smartphones. Unglücklicherweise scheint es eingestellt worden zu sein, aber es ist möglich, dass andere Hersteller die Idee wieder aufgreifen.

Apps können über die `SensorManager`-Methode `isDynamicSensorDiscoverySupported()` abfragen, ob das System das Erkennen von dynamischen Sensoren unterstützt. In diesem Fall lässt sich mit `registerDynamicSensorCallback()` ein Objekt des Typs `DynamicSensorCallback` registrieren. Seine Methoden `onDynamicSensorConnected()` und `onDynamicSensorDisconnected()` werden nach dem Verbinden bzw. Trennen eines dynamischen Sensors aufgerufen. Dies ist in Listing 8.1 zu sehen. Analog zu `SensorManager.getSensorList()` können Sie übrigens mit `getDynamicSensorList()` die Liste aller aktuell bekannten dynamischen Sensoren eines Typs abfragen.

Trigger-Sensoren

Viele Daten (zum Beispiel Temperatur, Luftdruck und Helligkeit) können bei Bedarf kontinuierlich erfasst werden, denn sie liegen immer vor. Deshalb ist es bewährte Praxis, Listener nur bei Bedarf zu registrieren und nach Gebrauch wieder zu entfernen. Je nach Sensor ist der Akku des Geräts sonst möglicherweise schnell leer. (Eine Ausnahme von dieser Regel stelle ich Ihnen übrigens im nächsten Abschnitt vor.) Es gibt aber auch Ereignisse, die unvorhersehbar irgendwann eintreten; dann ist eine kontinuierliche Messung sinnlos. Für solche Fälle kennt Android Trigger-Sensoren. Der *Significant-Motion*-Sensor ist ein Trigger. Er meldet sich, wenn das System eine Bewegung erkennt, die wahrscheinlich zu einer Positionsänderung führt. Dies ist beim Laufen, Fahrrad- oder Autofahren der Fall. Trigger-Sensoren liefern beim Aufruf von `getReportingMode()` den Wert `REPORTING_MODE_ONE_SHOT`. Um einen solchen Sensor zu aktivieren, registrieren Sie nicht mit `SensorManager.register()` einen `SensorEventListener`, sondern rufen `requestTriggerSensor()` auf und übergeben der Metho-

de ein `TriggerEventListener`-Objekt. Dessen Methode `onTrigger()` wird vom System aufgerufen, wenn der Trigger aktiviert wurde. Danach wird der Trigger automatisch deaktiviert. Um erneut informiert zu werden, müssen Sie deshalb wieder `requestTriggerSensor()` aufrufen. Mein Beispiel *Bewegungsmelder* fasst dies in einer kompakten App zusammen. Und so sieht die Hauptklasse `BewegungsmelderActivity` aus:

```
package com.thomaskuenneth.bewegungsmelder;

import android.app.Activity;
import android.hardware.Sensor;
import android.hardware.SensorManager;
import android.hardware.TriggerEvent;
import android.hardware.TriggerEventListener;
import android.os.Bundle;
import android.text.format.DateFormat;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Date;

public class BewegungsmelderActivity extends Activity {

  private static final String KEY1
      = "shouldCallWaitForTriggerInOnResume";
  private static final String KEY2 = "tv";

  private TextView tv;
  private Button bt;

  private SensorManager m;
  private TriggerEventListener l;
  private Sensor s;

  private boolean shouldCallWaitForTriggerInOnResume;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    tv = findViewById(R.id.tv);
    bt = findViewById(R.id.bt);
    bt.setOnClickListener((e) -> {
      shouldCallWaitForTriggerInOnResume = true;
```

```
    waitForTrigger();
  });

  m = getSystemService(SensorManager.class);
  if (m == null) {
    finish();
  }
  s = m.getDefaultSensor(Sensor.TYPE_SIGNIFICANT_MOTION);
  l = new TriggerEventListener() {
    @Override
    public void onTrigger(TriggerEvent event) {
      shouldCallWaitForTriggerInOnResume = false;
      bt.setVisibility(View.VISIBLE);
      tv.setText(
          DateFormat.getTimeFormat(
              BewegungsmelderActivity.this).
              format(new Date()));
    }
  };
  if (s == null) {
    shouldCallWaitForTriggerInOnResume = false;
    bt.setVisibility(View.GONE);
    tv.setText(R.string.no_sensors);
  } else {
    shouldCallWaitForTriggerInOnResume = true;
    if (savedInstanceState != null) {
      shouldCallWaitForTriggerInOnResume =
          savedInstanceState.getBoolean(KEY1);
      tv.setText(savedInstanceState.getString(KEY2));
    }
  }
}

@Override
protected void onSaveInstanceState(Bundle outState) {
  super.onSaveInstanceState(outState);
  outState.putBoolean(KEY1, shouldCallWaitForTriggerInOnResume);
  outState.putString(KEY2, tv.getText().toString());
}

@Override
protected void onResume() {
```



```
    super.onResume();
    if (s != null) {
      if (shouldCallWaitForTriggerInOnResume) {
        waitForTrigger();
      }
    }
  }

  @Override
  protected void onPause() {
    super.onPause();
    if (s != null) {
      m.cancelTriggerSensor(l, s);
    }
  }

  private void waitForTrigger() {
    bt.setVisibility(View.GONE);
    tv.setText(R.string.wait);
    m.requestTriggerSensor(l, s);
  }
}
```

