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

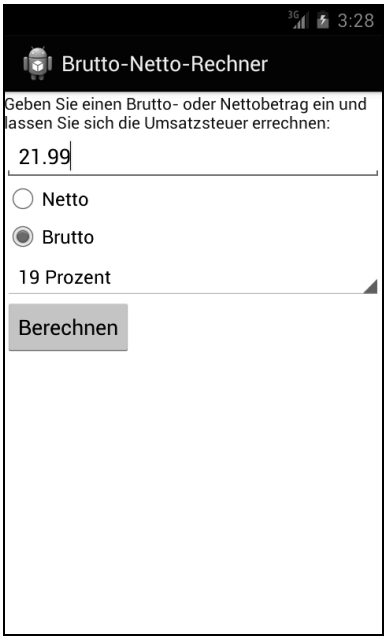


Abb. 1-1
Beispielanwendung im Emulator

Die Berechnung wird gestartet, indem man den Optionsmenüpunkt »Umrechnen« wählt. Das Ergebnis wird auf einer zweiten Bildschirmseite angezeigt (siehe Abb. 1-9 auf Seite 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.

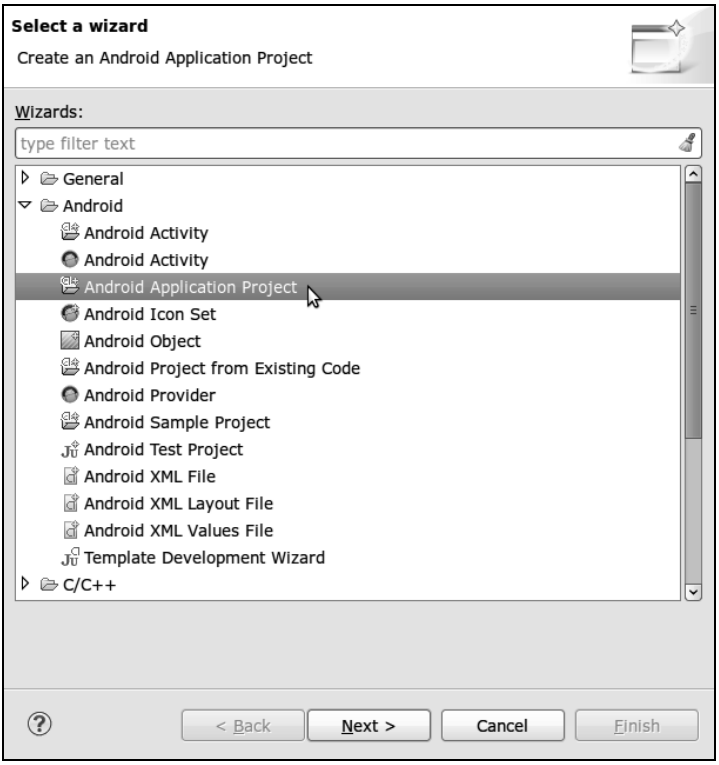
1.1 Projekt anlegen

Wir verwenden für die Entwicklung Eclipse mit dem Android-Plugin. Die Installation wollen wir hier nicht wiederholen. Sie ist sehr gut unter [42] beschrieben.

Android-Plugin verwenden

Mit Hilfe des Android-Plugin für Eclipse ist die Erstellung eines Projekts recht einfach. In Eclipse wählt man den Menüpunkt *File* und darunter *New -> Other* . . . . Nun klappt man den Listeneintrag *Android* auf (Abb. 1-2).

Abb. 1-2
Projekt auswählen



Hier gibt es zahlreiche Assistenten für verschiedene Projektarten, einzelne Bildschirmseiten oder einzelne Programmdateien. Es lohnt sich, sich später hiermit vertraut zu machen, da diese Assistenten einen Teil des

Codes für verschiedene Projektarten oder Programmbestandteile gleich automatisch generieren.

Da wir eine normale Android App implementieren wollen, wählen wir *Android Application Project* aus und drücken *Next*. Auf der folgenden Bildschirmseite (Abb. 1-3) geben wir der App einen Namen, so wie er im Programmmanager des Android-Geräts angezeigt werden soll.

