5 Oberflächengestaltung

Die Qualität der Programmoberfläche ist ein wichtiges Kriterium für die Akzeptanz durch den Anwender. Schon durch den geschickten Einsatz der bereitgestellten Oberflächenelemente und deren optisch ansprechende Positionierung auf dem Bildschirm kann man viel erreichen.

Wir wollen uns daher im ersten Teil unseres Projekts mit den Grundlagen der Oberflächengestaltung in Android befassen. Dieses Kapitel führt in die Erstellung von Bildschirmseiten und Menüs ein. Wir werden alle notwendigen Komponenten und deren Zusammenspiel am Beispiel der Amando-Anwendung vorstellen. Auf Basis dieser Grundlagen können Sie später komplexere und optisch anspruchsvollere Oberflächen erzeugen.

5.1 Ziel

Erstes Ziel: die Startseite

Ziel dieses Kapitels ist es, den Startbildschirm von Amando zu implementieren. Auf diesem werden, neben einem Hinweistext, Schaltflächen für die Operationen *Position senden*, *Geokontakte verwalten*, *Karte anzeigen* und *Simulation starten* dargestellt. Über die Menütaste des Geräts können die Funktionen *Einstellungen*, *Hilfe* und *Beenden* aufgerufen werden. Die Schaltflächen und das Menü werden zunächst noch ohne Funktion sein. Die Texte der Startseite werden in einer Ressourcendatei verwaltet, was eine spätere Umstellung auf Mehrsprachigkeit erleichtert. Sie werden lernen, wie Ressourcen verwaltet werden und wie man sie referenziert.

5.2 Schnelleinstieg: Activities, Layouts und Views

In diesem Abschnitt erklären wir, aus welchen Grundbausteinen sich eine Oberfläche zusammensetzt. Anschließend wird beschrieben, wie man diese Bausteine zu einer lauffähigen Anwendung verknüpft.

5.2.1 Grundbausteine der Oberflächengestaltung

Wir wollen nach dem Start der Amando-Anwendung eine Bildschirmseite mit mehreren Schaltflächen, etwas Text und einem Bild angezeigt bekommen. Diese einzelnen Bausteine nennt man *Views*. Sie werden mit Hilfe eines Layouts angeordnet. Die Funktionalität, also unter anderem das Reagieren auf Oberflächenereignisse innerhalb des Layouts und der Zugriff auf die darin enthaltenen Views, erfolgt durch eine Android-Komponente namens *Activity*. Eine Activity sollte immer zur Darstellung genau einer Bildschirmseite implementiert werden. Sie besitzt zu einem Zeitpunkt nur ein Layout, das sie zur Anzeige bringt. Sie ist darüber hinaus in der Lage, auf Datenquellen und die Hardware des Android-Geräts zuzugreifen. Sie besitzt also die Kontrolle darüber, welche Daten angezeigt werden und welche Eingaben auf der Oberfläche im Programm umgesetzt werden.

Activity := Kontrolle

Jedes an der Oberfläche sichtbare Element ist von der Klasse `android.view.View` abgeleitet. Die klassischen Oberflächenelemente wie Eingabefelder, Schaltflächen und Auswahllisten werden wir ab jetzt als *Oberflächen-Komponente* oder *View* bezeichnen.

View = Darstellung

Views werden mit Hilfe eines Layouts auf dem Bildschirm angeordnet. Es gibt verschiedene Layouts, die eine jeweils unterschiedliche Anordnungsvorschrift darstellen. Es gibt ein Layout für eine lineare Anordnung der View-Elemente, ein Layout für eine tabellarische Anordnung, ein Layout für eine Anordnung relativ zueinander usw.

Layout = Anordnung

Layouts sind von der Klasse `android.view.ViewGroup` abgeleitet. Mit dieser Klasse hat man als Programmierer in der Regel selten bis gar nicht zu tun, aber es ist wichtig, sich zu vergegenwärtigen, dass Layouts (= `ViewGroup`) Gruppen von Views sind, die auch ineinandergeschachtelt werden können. Views selbst sind schon recht komplexe Objekte, Layouts demzufolge sehr komplexe Objekte, und geschachtelte Layouts können so umfangreich werden, dass sie die Performanz und den Speicherverbrauch einer Anwendung spürbar beeinflussen und sogar verhindern, dass eine Activity lauffähig ist. Wir werden in Abschnitt 5.4.1 darauf zurückkommen.

Oberfläche := Baum von Views

Die Oberfläche einer Anwendung, die gerade angezeigt wird, nennen wir Bildschirmseite. Sie setzt sich zusammen aus dem Layout (mit den darin enthaltenen Views oder weiteren Layouts) und einer Activity-Klasse, die auf die Oberflächenereignisse reagiert und Daten in View-Elementen zur Anzeige bringen kann. Die Bildschirmseite ist also der Teil der gesamten Oberfläche, mit der ein Anwender in Kontakt mit dem Programm tritt. Über sie übt er die Kontrolle über die Anwendung aus.

Bildschirmseite := Activity + Layout

5.2.2 Oberflächen implementieren

Beispiel: Startseite

Wir werden nun die oben genannten Begriffe auf unsere Aufgabenstellung, die Implementierung von Amando, anwenden. Am Beispiel der Startseite des Programms werden wir uns sowohl theoretisch wie praktisch mit dem Erstellen von Activities befassen. Abbildung 5-1 zeigt die Bildschirmseite, die angezeigt wird, wenn man Amando startet.

Abb. 5-1
Bildschirmaufbau

Je nachdem, welche Android-Version im Emulator oder auf dem Android-Gerät läuft, sieht die Oberfläche unterschiedlich aus. Die Abbildungen zeigen ein Android-4.1-Gerät. Man sieht auf der Startseite eine Textzeile, vier Schaltflächen und ein Bild. Dies sind unsere Views, die mit Hilfe eines Layouts innerhalb der Activity angeordnet sind.

Oben rechts im Bildschirm sind drei graue Punkte zu sehen. Drückt man darauf, öffnet sich das Optionsmenü. Android-2.x-Geräte haben eine eigene Taste für das Optionsmenü. Wenn es geöffnet wurde, werden die Menüoptionen *Einstellungen*, *Hilfe* und *Beenden* angezeigt. Dieses Menü ist nicht Teil des Layouts, sondern wird von der Activity beigesteuert.

Zunächst wird wieder, wie im Einstiegsbeispiel in Kapitel 1, ein Android-Projekt angelegt. Tabelle 5-1 zeigt die notwendigen Einstellungen, die wir für das Amando-Projekt brauchen.

Tab. 5-1 *Projekteinstellungen für das Amando-Projekt*

Application Name:	Amando
Project Name:	amando5
Package Name:	de.visionera.androidbuch.amando5
Minimum Required SDK:	API 14: Android 4.0.3
Target SDK:	API 21: Android 5.0
Compile With:	API 21: Android 5.0
Theme:	Material Light
Create Activity:	Blank Activity
Activity Name:	Startseite
Layout Name:	startseite

Android Studio generiert uns das gewünschte Projekt. Um nun dem Startbildschirm das gewünschte Aussehen zu geben, füllen wir die Datei `startseite.xml` mit dem Inhalt aus Listing 5-1.

Tipp

Das gesamte Projekt finden Sie auf der Webseite www.androidbuch.de im Bereich *Download* zum Herunterladen.