Listing 8.2 Die Klasse »BewegungsmelderActivity«

Der Trigger wird beim Fortsetzen der App mit `requestTriggerSensor()` aktiviert (`onResume()`) und beim Pausieren (`onPause()`) mit `cancelTriggerSensor()` deaktiviert. Möchten Sie, dass Ihre App auch dann informiert wird, wenn keine Activity abgearbeitet wird, müssen Sie die beiden Methodenaufrufe in einen Service auslagern. Geht das Gerät aber in den Ruhezustand, während `BewegungsmelderActivity` aktiv ist, wird die Aktivität nach dem Aufwachen des Geräts aktualisiert. Der *Significant-Motion*-Sensor arbeitet nämlich weiter, während das Gerät schläft.

Hinweis

Vielleicht ist Ihnen beim Stöbern in der Dokumentation aufgefallen, dass die Klasse `TriggerEvent` einen Zeitstempel enthält, der den Zeitpunkt des Auftretens in Nanosekunden angibt. Dieser Wert ist nicht dafür gedacht, Uhrzeiten oder Datumsangaben abzuleiten. Er sollte nur verwendet werden, um Abstände zwischen Aufrufen eines Sensors zu ermitteln.

Um beim Drehen des Geräts den aktuellen Zustand speichern und wiederherstellen zu können, habe ich die Methode `onSaveInstanceState()` überschrieben. Sie schreibt

zwei Werte, die `boolean`-Variable `shouldCallWaitForTriggerInOnResume` sowie den Inhalt des Textfeldes `tv`. Beide werden gegebenenfalls in `onCreate()` wieder gesetzt.

8.1.3 Ein Schrittzähler

In diesem Abschnitt sehen wir uns einen weiteren Sensor an, den Schrittzähler, der die Anzahl der Schritte seit dem letzten Start des Geräts meldet, allerdings nur, solange er aktiviert ist. Google empfiehlt in der Dokumentation deshalb, **nicht** die Methode `unregisterListener()` aufzurufen, wenn Langzeitmessungen erfolgen sollen. Diese sind unproblematisch, weil der Sensor in Hardware implementiert ist und wenig Strom verbraucht. Befindet sich das Gerät im Ruhemodus, werden bei aktivierten Sensor weiterhin Schritte gezählt und nach dem Wiederaufwachen gemeldet. Alles in allem eine wirklich praktische Angelegenheit, oder?