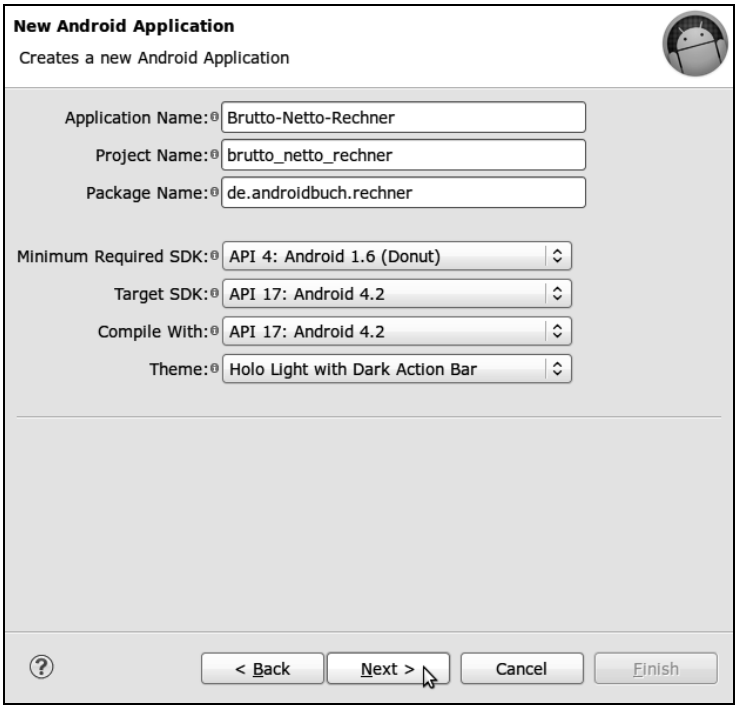


Abb. 1-3
Name und Android-Version

Der »Project Name« ist der Name des Eclipse-Projekts, so wie er im Package Exporer angezeigt wird. Der »Package Name« sollte einzigartig sein, da hierüber die Apps auf dem Android-Gerät unterschieden werden. Üblicherweise verwendet man hier einen registrierten Domänennamen in umgekehrter Reihenfolge, gefolgt vom Basispackage der Anwendung. Da wir die Domäne www.androidbuch.de registriert haben, verwenden wir »de.androidbuch.rechner«.

Die folgenden Einträge beziehen sich auf die Android-Versionen. Zum einen ist da 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 noch einige Anwender mit Android 2.1 gibt, ist dies ein guter Einstiegspunkt. Für unseren Brutto-Netto-Rechner reicht sogar Android 1.6. Benötigt man während der Implementierung zwingend Methoden oder Klassen, die erst in einer

Kleinster gemeinsamer Nenner

späteren Android-Version hinzugekommen sind, kann man die minimale Android-Version nachträglich in der Datei `AndroidManifest.xml` erhöhen.

Die nächsten beiden Einträge beziehen sich auf den Compiler, mit dem der Programmcode erzeugt werden soll. Beide Einträge sollte man auf den gleichen Wert setzen, um Fehler im Android-Manifest zu vermeiden. Das »Theme« beeinflusst das Aussehen der Anwendung. Es stehen verschiedene Themes zur Verfügung die sich unter anderem in Hintergrundfarbe, Aussehen von Oberflächenelementen und dem Aussehen des »Action Bar« unterscheiden.

Die folgende Seite des Assistenten (Abb. 1-4) bietet die Möglichkeit, das Projekt zu konfigurieren. Wir übernehmen die Standardeinstellungen. So wird unter anderem automatisch eine einfache Bildschirmseite generiert. Auf der nächsten Seite (Abb. 1-5) kann ein Icon für verschiedene Bildschirmtypen hochgeladen und angepasst werden. Entscheidend ist in erster Linie die Pixeldichte des Geräts, weniger die Auflösung des Bildschirms in Pixeln.

Eine Seite weiter (Abb. 1-6) kann man definieren, welche Art von Bildschirmseite beim Erzeugen des Projekts automatisch generiert werden soll. Wie wir später sehen werden, besteht die Implementierung einer Bildschirmseite aus mehreren Dateien. Durch die Wahl der Option »BlankActivity« generieren wir eine normale Bildschirmseite, was für den Brutto-Netto-Rechner ausreichend ist. Diese Bildschirmseite wird automatisch aufgerufen, sobald die Anwendung gestartet wird. Im letzten Schritt des Assistenten (Abb. 1-7) wird nun die zu generierende Bildschirmseite näher definiert. Unter »Activity Name« gibt man den Namen einer Java-Klasse an, die die Bildschirmseite implementiert. Entsprechend sollte der Name in der Java-üblichen Camel-Case-Schreibweise eingetragen werden. Unter »Layout Name« gibt man den Namen einer XML-Datei (ohne Endung) ein. Diese XML-Datei enthält später das Layout der Bildschirmseite. Zum Layout gehören z. B. Textfelder, Drop-down-Listen, Schaltflächen und Checkboxen. In Android ist es üblich, die Oberflächenelemente einer Bildschirmseite in XML anzuordnen und die Logik der Seite in einer Java-Klasse zu implementieren. Beides gehört zusammen, daher muss hier der Name der Layout-XML-Datei angegeben werden.

