# 21 Android-App-Entwicklung

Android ist ein von Google (weiter-) entwickeltes Betriebssystem, das insbesondere für mobile Endgeräte konzipiert worden ist. Damit einher gehen besondere Anforderungen an das Betriebssystem. Android muss im Hinblick auf den Energieverbrauch extrem ressourcenschonend sein, da die Endgeräte zumeist akkubetrieben sind. Dies ist ein wesentlicher Grund dafür, warum das Betriebssystem Linux nicht direkt als Betriebssystem für mobile Endgeräte verwendet werden kann, sondern lediglich als Ausgangsbasis für die Entwicklung von Android dient. Aber auch für Android entwickelte Anwendungen müssen besonderen Bedingungen genügen.

### Besondere Herausforderungen

1. Da es sehr unterschiedliche mobile Endgeräte gibt, muss die Ausgabe den verschiedenen Bildschirmgrößen angepasst werden können. Dieselbe APP sollte nach Möglichkeit auf Handys, Smartphones, Tablettes, TV-Geräten etc. lauffähig sein.
2. Sehr unterschiedliche Hardwarehersteller nutzen auf ihren Geräten Android als Betriebssystem. Somit kann nicht garantiert werden, dass die jeweilige Hardware auch wirklich mit der neuesten Android Betriebssystemversion kompatibel ist. Vielmehr muss der Softwareentwickler davon ausgehen, dass auf den Endgeräten der Nutzer diverse Android-Versionen vorzufinden sind.

### Lösungsansätze und Programmiertipps

***Android-Versionen***

Eine Android-App sollte auf möglichst vielen Endgeräten eingesetzt werden können. Daher ist es NICHT ratsam eine App ausschließlich für die zur Zeit aktuelle Android-Version zu entwickeln. Da die verschiedenen Android-Versionen in der Regel abwärtskompatibel sind, sollte man sich als Entwickler für eine Android-Version entscheiden, mit der mindestens 70% bis 80% der Endgeräte abgedeckt werden können. Nur so ist eine hinreichend große Zielgruppe erreichbar.

***Display*** ***activity_main.xml***

Auch die sehr unterschiedlichen Bildschirmgrößen der diversen Endgeräte stellen eine besondere Herausforderung dar. Um eine Android-App auf unterschiedlichen Ausgabemedien dynamisch anpassen zu können, wird bei der Android-App-Entwicklung stark auf die XML-Templatetechnik gesetzt. Dadurch wird es beispielsweise möglich das gesamte Layout einer Anwendung über XML-Dokumente zu definieren. Der Entwickler legt dabei nur den prinzipiellen Aufbau einer Bildschirmseite (Activity) einer App fest, ohne sich um die tatsächliche Umsetzung und das unterschiedliche Erscheinungsbild auf den diversen Endgeräten Gedanken machen zu müssen (siehe XML-Template, Kapitel 20).
Außerdem ist mithilfe der XML-Templates eine optimale Trennung zwischen Darstellung (View) und Steuerung (Controll) und somit eine ideale Umsetzung der MVC-Architektur erreichbar (vergleiche Kapitel 12.3 u. 18).

*Kapitel 20 12.3 18*

***Sprachen*** ***strings.xml***

Auch sämtliche Textausgaben einer App (wie etwa einfache Label oder auch Schalterbeschriftungen, etc.) lassen sich mittels XML-Templates in separaten Dateien definieren. Diese Technik ist immer dann besonders wichtig, wenn eine App auch in mehreren Sprachen angeboten werden soll. Denn dann kann die Anwendung extrem einfach, auch im Nachhinein, mehrsprachig angepasst werden. Wenn eine App auf dem internationalen Markt Erfolg haben soll, muss sie mindestens auch die Sprache Englisch unterstützen.