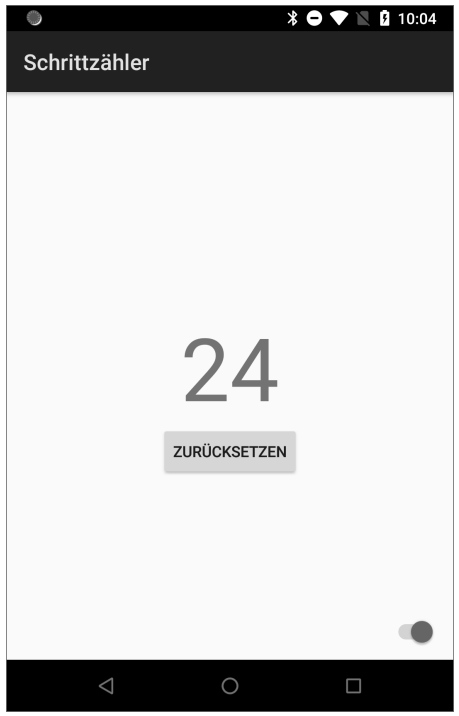


Abbildung 8.2 Die App »Schrittzähler«

Vielleicht fragen Sie sich, was passiert, wenn Sie die Anzahl der Schritte ganz bewusst zurücksetzen möchten. Da der Zähler die Schritte seit dem letzten Systemstart zählt, müssten Sie das Smartphone oder Tablet neu starten. Das klingt nicht sehr elegant. In meiner Beispiel-App *Schrittzähler* (sie ist in Abbildung 8.2 zu sehen) zeige ich Ihnen, wie Sie dieses Problem auf unkomplizierte Weise lösen. Bitte sehen Sie sich nun Listing 8.3 an. In der Methode `onCreate()` wird die Benutzeroberfläche geladen und an-

gezeigt. `getSystemService()` und `getDefaultSensor()` liefern wie gehabt Referenzen auf Objekte des Typs `SensorManager` bzw. `Sensor`. In `onResume()` rufe ich die private Methode `updateUI()` auf, die sich um das Aktualisieren der Bedienelemente kümmert. Je nach Zustand des Schalters `onOff` wird entweder `registerListener()` oder `unregisterListener()` aufgerufen. Das sonst übliche Deaktivieren des Sensors in `onPause()` entfällt. Sie erinnern sich: Google rät zu diesem Vorgehen, um Langzeitmessungen vornehmen zu können.

> **Tipp**
>
> Sie sollten in Ihrer App auf jeden Fall eine Möglichkeit vorsehen, den Sensor zu deaktivieren. In meinem Beispiel geschieht dies durch Anklicken des Schalters in der unteren rechten Bildschirmecke.


```
package com.thomaskuenneth.schrittzaehler;

import android.app.Activity;
import android.content.Context;
import android.content.SharedPreferences;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.Switch;
import android.widget.TextView;

import java.util.Locale;

public class SchrittzaehlerActivity extends Activity
    implements SensorEventListener {

  private static final String PREFS =
      SchrittzaehlerActivity.class.getName();

  private static final String PREFS_KEY = "last";

  private ProgressBar pb;
  private TextView steps;
  private Button reset;
  private Switch onOff;
```

```
private SensorManager m;
private Sensor s;
private int last;

@Override
protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  pb = findViewById(R.id.pb);
  steps = findViewById(R.id.steps);
  reset = findViewById(R.id.reset);
  reset.setOnClickListener((event) -> {
    updateSharedPrefs(this, last);
    updateUI();
  });
  m = getSystemService(SensorManager.class);
  if (m == null) {
    finish();
  }
  s = m.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
  onOff = findViewById(R.id.on_off);
  onOff.setOnCheckedChangeListener((buttonView, isChecked)
      -> updateUI());
  onOff.setChecked(s != null);
  updateUI();
}

@Override
protected void onResume() {
  super.onResume();
  updateUI();
}

@Override
public void onSensorChanged(SensorEvent sensorEvent) {
  float[] values = sensorEvent.values;
  int _steps = (int) values[0];
  last = _steps;
  SharedPreferences prefs = getSharedPreferences(
      SchrittzaehlerActivity.PREFS,
      Context.MODE_PRIVATE);
  _steps -= prefs.getInt(PREFS_KEY, 0);
  this.steps.setText(String.format(Locale.US,
      "%d", _steps));
```



```
  if (pb.getVisibility() == View.VISIBLE) {
    pb.setVisibility(View.GONE);
    this.steps.setVisibility(View.VISIBLE);
    reset.setVisibility(View.VISIBLE);
  }
}

@Override
public void onAccuracyChanged(Sensor sensor, int i) {
}

public static void updateSharedPrefs(Context context,
                                      int last) {
  SharedPreferences prefs =
      context.getSharedPreferences(
          SchrittzaehlerActivity.PREFS,
          Context.MODE_PRIVATE);
  SharedPreferences.Editor edit = prefs.edit();
  edit.putInt(SchrittzaehlerActivity.PREFS_KEY, last);
  edit.apply();
}

private void updateUI() {
  reset.setVisibility(View.GONE);
  onOff.setEnabled(s != null);
  if (s != null) {
    steps.setVisibility(View.GONE);
    if (onOff.isChecked()) {
      m.registerListener(this, s,
          SensorManager.SENSOR_DELAY_UI);
      pb.setVisibility(View.VISIBLE);
    } else {
      m.unregisterListener(this);
      pb.setVisibility(View.GONE);
    }
  } else {
    steps.setVisibility(View.VISIBLE);
    steps.setText(R.string.no_sensor);
    pb.setVisibility(View.GONE);
  }
}
```