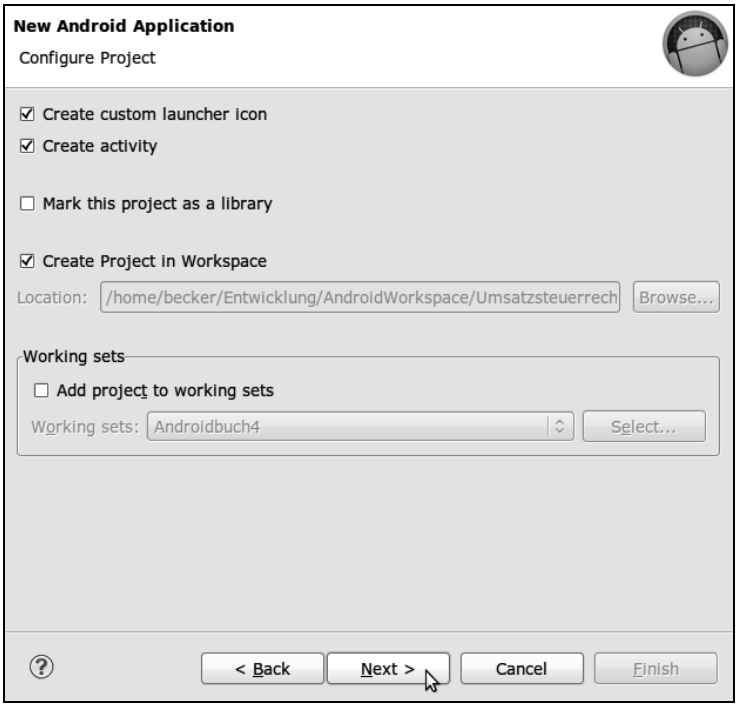


Abb. 1-4
Projektkonfigu ation

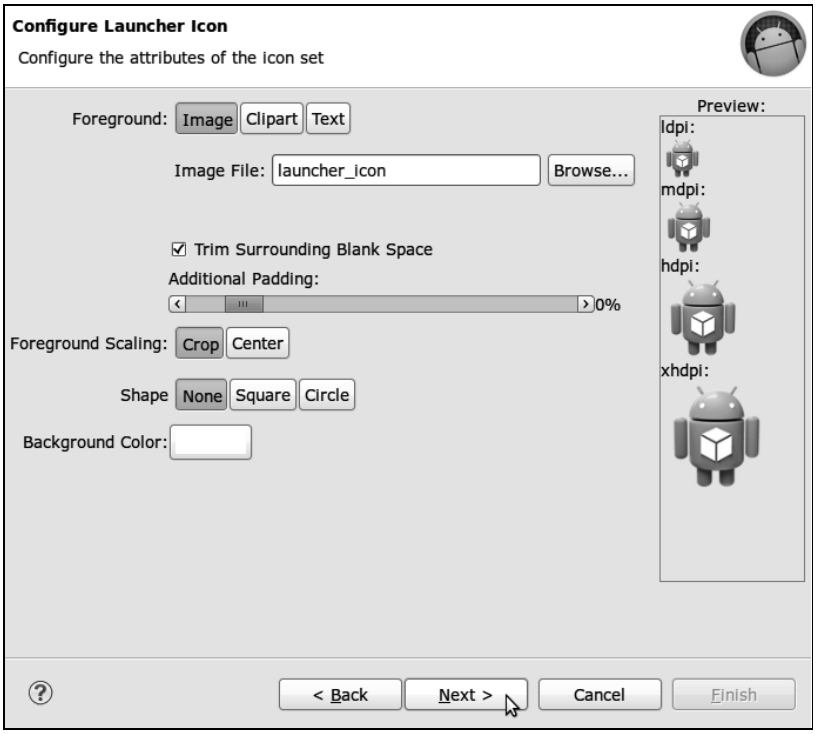


Abb. 1-5
Projekt-Icon

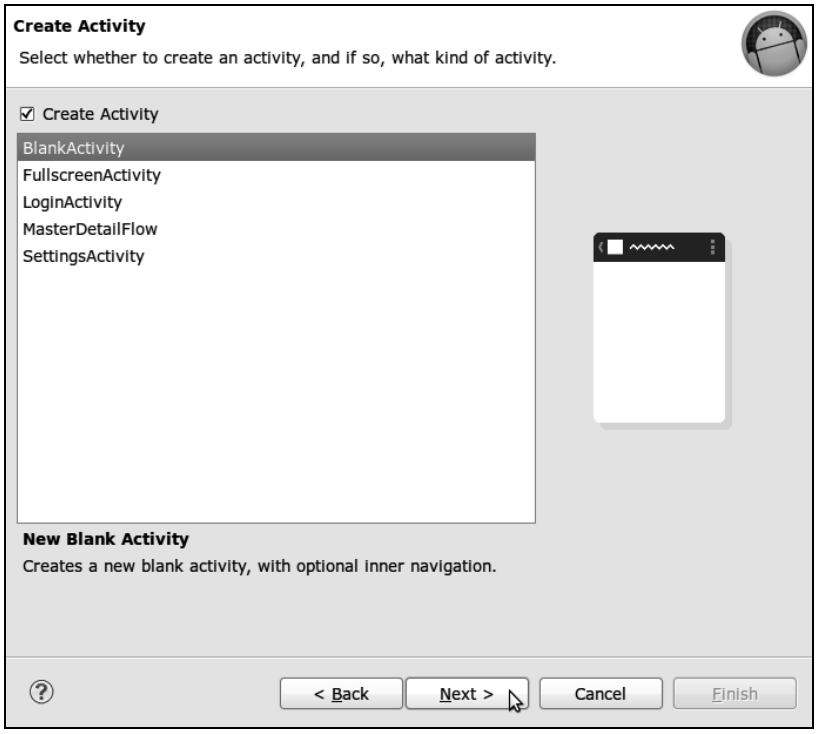


Abb. 1-6
Start-Activity anlegen

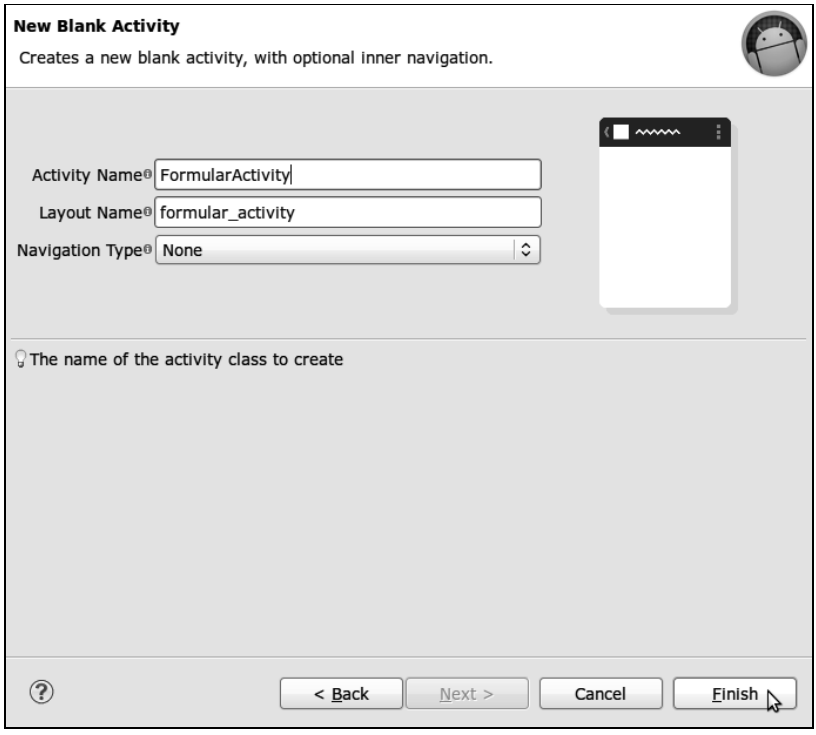


Abb. 1-7
Bildschirmseite defini ren

Nach Fertigstellung legt der Projekt-Wizard die folgende Ordnerstruktur für Android-Projekte an:

- *src*: Java-Quelltexte (u.a. auch unsere Activity `FormularActivity`)
- *gen*: Automatisch generierte Klassen, wie z. B. die `R`-Klasse mit den Indizes der Ressourcen (Abschnitt 5.3)
- *assets*: Zusätzlicher Ordner zur Ablage von Ressourcen. Erlaubt auch Unterordner.
- *bin*: Ergebnisse des Ressourcen-Compilers. Hier darf nichts von Hand geändert werden.
- *libs*: Ordner für zusätzlich benötigte Bibliotheken. In unserem Beispiel wird beispielsweise die Compatibility Library verwendet, die es ermöglicht, innerhalb eines Projekts für die Android-Versionen 1–4 (für Smartphones UND Tablets) zu implementieren.
- *res*: Ressourcen, d.h. alle Nicht-Java-Dateien. Hier werden u.a. die Dateien zur Definition der Oberflächen (Layouts), Bilder oder Textdefinitionen abgelegt.