***Java vs. XML-Template*** ***MainActivitiy.java***

Android-Anwendungen werden generell in Java programmiert. Obwohl es möglich ist, fast die komplette Standard-Java-API zu nutzen, sollte man sich, gerade was die GUI-Entwicklung angeht, von der „klassischen“ Programmierung lösen und unbedingt auf die XML-templatebasierte Programmierung umsteigen. Nur so lässt sich ein geräteunabhängiges und MVC-architekturkonformes Programm effizient realisieren.
Die Programmiersprache Java kommt spätestens dann ins Spiel, wenn es um die eigentliche Programmsteuerung, die Ausformulierung der Funktionalität geht (entspricht dem Controller der MVC-Architektur). Was beispielsweise bei einem Schalterdruck oder einer Menüauswahl tatsächlich passieren soll, muss in entsprechenden Java-Methoden ausformuliert werden.

### Dateien eines Android-Projektes

In einem minimalen Android-Projekt[1] mit nur einer einzigen Bildschirmseite (Activity) gibt es mindestens drei Dateien, in denen der Entwickler Änderungen vornehmen muss.

***activity_main.xml*** ***Layout (Display)***

Diese XML-Datei dient der Definition des Layouts.

***string.xml*** ***Values (Sprachen)***

In der **string.xml**-Datei sind alle landessprachenabhängigen Anzeigetexte und Beschriftungen definiert.

***MainActivity.java*** ***Java (Steuerung)***

Diese Java-Datei realisiert die Programmsteuerung.

Ein Android-Projekt umfasst allerdings noch eine Reihe weiterer Dateien, von denen der Entwickler mindestens noch die **AndroidManifest.xml**-Datei (enthält grundlegende Spezifikationen zur App) und die **R.java**-Datei (Schnittstellendatei zwischen View und Controller) kennen sollte. Beide Dateien und deren Bedeutung werden im folgenden Beispiel genauer beschrieben.

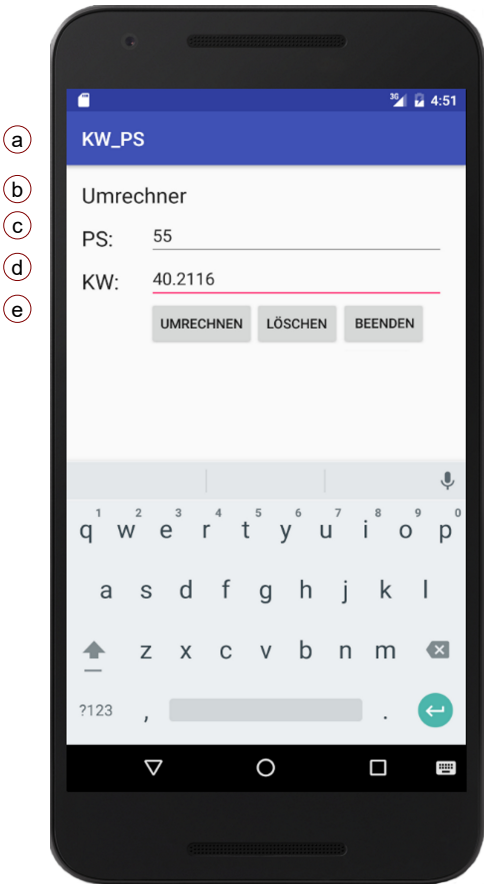
1 Ein minimales Android-Projekt besteht aus nur einer einzigen Bildschirmseite ohne Menüs.

# 21 Android-APP-Entwicklung

## 21.1 Erste Beispiel-App (KW-PS)

*Kapitel 11.4.2*

Im nachfolgenden Beispiel soll der Umrechner zwischen PS (Pferdestärken) und KW (Kilowatt) aus Kapitel 11.4.2 als Android-App ausgeführt werden.