Listing 5-1 *Bildschirmlayout startseite.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent" >

  <TextView
    style="@style/TextStyleUeberschrift"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="@color/schwarz"
    android:text="@string/txt_startseiteanzeigen_intro" />

  <Button
    android:id="@+id/sf_position_senden"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/sf_position_senden"
    android:onClick="onClickPositionSenden" />
```

```
  <Button
    android:id="@+id/sf_starte_geokontakte"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/sf_geokontakte_verwalten"
    android:onClick="onClickGeokontakteVerwalten" />

  <Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/sf_karte_anzeigen"
    android:onClick="onClickKarteAnzeigen" />

  <Button
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/sf_simulation"
    android:onClick="onClickSimulationStarten" />

  <ImageView
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_gravity="center_horizontal"
    android:src="@drawable/amando_logo" />

</LinearLayout>
```

Das Layout können wir nun der Start-Activity hinzufügen. Die Activity `Startseite` lädt das Layout während ihrer Erzeugung (Listing 5-2). Dies geschieht in der `onCreate`-Methode, die Teil der Activity ist und bei ihrer Erzeugung automatisch aufgerufen wird. Mit weiteren Methoden, die während der Erzeugung einer Activity automatisch durchlaufen werden, beschäftigen wir uns in Kapitel 15 über Lebenszyklen von Komponenten.

Eintrittspunkt einer Activity

Listing 5-2
Verknüpfung Layout mit Activity

```
package de.visionera.androidbuch.amando5.gui;

public class Startseite extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.startseite);
  }
}
```

Mehr müssen wir zunächst einmal nicht tun, um Layout, Views und Activity zusammenzubringen. Bei dem Layout (siehe Listing 5-1) handelt es sich um ein `LinearLayout`, das seine View-Elemente hintereinander oder untereinander anordnet. Es enthält drei Arten von View-Elementen: `TextView`, `Button` und `ImageView`. Wir werden in den folgenden Abschnitten darüber hinaus die wichtigsten Bestandteile, die für den Bau von Android-Oberflächen notwendig sind, kennenlernen.

Drei Arten von Views

In Android ist es möglich, Oberflächen einerseits in XML zu definieren, andererseits entspricht jedem in XML definierten Element ein konkretes Java-Objekt. Man könnte auf XML verzichten und die Oberflächen komplett in Java implementieren. Dies empfehlen wir jedoch nicht, da die Definition in XML wesentlich übersichtlicher ist und den Java-Quelltext schlanker hält. Ein weiterer Grund ist, dass Werkzeuge zur Oberflächenerstellung in Android XML als Ausgabe produzieren. Das im Android-Plugin eingebaute Werkzeug zur Oberflächengestaltung arbeitet z. B. mit XML.

Oberflächen nicht in Java erstellen!

Zurück zu unserem Layout. Bei der Definition der Views fällt auf, dass einige der XML-Attributwerte ein @-Zeichen enthalten. Beispielsweise findet man dort:

```
@style/TextStyleUeberschrift
@string/txt_startseiteanzeigen_intro
@color/schwarz
@drawable/amando_logo
```

Verweise auf Ressourcen definieren

Die durch @ eingeleiteten Attributwerte verweisen auf sogenannte *Ressourcen*. Dabei handelt es sich um Texte, Farbdefinitionen, Schlüsselwerte, Bilder oder ähnliche Nicht-Java-Bestandteile der Anwendung. Ressourcen spielen bei der Oberflächendefinition eine wichtige Rolle. Die nächsten Abschnitte befassen sich mit der Definition von Ressourcen und stellen die wichtigsten Ressourcenarten und ihre Anwendungsgebiete vor.

5.3 Ressourcen

Ein Programm besteht meist nicht nur aus Quellcode, sondern nutzt beispielsweise Grafiken und Piktogramme oder bringt eigene Audiodateien für bestimmte Aktionen mit. Wenn eine Anwendung mehrere Landessprachen unterstützen soll, müssen alle sichtbaren Texte mehrsprachig vorliegen. Auch Farbdefinitionen, Menüeinträge, Listen mit Daten oder Layouts werden nicht im Java-Quellcode definiert, sondern in XML als sogenannte *Ressourcen* hinterlegt. Je nach Format und Verwendungszweck unterscheiden wir folgende *Ressourcenarten*:

Definition in XML

1 Ein erstes Beispiel

In diesem Abschnitt werden wir ein erstes Android-Programm erstellen. Es dient dem schnellen Einstieg in die Programmierung von Android.

Dabei handelt es sich um ein Programm zur Berechnung der Umsatzsteuer. Man gibt den Ausgangsbetrag an, wählt aus, ob man den Brutto- oder Nettobetrag angegeben hat, und wählt die Umsatzsteuer in Prozent. Abbildung 1-1 zeigt das Formular für die Eingabe.

Brutto-Netto-Umrechner

Abb. 1-1
Beispielanwendung im Emulator

Die Berechnung wird gestartet, indem man die Schaltfläche »Berechnen« drückt. Das Ergebnis wird auf einer zweiten Bildschirmseite angezeigt (siehe Abb. 1-8 auf S. 21).

Wir werden uns bei der Erstellung des Programms nur auf die unbedingt notwendigen Elemente beschränken, um eine lauffähige Anwendung zu implementieren. Dabei werden wir noch nicht alles genau erklären. Im zweiten Teil des Buchs werden wir sehr viel tiefer ins Detail gehen und die offenen Fragen beantworten.

1.1 Projekt anlegen

Wir verwenden für die Entwicklung Android Studio. Die Installation wollen wir hier nicht wiederholen. Sie ist sehr gut unter [17] beschrieben.

Android-SDK

Android Studio sorgt bei der Installation gleich für die Einrichtung des Android SDK. Das Android SDK enthält alle Klassen, Bibliotheken und Tools zum Erstellen, Debuggen und Monitoren von Android Apps. Zum Erstellen eines Android-Projekts wählt man im Menü von Android Studio den Menüpunkt *File* und darunter *New Project* ....

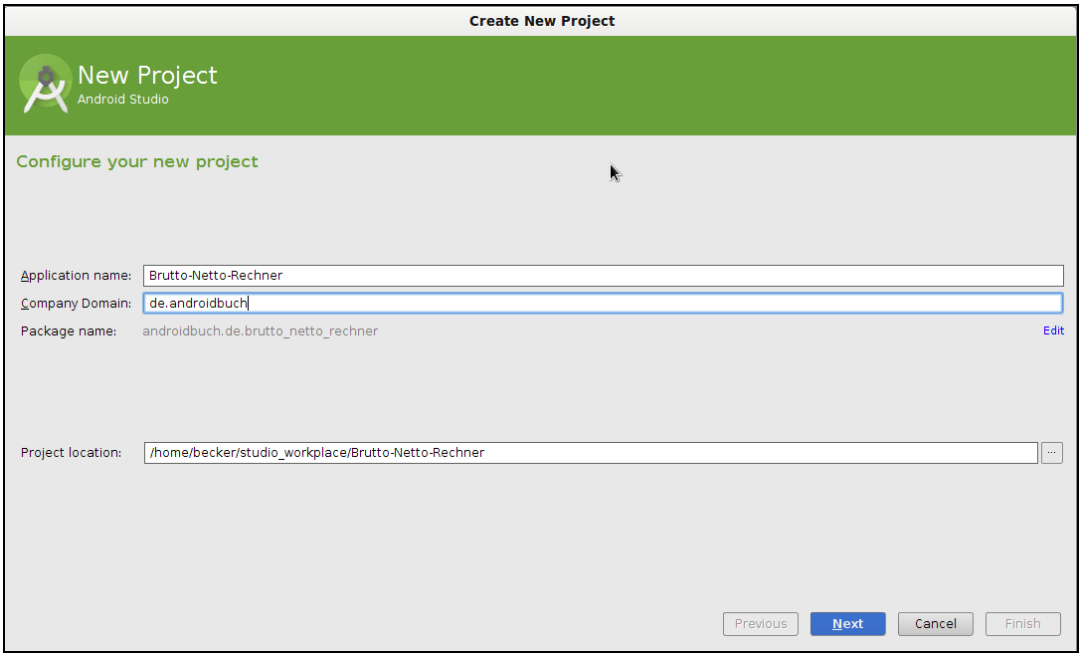

Abb. 1-2
Projekt auswählen

Die Anwendung erhält einen Namen. Dieser wird später nach Installation der App im Programmmanager des Android-Geräts angezeigt. Der zweite Eintrag ist der Domainname der Firma. Aus beiden Einträgen generiert Android Studio einen Paketnamen. Dieser ist hier leider nicht beeinflussbar. Als letzte Eingabe gibt man an, wo das Projekt gespeichert werden soll.

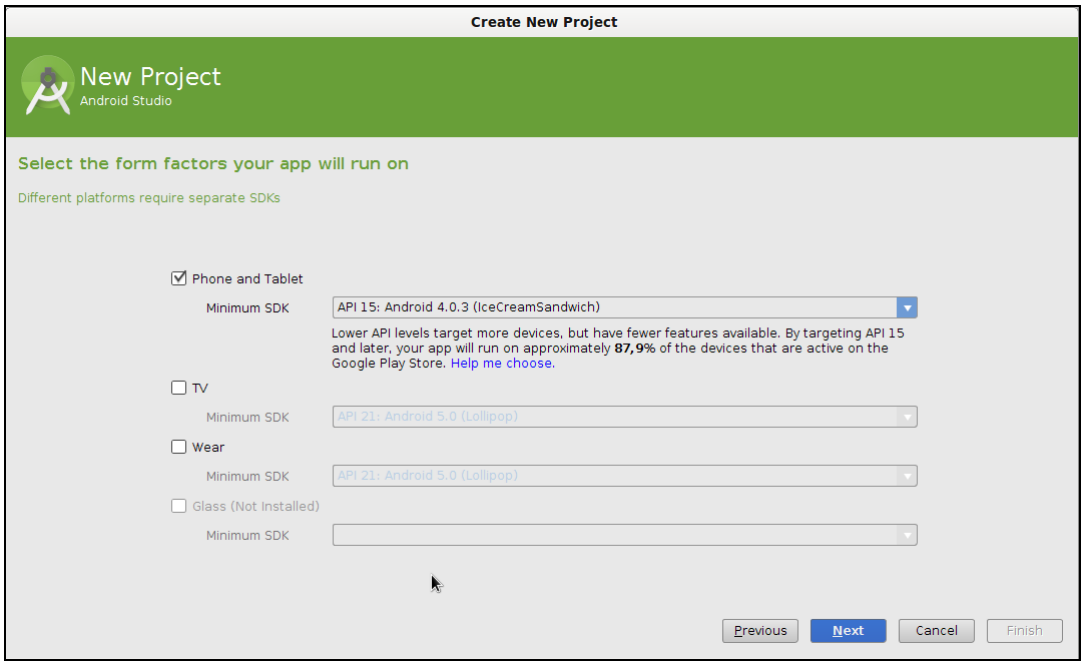

Abb. 1-3
Auswahl der Zielgerätegruppe

Smartphone bzw. Tablets, TV oder Wearables

Auf der nächsten Seite des Wizards gibt man die Gerätegruppe an, auf der die App später laufen soll. Derzeit (Android Studio 1.0.4) können Apps für drei Gruppen erstellt werden: Smartphone und Tablets, TV-Geräte und Wearables. Die Datenbrille Google Glass ist in Vorbereitung. Wir wählen »Phone and Tablet« und wählen das minimale SDK aus. Es handelt sich dabei um die minimale Android-Version (»Minimum Required SDK«). Hiermit legt man fest, bis zu welcher Version die Anwendung abwärtskompatibel sein soll. Man sollte hier zunächst eine möglichst niedrige Version wählen. Da es kaum noch Android-2-Geräte gibt und Tablets heute nahezu ausnahmslos mit Android 4 laufen, folgen wir dem Vorschlag von Android Studio und bleiben bei Android 4.0.3. Für unseren Brutto-Netto-Rechner würde sogar Android 1.6 reichen. Benötigt man während der Implementierung zwingend Methoden oder Klassen, die erst in einer späteren Android-Version hinzugekommen sind, kann man die minimale Android-Version nachträglich in der Datei `AndroidManifest.xml` erhöhen.

Einen Bildschirm weiter hat man die Möglichkeit, sich Quellcode automatisch generieren zu lassen. Man kann für eine Reihe gängiger Anwendungsfälle eine Startseite der App generieren lassen. Teils wird recht viel Quellcode produziert, und man spart eine Menge Arbeit. Die Bildschirmseite erhält ein passendes Design, was schnell an die eige-

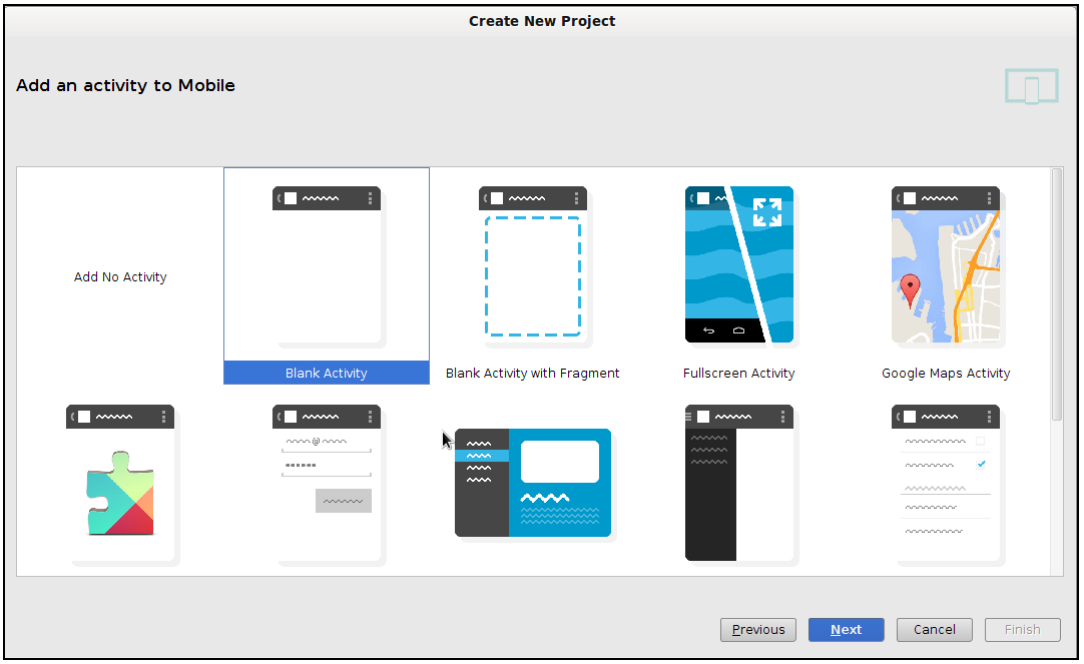

Abb. 1-4 *Design des Startbildschirms definieren*

nen Bedürfnisse angepasst werden kann. Sogar eine Activity mit Google Maps kann erstellt werden. Es lohnt sich, die verschiedenen Generatoren mal auszuprobieren und sich den Quellcode anzuschauen. Wir starten für unsere Anwendung mit der »Blank Activity«, einer leeren Bildschirmseite. Diese Bildschirmseite wird automatisch aufgerufen, sobald die Anwendung gestartet wird.

Es folgt die Eingabe einiger Parameter zur Startbildschirmseite. Jede Activity benötigt einen eindeutigen Namen, da es sich hier zeitgleich um einen Klassennamen handelt. Neben dem Java-Quellcode einer Bildschirmseite gibt es das Design der Bildschirmseite. In aller Regel wird das Design in XML erstellt. Daher gibt man den Namen der XML-Datei unter »Layout Name« an. Der Titel wird im Action Bar angezeigt. Hier sollte man also einen sprechenden Namen für die Seite verwenden. Falls die Seite ein Menü enthält, kann man noch den Namen der XML-Datei angeben, die später die Definition des Menüs enthalten wird.

Nach Fertigstellung legt der Projekt-Wizard eine Gradle-Projektstruktur an. Gradle ist das Build-Tool, welches bevorzugt bei Android zum Einsatz kommt und in Android Studio integriert ist. Gradle erzeugt aus den vielen einzelnen Dateien die fertige Android App und ist sehr mächtig. Kapitel 25 geht näher auf Gradle ein.

Abbildung 1-6 zeigt einen Ausschnitt aus Android Studio mit dem fertigen Brutto-Netto-Umrechner.

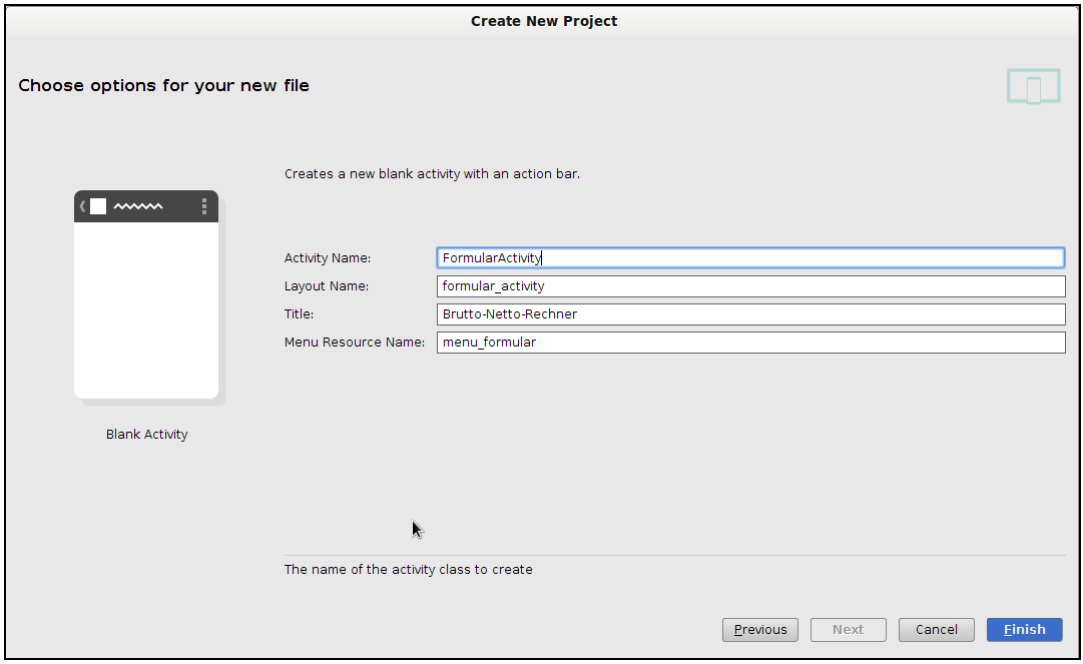

Abb. 1-5
Metadaten des Startbildschirms

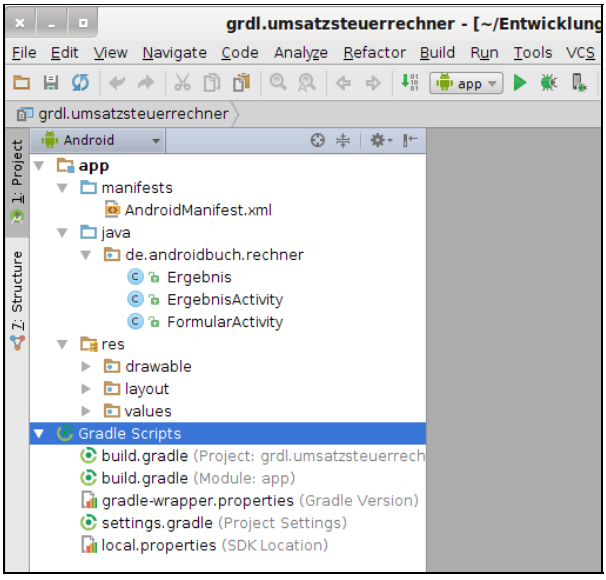

Abb. 1-6
Aufbau eines Gradle-Android-Projekts

- *manifests*: Ablageort für die Manifest-Dateien. Im Manifest werden die lose gekoppelten Programmteile der App zu einer Einheit zusammengefügt.
- *java*: Java-Quelltexte (u. a. auch unsere Activity `FormularActivity`)
- *res*: Ressourcen, d. h. alle Nicht-Java-Dateien. Hier werden u. a. die Dateien zur Definition der Oberflächen (Layouts), Bilder oder Textdefinitionen abgelegt.

Unterhalb von »Gradle Scripts« folgen dann die Skripte, die die App zusammenbauen. Gradle ist sehr mächtig, und es lohnt sich, sich hier näher einzuarbeiten.

Android 5

Mit Android 5 ist Gradle quasi zur Pflicht geworden. Wer zuvor Maven als Build-Tool verwendet hat, steht seit längerem vor dem Problem, dass notwendige Bibliotheken wie z. B. die Support Library nicht mehr über die offiziellen Maven Repositories verfügbar sind. Mit Gradle kein Problem! Da Gradle sehr gut in Android Studio eingebunden ist, sollte man spätestens mit Android 5 den Umstieg auf Android Studio wagen.

1.2 Die erste Activity

Startseite implementieren

Wir implementieren nun unsere erste Activity, die die Startseite unserer Anwendung anzeigen wird.

Listing 1-1 *Eingaben erfassen: FormularActivity*