Im Wurzelverzeichnis befindet sich die zentrale Datei zur Definition von Metadaten der Anwendung: das `AndroidManifest.xml`.

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 definie en

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="menu_settings">Settings</string>
</resources>
```

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.

Text wird ausgelagert.

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 Tabelle 1-1).

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

Tab. 1-1
Feldliste »Eingabe erfassen«

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

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  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:checked="true" />

    <RadioButton android:id="@+id/rb_art_brutto"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/txt_brutto" />
  </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:onClick="onClickBerechnen" />

</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`.

Spinner = Auswahlliste

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

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

```
  <array name="ust_werte">
    <item>19</item>
    <item>16</item>
    <item>7</item>
  </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>
```

Geschafft! Unser Formular zur Erfassung der Eingabewerte ist fertig. Nun muss die Anwendung nur noch im Emulator gestartet werden. Auch dazu bedienen wir uns der vorhandenen Eclipse-Umgebung. Man klickt das Projekt an und drückt die rechte Maustaste. In dem Kontext-Menü wählt man *Run As –> Android Application*. Nach einer Wartezeit sollte die folgende Bildschirmseite erscheinen (Abb. 1-8).

Eine Anwendung starten

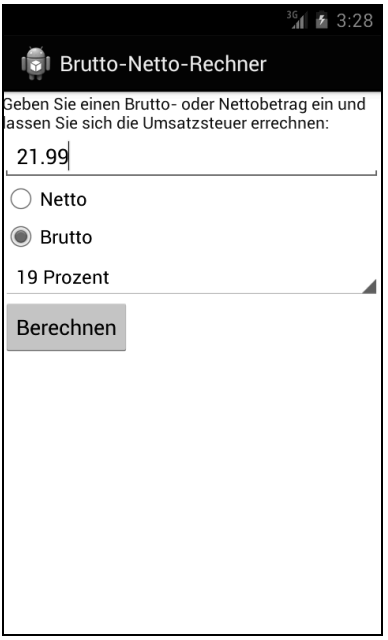


Abb. 1-8
Beispielanwendung im Emulator

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

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

</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"
    package="de.androidbuch.rechner"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="4"
        android:targetSdkVersion="17" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >  (1)
        <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 im Wurzelverzeichnis des 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).

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

Ergebnis berechnen

Als nächsten Schritt lassen wir die beiden Activities miteinander kommunizieren. Konkret soll `FormularActivity` nach Auswahl der Menüoption »Umrechnen« 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 aus `FormularActivity`.

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";

@Override
public void onClickBerechnen(View button) { // (1)
  // Betrag:
  final EditText txtBetrag =
    (EditText) findViewById(R.id.edt_betrag); // (2)
  final float betrag = Float.parseFloat(
      txtBetrag.getText().toString()); // (3)

  // 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)
}
```

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.

Programmablauf

- 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).

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.

Ergebnis berechnen

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 1-9 zeigt die Ergebnisseite im Emulator.

Zugriff auf Views

Abb. 1-9
Die Ergebnisseite des Umsatzsteuerrechners

Abschließend erweitern wir die Anwendung noch um ein Menü. Dazu legen wir eine zusätzliche XML-Datei an, die den Menüeintrag enthalten soll. Auch hierfür gibt es einen Assistenten. In der Menüleiste von

Eclipse klickt man auf *File -> New -> Other*. Unter der Rubrik *Android* wählt man *Android XML File*. Auf der folgenden Seite wählt man unter *Ressource Type* den Eintrag *Menu* und gibt der Datei einen Namen. Wir legen auf diese Weise die Datei *menu.xml* im Ordner */res/menu* an. Es gibt zahlreiche weitere Assistenten, die einem beim Erstellen von Ressourcendateien unterstützen. Es lohnt sich, sich mit diesem vertraut zu machen.

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" >

  <item
    android:id="@+id/opt_beenden"
    android:title="@string/mnu_ende" />

</menu>
```

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. 1-9). 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.