Die App selbst ist sehr einfach gehalten! Sie besitzt nur eine einzige Anzeigeseite, auch Activity genannt. Die Activity selbst soll folgende Komponenten anzeigen:

| | | |
|---|---|---|
| ⓐ ***app_name*** | | |
| Name der App | KW_PS | Label |
| ⓑ ***Beschreibungstext (klein)*** | | |
| Beschreibung | Umrechner | TextView |
| ⓒ ***PS-Eingabezeile*** | | |
| Beschriftung | PS: | TextView |
| Eingabefeld | _____ | EditText |
| ⓓ ***KW-Eingabezeile*** | | |
| Beschriftung | KW: | TextView |
| Eingabefeld | _____ | EditText |
| ⓔ ***Schalter*** | | |
| Beschriftung | UMRECHNEN | Button |
| Beschriftung | LÖSCHEN | Button |
| Beschriftung | BEENDEN | Button |

**Funktionsweise**

Der Schalter BEENDEN soll das gesamte Programm beenden und die App schließen.
Über den Schalter LÖSCHEN kann man beide Eingabefelder leeren.
Sind beide Eingabefelder leer, so lässt sich entweder ein PS- oder ein KW-Wert eingeben und über den UMRECHNEN-Schalter der jeweils andere Wert berechnen und anzeigen.

### 21.1.1 Entwicklungsumgebung


Bevor man nun mit der eigentlichen Programmentwicklung beginnen kann, muss zunächst die Android-Entwicklungsumgebung **Android Studio** installiert werden.
Dabei basiert die von Google kostenlos bereitgestellte IDE auf der **IntelliJ IDEA**. Andere IDE's wie etwa die **Eclipse** werden von Google nicht mehr supportet.

***Installation***

Während der Installation von **Android Studio** besteht die Möglichkeit auch direkt ein virtuelles Device, einen Emulator für Android-Endgeräte zu installieren. Dies ist immer dann sinnvoll, wenn man die Applikationen nicht direkt auf einem Android-Gerät testen kann oder man den damit verbundenen technischen Aufwand scheut.

***Neues Projekt anlegen***

Ist **Android Studio** installiert, kann man ein neues Projekt anlegen. Dafür sind folgende Angaben notwendig:

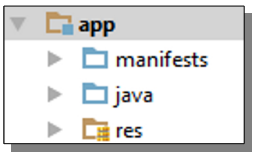
1. Application-Name (Applikationsname)
   a) sollte mit einem Großbuchstaben beginnen.
2. Company Domain (Unternehmensdomain)
   a) kann auch eine Fantasiedomain sein.
3. SDK für unterschiedliche Plattformen
   a) SDK für "Phone and Tablet" auswählen.
   b) Minimale Android-Version bestimmen.
   (so wählen, dass mehr als 70% aller Endgeräte die App auch später einsetzen können!)
4. Start-Avtivity (Grundaufbau) wählen
   a) Da die Beispiel-App sehr einfach ist und sogar ohne Menü auskommt genügt hier die Auswahl "Empty Activity".
   b) Von der Auswahl hängt ab, wie beispielsweise die **AndroidManifest.xml** Datei des Projektes anschließend automatisch generiert wird.

> ***Anmerkung***
>
> Diese Informationen zur Installation und Projekterzeugung in **Android Studio** sollen hier genügen. Genauere und den einzelnen Android-Studio-Versionen exakt angepasste "Schritt für Schritt"-Installationsanleitungen lassen sich im Internet finden.

*Aufgabe 22.21.1*

# 21 Android-APP-Entwicklung

## 21.1.2 Verzeichnisstruktur




Ist das **Android Studio** installiert und das erste Projekt angelegt, so sind in dem Applikationsverzeichnis folgende Unterordner zu finden.

***manifests***

Der Ordner **manifests** enthält alle übergeordneten Spezifikationsdateien, so auch die **AndroidManifest.xml**-Datei der Beispiel-App. Auf die Bedeutung dieser Datei wird zu einem späteren Zeitpunkt noch eingegangen.

***java***

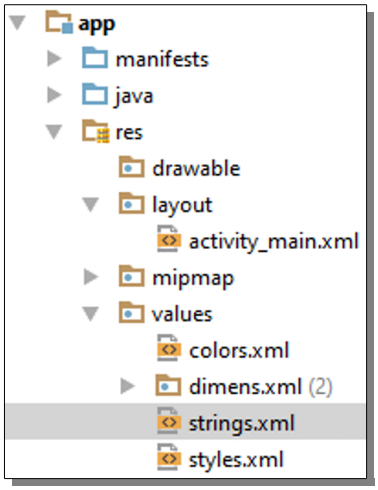
Im Ordner **java** sind alle Java-Quelldateien zu finden. Auch die Datei **MainActivity.java**, die die einzige Avtivity der Beispiel-App steuert, wird in Unterverzeichnissen dieses Ordners zu finden sein.