```
package de.androidbuch.rechner;

import android.app.Activity;
import android.os.Bundle;

public class FormularActivity extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.formular_activity);
  }
}
```

Für den Anfang reicht es uns zu wissen, dass unsere eigenen Activities von der Android-API-Klasse `Activity` abgeleitet werden müssen. Activities implementieren die Logik einer einzelnen Bildschirmseite und behandeln Ereignisse wie beispielsweise einen Klick auf eine Schaltfläche oder einen Menüeintrag.

1.3 Layout definieren

XML GUI

Wenden wir uns nun der Erstellung unserer Eingabemaske zu. Die Maskenelemente werden in einer XML-Datei definiert. Der Vollständigkeit halber sei noch erwähnt, dass die Masken auch via Programmcode erstellt werden können. Dies ist aber, wie im Falle von Webanwendungen (JSP vs. Servlets), aus Gründen der Übersichtlichkeit und Wartbarkeit stets die zweite Wahl und wird daher nicht Thema dieses Buches sein.

Der Assistent zum Anlegen eines Android-Projekts hat bereits eine solche XML-Datei `res/layout/formular_activity.xml` erstellt, die in Listing 1-2 dargestellt ist.

Listing 1-2
Ein einfaches Layout

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

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true"
    android:text="@string/hello_world" />

</RelativeLayout>
```

Es gibt verschiedene Layout-Typen.

Ähnlich wie bei Swing-Anwendungen können verschiedene Layouts für den Aufbau der Maske verwendet werden. Beim Erstellen eines Android-Projekts wird automatisch ein *RelativeLayout* generiert. *RelativeLayouts* eignen sich gut für komplexe Layouts, die auf verschiedenen Bildschirmauflösungen laufen sollen. Zudem sind sie recht performant. Nachteil ist, dass sie schwerer zu implementieren sind. Daher werden wir mit einem einfacheren Layout starten, dem *LinearLayout*.

Das XML-Element `TextView` in Listing 1-2 enthält ein Attribut `android:text`. Hier handelt es sich um einen Verweis auf eine Zeichenkettendefinition. Sie befindet sich in der Datei `strings.xml` im Ordner `/res/values`. Die Datei hat folgenden Inhalt:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">Brutto-Netto-Rechner</string>
  <string name="hello_world">Hello world!</string>
  <string name="action_settings">Settings</string>
</resources>
```

Text wird ausgelagert.

Der Schlüssel für den Text, der in dem Anzeigeelement `TextView` dargestellt werden soll, lautet »`hello`«. Er wird in der `TextView` zur Laufzeit durch den Wert aus `strings.xml` ersetzt. Das Attribut heißt `android:text`. Mit `@string` wird dem Ressourcen-Compiler mitgeteilt, dass in den Dateien im Ordner `/res/values` nach einem XML-Attribut vom Typ String gesucht werden soll, dessen `name`-Attribut `hello_world` lautet.

Der nächste Schritt ist nun, dieses automatisch generierte Layout für unsere Zwecke anzupassen. Dazu überlegen wir uns, welche Oberflächenelemente für den Brutto-Netto-Rechner nötig sind (siehe Tab. 1-1).

Tab. 1-1 *Feldliste »Eingabe erfassen«*

Feldname	Funktion	Darstellung
betrag	Fließkommazahl (Euro)	Texteingabe
art	»Brutto«, »Netto«	Radiobutton
umsatzsteuer	»19 Prozent«, »16 Prozent«, »7 Prozent«	Auswahlliste

Nun passen wir die Oberfläche an unsere Anforderungen an. Dazu definieren wir die Formularelemente aus Tabelle 1-1 in XML. Die Datei `formular_activity.xml` sieht nach der Erweiterung wie folgt aus:

Listing 1-3 *formular_activity.xml für den Brutto-Netto-Rechner*