Listing 8.3 Die Klasse »SchrittzaehlerActivity«

In der Methode `onSensorChanged()` wird aus dem Feld `sensorEvent.values` die Anzahl der Schritte seit dem letzten Systemstart ausgelesen und der Variablen `_steps` zugewiesen. Anschließend sieht meine Implementierung in den anwendungsspezifischen Voreinstellungen nach, ob dort schon ein Schlüssel mit dem Namen »last« (in der `String`-Konstante `PREFS_KEY` hinterlegt) gespeichert wurde. Falls nicht, liefert `getInt()` den als zweiten Parameter übergebenen Ersatzwert zurück. In meinem Fall ist dies 0. Der so ermittelte Wert wird von `_steps` abgezogen.

Wenn der Anwender die Schaltfläche Zurücksetzen anklickt, wird in den anwendungsspezifischen Voreinstellungen der Schlüssel `PREFS_KEY` gespeichert. Damit lässt sich ohne großen Aufwand das Auf-null-Stellen des Schrittzählers nachbauen. Eine Kleinigkeit gibt es aber zu beachten: Wenn das Gerät neu gestartet wurde, muss der gespeicherte Wert selbst auf 0 gesetzt werden, damit die App-Anzeige nicht verfälscht wird. Deshalb registriere ich in der Manifestdatei einen Receiver, der auf `android.intent.action.BOOT_COMPLETED` reagiert. Die Implementierung ist einfach. Sie ist in Listing 8.4 zu sehen:

```
package com.thomaskuenneth.schrittzaehler;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class BootCompletedReceiver extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
    if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
      SchrittzaehlerActivity.updateSharedPrefs(context, 0);
    }
  }
}
```

Listing 8.4 Die Klasse »BootCompletedReceiver«

Durch Aufruf der Methode `updateSharedPrefs()` der Klasse `SchrittzaehlerActivity` wird der beim letzten Anklicken der Schaltfläche Zurücksetzen gespeicherte Wert mit 0 überschrieben, und damit stimmt die Berechnung der Schritte wieder. Um Voreinstellungen zu schreiben, ermitteln Sie als Erstes mit `getSharedPreferences()` eine `SharedPreferences`-Instanz. Die Methode `edit()` liefert ein Objekt des Typs `SharedPreferences.Editor`. Mit seinen `put...()`-Methoden schreiben Sie zum Beispiel `int`- oder `String`-Werte. `apply()` persistiert die Änderungen.

Auf einen Blick

Inhalt

TEIL III Telefonfunktionen nutzen

TEIL IV Dateien und Datenbanken

TEIL V Organizer und Multimedia