***res***

Die Abkürzung **res** steht für Ressource und bezeichnet damit ein Verzeichnis, das unterschiedlichste Quelldateien wie Layoutbeschreibungen oder Grafiken (Icon) sowie Textfragmente in Unterverzeichnissen zusammenfasst.

> *Anmerkung*
>
> Bereits nach dem Anlegen des ersten Projektes sind in dem Verzeichnisbaum alle notwendigen Dateien für eine lauffähige Applikation vorhanden. Startet man das Projekt, so erscheint im Emulator bereits (nach einiger Zeit…) die eigene App mit der Ausgabe "Hallo World". Zur Realisierung der Beispiel-App sind somit nur die bestehenden Dateien richtig anzupassen.

## 21.1.3 Die Datei string.xml




Die Datei **strings.xml** stellt landessprachenabhängige Texte bereit. Sie bildet damit die Grundlage für Elemente der Layoutbeschreibung. Daher ist es sinnvoll, gerade mit der Bearbeitung dieser Datei zu beginnen.

**Android Studio** stellt allerdings Hilfsmittel zur Verfügung um auch nachträglich noch String definitionen nachzureichen, so dass es nicht allzu problematisch ist, sollte man mal eine Textdefinition vergessen haben. Da die Datei **string.xml** Werte (**values**) von Textvariablen (Strings) als Quellinformationen (**res**sourcen) bereitstellt, ist die Datei logischerweise im Verzeichnis

```
app\res\values\string.xml
```

zu finden. Die Datei **strings.xml** hat folgenden Inhalt: Alle Texte der Beispiel-App sind darin enthalten.

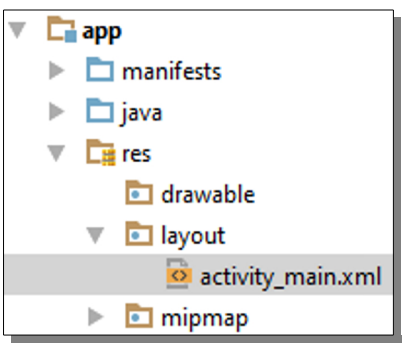
**string.xml**

```
1  <resources>
2      <string name="app_name">KW_PS</string>
3      <string name="textView_ueberschrift">
4        Umrechner</string>
5      <string name="textView_ps">PS:</string>
6      <string name="textView_kw">KW:</string>
7      <string name="button_umrechnen">
8          UMRECHNEN</string>
9      <string name="button_beenden">
10         BEENDEN</string>
11     <string name="button_loeschen">
12         LÖSCHEN</string>
13 </resources>
```

Dabei handelt es sich um eine reine XML-Datei, die es ermöglicht Texte wie **UMRECHNEN** (Zeile 8) über eindeutige Namen wie **button_umrechnen** (Zeile 7) in anderen Dateien anzusprechen.

Möchte man nun das Programm auf andere Landessprachen umstellen, genügt es den eigentlichen Text **UMRECHNEN** durch z. B. **CONVERT** zu ersetzen. Alle anderen Programmdateien sind von dieser Änderung nicht betroffen, da auch der geänderte Text weiterhin über den internen Namen **button_umrechnen** ansprechbar bleibt.

## 21.1.4 Die Datei activity_main.xml




Ist die Datei **string.xml** vorhanden, lässt sich mit deren Hilfe nun das Layout (View) der Activity der Beispielapplikation bestimmen. Auch hierbei handelt es sich um eine XML-Datei, die die Position und das Aussehen einzelner Grafikkomponenten definiert. Die Datei **activity_main.xml** ist zu finden im Verzeichnis:

```
app\res\layout\activity_main.xml
```

Bitte beachten Sie den Zusammenhang zwischen den rot hinterlegten Stellen hier und in der Datei **string.xml**.