```
<?xml version="1.0" encoding="utf-8"?>
<tools:LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FormularActivity">

  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/txt_anweisung" />

  <EditText android:id="@+id/edt_betrag"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="numberDecimal"/>

  <RadioGroup android:id="@+id/rg_art"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
```

```
    <RadioButton android:id="@+id/rb_art_netto"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/txt_netto"
      android:textSize="16dp"
      android:checked="true" />

    <RadioButton android:id="@+id/rb_art_brutto"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/txt_brutto"
      android:textSize="16dp" />
  </RadioGroup>

  <Spinner android:id="@+id/sp_umsatzsteuer"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:drawSelectorOnTop="true"
   android:entries="@array/ust_anzeige"
   android:entryValues="@array/ust_werte" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/txt_berechnen"
    android:layout_gravity="center"
    android:onClick="onClickBerechnen" />

</tools:LinearLayout>
```

Wir haben das *RelativeLayout* in ein *LinearLayout* geändert. Die *TextView* wurde angepasst. Sie holt sich nun ihren Text über das Attribut `txt_anweisung` aus der Datei */res/values/strings.xml*. Neu hinzugekommen sind ein Texteingabefeld (`EditText`), eine `RadioGroup`, bestehend aus zwei `RadioButtons`, ein `Spinner` und eine Schaltfläche. Ein Spinner ist eine Auswahlliste. In unserem Beispiel ist die Wertebelegung für den Spinner statisch, so dass wir sie in eine weitere XML-Datei im `values`-Verzeichnis auslagern können (Listing 1-4). Wir geben ihr den Namen `arrays.xml`. In `string-array` stehen die Texte, die im Spinner angezeigt werden sollen. Wir haben hier auch die Prozentwerte als Integer hinterlegt. In `integer-array` stehen die Prozentwerte, die wir später im Programmcode auf den ausgewählten Eintrag im Spinner mappen. Denn dort erhalten wir nur die ausgewählte Position im Spinner. Aus dieser müssen wir dann auf den Prozentwert schließen.

Spinner = Auswahlliste

Listing 1-4 *res/values/arrays.xml*

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string-array name="ust_anzeige">
    <item>19 Prozent</item>
    <item>16 Prozent</item>
    <item>7 Prozent</item>
  </string-array>
  <integer-array name="ust_werte">
    <item>19</item>
    <item>16</item>
    <item>7</item>
  </integer-array>
</resources>
```

Nun erweitern wir die Datei `strings.xml` im Ordner `/res/values`. Das Formular enthält einige Verweise auf String-Ressourcen. Das Auslagern von Texten in eine eigene Ressourcendatei ist sinnvoll, da es dadurch später sehr einfach ist, die Anwendung mehrsprachig zu machen. Zwei der automatisch generierten Einträge löschen wir und fügen gleich alle Texte hinzu, die wir jetzt oder später brauchen.

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

    <string name="app_name">Brutto-Netto-Rechner</string>

    <!-- Formulartexte -->
    <string name="txt_anweisung">Geben Sie einen Brutto-
        oder Nettobetrag ein und lassen Sie sich die Umsatzsteuer
        errechnen:</string>
    <string name="txt_netto">Netto</string>
    <string name="txt_brutto">Brutto</string>
    <string name="txt_nettobetrag">Nettobetrag:</string>
    <string name="txt_umsatzsteuer">Umsatzsteuer:</string>
    <string name="txt_bruttobetrag">Bruttobetrag:</string>
    <string name="txt_berechnen">Berechnen</string>

    <!-- Menüeintrag -->
    <string name="mnu_ende">Beenden</string>

</resources>
```

Eine Anwendung starten

Geschafft! Unser Formular zur Erfassung der Eingabewerte ist fertig. Nun muss die Anwendung nur noch im Emulator gestartet werden. Dazu drückt man den grünen Pfeil in der Mitte der Menüleiste (»Run App«). Es öffnet sich ein Dialog zur Auswahl des Zielgeräts. Wahlweise kann die App auf einem echten Android-Gerät gestartet werden, falls

eines über USB angeschlossen ist. Ist kein Gerät angeschlossen, bleibt einem der Emulator. Man wählt einen der vorkonfigurierten Emulatoren aus und die App wird im Emulator gestartet. Nach einer Wartezeit sollte die folgende Bildschirmseite erscheinen (Abb. 1-7).

Abb. 1-7
Beispielanwendung im Emulator

Tipp!

Der Emulator braucht recht lange zum Starten. Starten Sie ihn daher zu Beginn einmal und schließen Sie das Emulatorfenster nicht. Jeder weitere Start der Anwendung erfolgt dann sehr schnell.

1.4 Activities aufrufen

Interaktionen zwischen Activities

Beschäftigen wir uns nun mit Interaktionen zwischen Activities. Wenn die Schaltfläche »Berechnen« gedrückt wird, soll eine neue Seite erscheinen, auf der das Ergebnis der Berechnung angezeigt werden soll. Abhängig davon, ob die eingegebene Zahl einen Brutto- oder Nettobetrag darstellt, soll entsprechend der Prozentzahl der korrekte Umsatzsteuerbetrag angezeigt werden. Wir brauchen dafür eine weitere Bildschirmseite, die folgende Ergebnisfelder enthalten soll:

- Nettobetrag: Betrag ohne Umsatzsteuer
- Umsatzsteuerbetrag
- Bruttobetrag: Betrag inkl. Umsatzsteuer

Mehr Aktivität

Für die zweite Bildschirmseite werden wieder ein Layout und eine Activity benötigt. Anhand dieser Seite demonstrieren wir

- die Verwendung von dynamischen Inhalten in Activities,
- die Interaktion zwischen Activities,
- die Verwendung von Schaltflächen.

Beginnen wir mit der Definition der Oberfläche. Hierzu erzeugen wir das Layout (Seiten-Beschreibungsdatei) `ergebnis_anzeigen.xml` und legen sie unterhalb von `/res/layout` ab.

Hinweis

Der Name von Dateien unterhalb des Ordners `/res` darf nur aus Ziffern und Kleinbuchstaben sowie dem Unterstrich bestehen. Der eingängigere Name `ergebnisAnzeigen.xml` (die Java-übliche »Camel-Case«-Schreibweise) wäre daher nicht erlaubt. Wir verwenden die Schreibweise generell in den Ressourcendateien (vgl. Listing 1-4) zur Namensgebung, auch wenn dort die Camel-Case-Schreibweise erlaubt ist.

Listing 1-5
Table Layout, Ergebnis anzeigen

```
<?xml version="1.0" encoding="utf-8"?>
<tools:TableLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".ErgebnisActivity">

  <TableRow>
    <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/txt_nettobetrag" />
    <TextView
      android:id="@+id/txt_nettobetrag"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
  </TableRow>
```

```
  <TableRow>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/txt_umsatzsteuer" />
    <TextView
      android:id="@+id/txt_umsatzsteuer"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
  </TableRow>

  <TableRow>
    <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:text="@string/txt_bruttobetrag" />
    <TextView
      android:id="@+id/txt_bruttobetrag"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
  </TableRow>

</tools:TableLayout>
```

TableLayout

Für diese View verwenden wir ein `TableLayout`, da die Ergebnisse in Tabellenform dargestellt werden sollen. Ansonsten gleicht diese Bildschirmseitendefinition der vorherigen.

1.5 Das Android-Manifest

Die mit diesem Layout verknüpfte Activity `ErgebnisActivity` muss dem System erst noch bekannt gemacht werden. Dazu wird sie im `AndroidManifest.xml` des Projektes registriert. Listing 1-6 zeigt das vollständige Android-Manifest der Einführungsanwendung.

Listing 1-6
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=
    "http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="de.androidbuch.rechner"
    android:versionCode="3"
    android:versionName="3.0">
```

```
  <application android:icon="@drawable/icon"
      android:label="@string/app_name"
      android:theme="@android:style/Theme.Holo.Light" (1)
      tools:ignore="NewApi" >

    <activity android:name=".FormularActivity"
        android:label="@string/app_name">
      <intent-filter>
        <action android:name=
          "android.intent.action.MAIN" />
        <category android:name=
          "android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>

    <activity android:name=".ErgebnisActivity" /> (2)
  </application>

</manifest>
```

Das Android-Manifest liegt in der Rubrik »manifests« des Android-Studio-Projekts. Es wurde automatisch generiert, als wir unser Projekt angelegt haben. Wir führen nun die notwendigen Erweiterungen durch.