**activity_main.xml** — view

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout
3
4  xmlns:android="http://schemas.android.com/apk/res/android"
5  xmlns:tools="http://schemas.android.com/tools"
6  android:layout_width="match_parent"
7  android:layout_height="match_parent"
8  android:paddingBottom="@dimen/activity_vertical_margin"
```

```
9  android:paddingLeft="@dimen/activity_horizontal_margin"
10 android:paddingRight="@dimen/activity_horizontal_margin"
11 android:paddingTop="@dimen/activity_vertical_margin"
12 tools:context="com.kw_ps.kw_ps.MainActivity">
13   <TextView
14     android:layout_width="wrap_content"
15     android:layout_height="wrap_content"
16     android:text="@string/textView_ueberschrift"
17     android:id="@+id/textView_ueberschrift" />
18   <EditText
19     android:layout_width="match_parent"
20     android:layout_height="wrap_content"
21     android:id="@+id/editText_ps"
22     android:layout_below="@+id/textView_ueberschrift"
23     android:layout_toRightOf="@+id/textView_ueberschrift"
24     android:layout_gravity="left|right" />
25   <EditText
26     android:layout_width="match_parent"
27     android:layout_height="wrap_content"
28     android:id="@+id/editText_kw"
29     android:layout_below="@+id/editText_ps"
30     android:layout_toRightOf="@+id/textView_ueberschrift"
31     android:layout_toEndOf="@+id/textView_ueberschrift" />
32   <TextView
33     android:layout_width="wrap_content"
34     android:layout_height="wrap_content"
35     android:text="@string/textView_ps"
36     android:id="@+id/textView_ps"
37     android:layout_above="@+id/editText_kw"
38     android:layout_alignParentLeft="true"
39     android:layout_alignParentStart="true" />
40   <TextView
41     android:layout_width="wrap_content"
42     android:layout_height="wrap_content"
43     android:text="@string/textView_kw"
44     android:id="@+id/textView_kw"
45     android:layout_alignBottom="@+id/editText_kw"
46     android:layout_alignParentLeft="true"
47     android:layout_alignParentStart="true" />
48   <Button
49     android:layout_width="wrap_content"
50     android:layout_height="wrap_content"
51     android:text="@string/button_umrechnen"
52     android:id="@+id/button_umrechnen"
53     android:layout_below="@+id/textView_kw"
54     android:layout_toRightOf="@+id/textView_ueberschrift"
55     android:layout_toEndOf="@+id/textView_ueberschrift"
56     android:onClick="onClickUmrechnen" />
57  <Button
58     android:layout_width="wrap_content"
59     android:layout_height="wrap_content"
60     android:text="@string/button_beenden"
61     android:id="@+id/button_beenden"
62     android:layout_below="@+id/editText_kw"
63     android:layout_toRightOf="@+id/button_loeschen"
64     android:layout_toEndOf="@+id/button_loeschen" />
65     android:onClick="onClickBeenden"
66 <Button
67     android:layout_width="wrap_content"
68     android:layout_height="wrap_content"
69     android:text="@string/button_loeschen"
70     android:id="@+id/button_loeschen"
71     android:layout_below="@+id/editText_kw"
72     android:layout_toRightOf="@+id/button_umrechnen"
73     android:layout_toEndOf="@+id/button_umrechnen"
74     android:onClick="onClickLoeschen" />
75
76 </RelativeLayout>
```

**Android Studio** unterstützt auch hier den Entwickler bei der Erstellung dieser Layoutdatei. Über komfortable Editoren kann man die einzelnen Grafikkomponenten an den gewünschten Stellen positionieren, ohne sich in die zulässige Android-XML-Tagsyntax einlesen zu müssen. Auf diese Weise können beispielsweise alle XML-Tag-Argumente, die mit **android:layout_** beginnen, automatisch generiert werden.

***android:text***

Das XML-Argument **android:text** verweist mithilfe der Angabe **@string** auf die Einträge der **string.xml** Datei. Achten Sie darauf, dass Sie in der **activity_main.xml**-Datei KEINE Texte direkt angeben!

***android:id***

Über dieses Argument werden den Grafikkomponenten eindeutige id-Bezeichnungen zugeordnet, unter denen sie später eindeutig identifiziert und angesprochen werden können. Die Angabe **@+id** sorgt für das automatische Anlegen einer entsprechenden Referenz in der Datei **R.java** (siehe folgenden Abschnitt).

Will man beispielsweise später im eigenen Java-Programm den Wert eines Eingabefeldes (**EditText** Zeile 18) auslesen, so benötigt man dazu dessen **id** (**editText_ps**, Zeile 21). **id**-Werte sind frei, allerdings möglichst "sprechend" und eindeutig zu wählen.

***android:onClick***

Um einen Schalter mit Funktionalität hinterlegen zu können, ist bei dem Attribut **android:onClick** der Name einer Funktion anzugeben, die beim Drücken des Schalters ausgeführt werden soll.