Innerhalb des `<application>`-Elements findet die Deklaration der Activities statt. Für die Activity `FormularActivity` wird ein sogenannter Intent-Filter definiert. Mit Intents und Intent-Filtern werden wir uns in Kapitel 7 ausführlicher befassen. Ein Intent repräsentiert einen konkreten Aufruf einer anderen Activity, eines Hintergrundprozesses oder einer externen Anwendung. Wir können den englischen Begriff mit »Absichtserklärung« übersetzen. Der hier verwendete Intent-Filter sorgt dafür, dass die Brutto-Netto-Rechner-Anwendung gestartet wird, indem die Activity `FormularActivity` angezeigt wird. Der Intent selbst wird vom Android-System verschickt, sobald man die Anwendung startet.

Intent ruft Activity auf.

Wir haben im Manifest zwei Änderungen durchgeführt. Zunächst haben wir durch die Verwendung eines sogenannten »Themes« dafür gesorgt, dass unsere Anwendung etwas freundlicher und heller aussieht, indem unter anderem ein heller Hintergrund verwendet wird statt einem schwarzen (1). Dazu haben wir das `<application>`-Element um ein XML-Attribut erweitert. Anschließend haben wir die Activity zur Darstellung des Ergebnisses eingefügt (2).

Ergebnis berechnen

Zum Berechnen des Ergebnisses benötigen wir eine Schaltfläche auf der Bildschirmseite `FormularActivity`. Im Layout (siehe Listing 1-3) der `FormularActivity` haben wir eine Schaltfläche (`Button`) definiert. Die enthält das Attribut `android:onClick`. Der Wert dieses Attributs ist ein Me-

thodenname (`onClickBerechnen`), den es in der zum Layout gehörenden Activity zu implementieren gilt.

Als nächsten Schritt lassen wir die beiden Activities miteinander kommunizieren. Konkret soll `FormularActivity` nach Anklicken der Schaltfläche »Berechnen« die Activity `ErgebnisActivity` aufrufen. Dabei sollen die Formularwerte übertragen werden, das Ergebnis berechnet und auf dem Bildschirm dargestellt werden.

Werte übergeben

In unserem Fall muss `FormularActivity` einen Intent erzeugen, ihn mit Übergabeparametern versehen und anschließend ausführen, damit `ErgebnisActivity` aufgerufen wird. Listing 1-7 zeigt den dafür erforderlichen Code, den wir in die schon existierende `FormularActivity` einfügen.

Listing 1-7
Aufruf anderer Activities per Intent

```
public static final String BETRAG_KEY = "betrag";
public static final String BETRAG_ART = "art";
public static final String UST_PROZENT = "ust";

public void onClickBerechnen(View button) { // (1)
    // Betrag:
    final EditText txtBetrag =
        (EditText) findViewById(R.id.edt_betrag); // (2)
    final String tmpBetrag =
        txtBetrag.getText().toString(); // (3)
    float betrag = 0.0f;
    if (tmpBetrag.length() > 0) {
      betrag = Float.parseFloat(tmpBetrag);
    }

    // Art des Betrags (Brutto, Netto):
    boolean isNetto = true;
    final RadioGroup rg =
        (RadioGroup) findViewById(R.id.rg_art);
    switch (rg.getCheckedRadioButtonId()) {
      case R.id.rb_art_netto:
        isNetto = true;
        break;
      case R.id.rb_art_brutto:
        isNetto = false;
        break;
      default:
    }

    // Prozentwert Umsatzsteuer:
    final Spinner spinner =
        (Spinner) findViewById(R.id.sp_umsatzsteuer);
    final int pos = spinner.getSelectedItemPosition();
```

```
        final int[] prozentwerte =
            getResources().getIntArray(R.array.ust_werte);
        final int prozentwert = prozentwerte[pos];

        final Intent intent = new Intent(this, // (4)
            ErgebnisActivity.class);

        intent.putExtra(BETRAG_KEY, betrag); // (5)
        intent.putExtra(BETRAG_ART, isNetto);
        intent.putExtra(UST_PROZENT, prozentwert);

        startActivity(intent); // (6)
    }
```

Programmablauf

Wir wollen hier im Einstiegsbeispiel den Quellcode noch nicht im Detail erklären, dies geschieht später ausführlich in den entsprechenden Kapiteln. Wir geben lediglich einen Überblick über die Funktionsweise des Programmcodes.

- Methode `onClickBerechnen` wird ausgeführt, wenn die Schaltfläche »Berechnen« gewählt wurde (1).
- `findViewById` liefert Zugriff auf ein Oberflächenelement (View genannt, z. B. das Texteingabefeld) (2).
- Es wird der Wert aus dem View-Element des Formulars geholt (3).
- Es wird ein Intent zum Aufruf der Folge-Activity erzeugt (4).
- Der Wert aus dem Formular wird dem Intent als Übergabeparameter hinzugefügt (5).
- Die Folge-Activity wird mit Hilfe des Intents aufgerufen (6).

Ergebnis berechnen

Auf der Gegenseite muss nun die Activity `ErgebnisActivity` erstellt werden. In ihrer `onCreate`-Methode wird der Intent entgegengenommen, und seine Daten werden verarbeitet. Zuvor implementieren wir jedoch eine Hilfsklasse, die die Berechnung der Ergebniswerte übernimmt. Ihr kann man die Formularwerte übergeben und den Umsatzsteuerbetrag berechnen lassen. Die entsprechenden Attribute der Klasse liefern die Ergebnisse (Netto-, Brutto- und Umsatzsteuerbetrag), welche in der `ErgebnisActivity` zur Anzeige gebracht werden sollen.

Listing 1-8 *Hilfsklasse zur Berechnung des Ergebnisses*

```
public class Ergebnis {
  public float betrag;
  public boolean isNetto;
  public int ustProzent;

  public float betragNetto;
  public float betragBrutto;
  public float betragUst;
```

```
  public void berechneErgebnis() {
    // Berechne Bruttobetrag aus Nettobetrag:
    if (isNetto) {
      betragNetto = betrag;
      betragUst = betrag * ustProzent / 100;
      betragBrutto = betrag + betragUst;
    } else {
      // Berechne Nettobetrag aus Bruttobetrag:
      betragBrutto = betrag;
      betragUst = betrag * ustProzent /
          (100 + ustProzent);
      betragNetto = betrag~-- betragUst;
    }
  }
}
```

Der Einfachheit halber haben wir hier auf Getter- und Setter-Methoden verzichtet. Nach dem Erstellen der Ergebnis-Klasse können wir die zugehörige Activity implementieren. Listing 1-9 zeigt den Quellcode.

Listing 1-9
Activity zur Anzeige des Ergebnisses

```
public class ErgebnisActivity extends Activity {

  @Override
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    setContentView(R.layout.ergebnis_anzeigen);

    final Bundle extras = getIntent().getExtras();

    if (extras != null) {
      final Ergebnis ergebnis = new Ergebnis();

      ergebnis.betrag =
          extras.getFloat(FormularActivity.BETRAG_KEY);
      ergebnis.isNetto =
          extras.getBoolean(FormularActivity.BETRAG_ART,
              true);
      ergebnis.ustProzent =
          extras.getInt(FormularActivity.UST_PROZENT);

      zeigeErgebnis(ergebnis);
    }
  }
```

```
  /**
   * @param ergebnis
   */
  private void zeigeErgebnis(Ergebnis ergebnis) {
    setTitle("Ergebnis");

    ergebnis.berechneErgebnis();

    final TextView txtNettobetrag =
        (TextView) findViewById(R.id.tv_nettobetrag);
    txtNettobetrag.setText(String.valueOf(
        ergebnis.betragNetto));

    final TextView txtUmsatzsteuer =
        (TextView) findViewById(R.id.tv_umsatzsteuer);
    txtUmsatzsteuer.setText(
        String.valueOf(ergebnis.betragUst));

    final TextView txtBruttobetrag =
        (TextView) findViewById(R.id.tv_bruttobetrag);
    txtBruttobetrag.setText(
        String.valueOf(ergebnis.betragBrutto));
  }
}
```

In der `onCreate`-Methode holen wir uns die Formularwerte aus dem Intent, den wir von `FormularActivity` erhalten haben. Die Methode `zeigeErgebnis` führt die Berechnung in der Klasse `Ergebnis` durch. Wir holen uns die für die Ergebnisanzeige nötigen View-Elemente aus dem Layout (siehe Listing 1-5) und setzen die Ergebniswerte mittels der Methode `setText`. Abbildung 1-8 zeigt die Ergebnisseite.

Zugriff auf Views

Abschließend erweitern wir die Anwendung noch um ein Menü. Dazu legen wir eine zusätzliche XML-Datei an, die den Menüeintrag enthalten soll. Ist noch kein Menü vorhanden, legt man zunächst einen neuen Ordner im Verzeichnis `/res` an. Dazu navigiert man mit der Maus im Projektverzeichnis von Android Studio auf den Ordner `/res` und klickt auf die rechte Maustaste. Man wählt *New -> Android resource file* aus. Abbildung 1-9 zeigt den Vorgang.

Wir erhalten daraufhin eine neue Datei `/res/menu/menu.xml`. Android Studio bietet eine ganze Reihe Assistenten, die es einem erheblich erleichtern, neue Ressourcen, Activities, Layouts oder ähnliche Bestandteile einer App anzulegen. Es lohnt sich, sich mit diesem Assistenten vertraut zu machen.

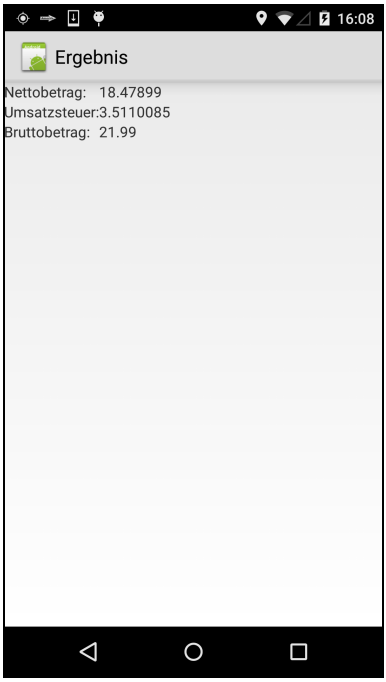

Abb. 1-8
Die Ergebnisseite des Umsatzsteuerrechners

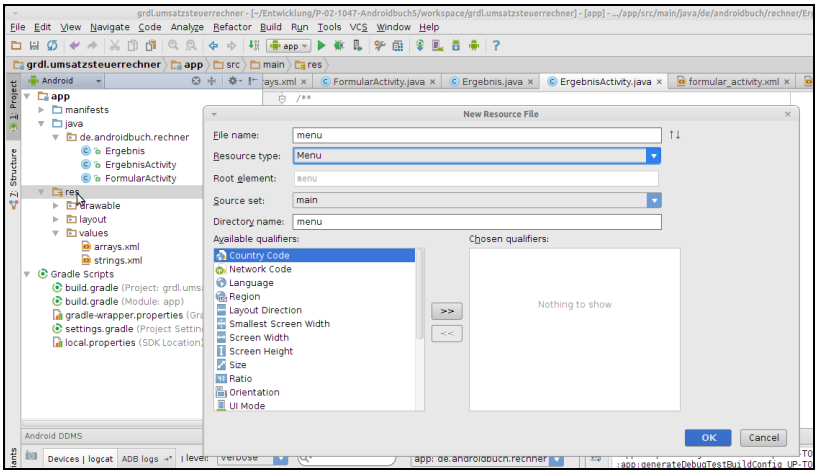

Abb. 1-9
Ein Menü anlegen

Wir fügen nun einen Menüeintrag in der Datei `menu.xml` hinzu, um später die Anwendung zu schließen (dies ist im Grunde nicht richtig, da sich Android-Anwendungen nicht beenden lassen. Aber dazu kommen wir später).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">

  <item
      android:id="@+id/opt_beenden"
      android:title="@string/mnu_ende"
      app:showAsAction="ifRoom"
      android:icon="@android:drawable/ic_menu_close_clear_cancel" />
</menu>
```

Unser Menü enthält nur einen Eintrag. Später wird der Eintrag anhand seiner Id `opt_beenden` erkannt. Der Text des Menüeintrags kommt wieder aus der Datei `strings.xml`. Das Attribut `showAsAction` bestimmt das Aussehen. In unserem Fall sorgen wir dafür, dass Geräte ohne eine Menütaste ein Icon im ActionBar, also der oberen Titelleiste der Bildschirmseite, anzeigen. Als Icon verwenden wir ein Icon, welches uns das Android SDK zur Verfügung stellt, was durch `@android:drawable` kenntlich gemacht wird.

Nun müssen wir noch das Menü in der `FormularActivity` bekannt machen und die Activity in die Lage versetzen, auf Menüeinträge zu reagieren. Wir erweitern die Activity um die folgenden zwei Methoden:

Listing 1-10 *Implementierung Standardmenü*

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  getMenuInflater().inflate(R.menu.menu, menu);
  return super.onCreateOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case R.id.opt_beenden:
      finish();
      break;
    // ...
    default:
  }

  return super.onOptionsItemSelected(item);
}
```

Die Methode `onCreateOptionsMenu` lädt unsere gerade erstellte Menüdefinition. In der Methode `onOptionsItemSelected` wird auf die Auswahl eines Menüitems reagiert. In unserem Fall beenden wir mit `finish` die Activity (nicht die Anwendung).

In diesem Beispiel ging es darum, das Prinzip der losen Kopplung von Komponenten, hier zwei Activities, zu verdeutlichen. Wir haben gesehen, wie eine Activity einen Intent verschickt, sobald die Schaltfläche *Berechnen* gedrückt wurde. Die auf Intents basierende offene Kommunikationsarchitektur stellt eine der Besonderheiten von Android dar.

Das Ergebnis im Emulator

Zum Abschluss dieser Einführung schauen wir uns das Ergebnis im Emulator an (siehe Abb. 1-8). Der vollständige Quellcode steht unter `http://www.androidbuch.de` zum Herunterladen bereit. Es empfiehlt sich, dieses Beispiel auf eigene Faust zu verändern und die Resultate unmittelbar im Emulator zu betrachten.

1.6 Fazit

In diesem Abschnitt gaben wir Ihnen einen kurzen Einblick in die Programmierung von Android-Geräten. Kennengelernt haben wir die Grundbestandteile

- Bildschirmseiten-Erstellung
- Formularverarbeitung
- Interaktion zwischen Bildschirmseiten
- Schaltflächen
- Menüs
- Start der Laufzeitumgebung und des Emulators

sowie die Android-Artefakte

- Activity
- Layout
- View
- Intent
- Android-Manifest

Nun ist es an der Zeit, sich ein wenig mit der Theorie zu befassen. Der Rest dieses ersten Teils beschäftigt sich mit den Konzepten hinter Android.

Dipl.-Inform. Arno Becker ist bei der visionera GmbH verantwortlich für den Bereich »Mobile Lösungen«. Nach langjähriger Erfahrung mit Java ME beschäftigte er sich von Beginn an intensiv mit Android. Als technischer Leiter in zahlreichen Android-Projekten wie beispielsweise »Finanzblick«, »Aeonos Zeitwirtschaft« oder »Smazaar« hat er Erfahrung in komplexen Projekten mit Datenaustausch über das Internet gesammelt. Diese Kenntnisse gibt er als Berater, Fachartikelautor, in Schulungen und auf Vorträgen weiter.

Dipl.-Inform. Marcus Pant arbeitet für die visionera GmbH als Berater in Kundenprojekten. Seine Schwerpunkte liegen in der Entwicklung von Java-EE-Systemen und im Projektmanagement. Er beschäftigt sich seit 2007 mit Android und hat sich auf die Bereiche Datenspeicherung und Tests spezialisiert.

Arno Becker · Marcus Pant

Android 5

Programmieren für Smartphones und Tablets

4., aktualisierte und erweiterte Auflage

dpunkt.verlag

Arno Becker · Arno.Becker@visionera.de

Marcus Pant · Marcus.Pant@visionera.de

Lektorat: René Schönfeldt
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Da-TeX, Leipzig
Herstellung: Birgit Bäuerlein
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

ISBN:
Buch 978-3-86490-260-4
PDF 978-3-86491-661-8
ePub 978-3-86491-662-5

4., aktualisierte und erweiterte Auflage 2015

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Vorwort

Am 12. November 2007 veröffentlichte Google eine Vorabversion des Android-SDK, einer Entwicklungsumgebung für die Android-Plattform. Die positive Reaktion darauf verdeutlichte schon damals, wie groß das Interesse der Entwickler und der Hersteller an einer offenen Plattform für Embedded Systems schon zu diesem Zeitpunkt war. Android läuft mittlerweile auf einer Vielzahl verschiedener Gerätetypen. Dank zahlloser Apps für Android lässt sich die Funktionsvielfalt der Geräte nach dem Kauf erweitern und an die eigenen Bedürfnisse anpassen. Doch irgendjemand muss die Apps schreiben …

Android läuft nicht nur auf Mobiltelefonen.

Android wurde ursprünglich als Plattform für Mobiltelefone konzipiert. Heute sind schon Android-Geräte aus den Bereichen Auto-Infotainment, Home Entertainment, Fernseher, Netbook, Tablet-PC oder Festnetztelefon auf dem Markt. Auch wenn die Geräte grundsätzlich verschieden sind und z. B. nicht immer ein GSM-Modul zum Telefonieren oder ein GPS-Modul für die Positionsbestimmung besitzen, so haben sie doch eines gemeinsam: Auf ihnen laufen Android-Programme.

Am 17. Oktober 2014 wurde Android 5 offiziell vorgestellt. Schon im Sommer zuvor stand den Entwicklern eine Vorabversion des Android 5-SDK zur Verfügung. Die hier vorliegende 4. Auflage wurde umfassend überarbeitet und auf Android 5 aktualisiert. Als Entwicklungsumgebung haben wir Android Studio verwendet. Als Buildsystem zum Bauen der Apps dient Gradle, welches gut in Android Studio integriert ist.

Ein Buch zu Android

Ziel: Grundprinzipien praktisch vermitteln

Wir werden in diesem Buch die Grundprinzipien von Android vorstellen. Dabei geht es uns nicht darum, die Dokumentation von Android abzuschreiben, sondern anhand von Codebeispielen einen zielgerichteten Blick auf die grundlegenden Themen der Softwareentwicklung mit dem Android-SDK zu werfen.

Wir konzentrieren uns auf Kernthemen, die fast jede Android-Anwendung benötigt: Oberflächen und Menüs, Datenübertragung,

Datenspeicherung, Hintergrunddienste, GPS und lose Kopplung von Android-Komponenten. Weniger von Interesse sind für uns multimediale Themen, wie zum Beispiel die Wiedergabe von Videos oder die Audio-Schnittstelle.

Wir werden die Bausteine von Android kennenlernen und uns anschauen, wie diese miteinander interagieren. Wir erklären, was hinter den Kulissen von Android passiert und wie man mit diesem Wissen stabile und performante Anwendungen schreibt. Darüber hinaus werden wir zeigen, wie man eine Android-Anwendung »marktreif« macht.

Warum dieses Buch?

Den ersten Einstieg in Android zu finden ist, dank der guten Herstellerdokumentation, einfach. Allerdings reichen diese Informationen nicht immer aus, professionelle Software für Android zu entwickeln. Dazu fehlt es manchmal an der nötigen Detailtiefe. Daher haben wir Bedarf gesehen, die Kernthemen von Android in einem deutschsprachigen Buch ausführlich vorzustellen.

Für wen ist dieses Buch?

Das Buch richtet sich in erster Linie an Softwareentwickler. Grundkenntnisse der Programmiersprache Java sollten vorhanden sein.

Wir sprechen mit dem Buch aber auch technische Projektleiter an. Viele Fragestellungen und Herausforderungen des »Mobile Business«, wie z. B. die Themen Sicherheit und Verschlüsselung, werden in das Buch mit einbezogen.

Aufbau des Buchs

Teil I

Wir werden in Teil I des Buchs mit einem einfachen Beispiel beginnen, welches aber schon über die übliche Hello-World-Anwendung hinausgeht. Es stellt die wichtigsten Elemente einer Anwendung vor. Dem folgt ein wenig Theorie, die für das Verständnis von Android wichtig ist.

Teil II

In Teil II steigen wir weiter in die Praxis ein. An einem durchgängigen Beispiel stellen wir Kapitel für Kapitel wichtige Elemente des Android-SDK vor. Jedes Kapitel enthält einen theoretischen und einen praktischen Teil. Der theoretische Teil soll helfen, ein tieferes Verständnis für die Arbeitsweise der einzelnen Komponenten und Bestandteile von Android zu vermitteln. Im Praxisteil wenden wir dann das Wissen an.

Teil III

In Teil III befassen wir uns mit weiterführenden Themen rund um die Android-Anwendungsentwicklung: Debugging, Anwendungen »marktreif« machen, Sicherheit und Verschlüsselung, Testen und dem Buildsystem *Gradle*.

Wie lese ich dieses Buch?

Wir empfehlen Ihnen, das Einsteigerbeispiel in Teil I durchzugehen. Der Rest von Teil I ist theoretischer Natur und kann jederzeit separat gelesen werden.

Teil II sollten Sie in der vorgegebenen Reihenfolge der Kapitel durcharbeiten, da diese aufeinander aufbauen.

Teil III kann isoliert betrachtet werden. Wer gleich von Beginn des Buchs an viel selbst mit den Codebeispielen experimentiert, kann sich ein paar gute Tipps in Kapitel 19 (Debugging und das DDMS-Tool) holen.

Geschlechtsneutralität

Mit dem Thema »Geschlechtsneutralität« halten wir es wie Peter Rechenberg in [44]: »Rede ich von ›dem Leser‹, meine ich ja keinen *Mann*, sondern einen *Menschen*, und der ist nun einmal im Deutschen grammatisch männlich. Selbstverständlich ist mit ›dem Leser‹ der männliche *und* der weibliche Leser gemeint.«

Die Website zum Buch

Auf der Website zum Buch (www.androidbuch.de) finden Sie den Quelltext der Programmierbeispiele, Errata, ein Glossar mit Android-Fachbegriffen sowie weiterführende Links zum Thema Android-Entwicklung.

Danksagung

Wir danken unseren Familien, Partnern, Freunden und Kollegen für die Unterstützung und die Geduld.

Ebenfalls danken möchten wir dem *dpunkt.verlag*, insbesondere Herrn Schönfeldt, für die angenehme und produktive Zusammenarbeit.

Bedanken möchten wir uns auch bei allen Lesern der Vorauflagen des Buchs, für ihre Kommentare und die vielen wertvollen und hilfreichen Hinweise.