```
android:onClick="onClickUmrechnen"
```

Beispielsweise sorgt die Festlegung in Zeile 56 dafür, dass die Methode **onClickUmrechnen()** ausgeführt wird, sobald man den Schalter UMRECHNEN drückt.

### 21.1.5 Die Datei R.java

Die Datei **R.java**[1] dient als Schnittstelle zwischen den XML-Template-Dateien und dem eigentlichen Java-Programm. Nach jeder Änderung, beispielsweise an einer XML-Layoutdatei, wird die **R.java** Datei automatisch neu erzeugt und aktualisiert. Daher ist es ausgesprochen unsinnig an dieser Datei eigenständig Änderung vornehmen zu wollen, da sie ständig überschrieben würde. Methoden wie **findViewById()** ermöglichen zusammen mit der Datei **R.java** den Zugriff auf die Grafikkomponenten einer Activity mithilfe deren **id**.

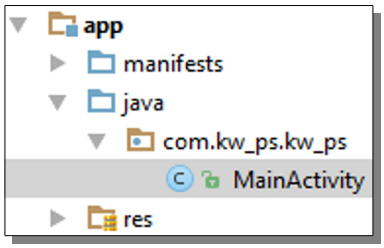
```
e_ps = (EditText)this.findViewById(R.id.editText_ps);
```

Die Variable **e_ps** wird anschließend eine Referenz auf den **EditText** mit dem **id**-Wert **editText_ps** besitzen, aus der mit dem Befehl **getText()** der Eingabetext ausgelesen und in einen String umgewandelt werden kann.

```
String s_ps = e_ps.getText().toString();
```

1 Das R steht als Abkürzung für Ressource.

### 21.1.6 Die Datei MainActivity.java




Nachdem das Layout der Anwendung und somit der View der MVC-Architektur bestimmt worden ist, geht es nun um die Programmsteuerung (Controller). Schon in der Layoutdatei **activity_main.xml** hat man mithilfe des Attributes **android:onClick** Methoden bestimmt, die es nun zu konkretisieren gilt.

`app\java\Paketabhängiger-Name\activity_main.xml`

Die eigentliche Programmlogik wird in Java geschrieben und ist in der Datei **MainActivity.java** zu finden.

controller

**MainActivity.java**

```
1  package com.kw_ps;
2  import android.support.v7.app.AppCompatActivity;
3  import android.os.Bundle; import android.view.View;
4  import android.widget.EditText;
5
6  public class MainActivity extends AppCompatActivity
7  {
8      //Umrechnungskonstanten
9      private final static float PS_TO_KW = 0.73112f;
10     private final static float KW_TO_PS = 1.36f;
11     //Benötigte Textfelder
12     private EditText e_ps; private EditText e_kw;
13
14     protected void onCreate(Bundle savedInstanceState)
15     {
16       super.onCreate(savedInstanceState);
17       setContentView(R.layout.activity_main);
18       //Eingabefelder zuordnen bzw. identifizieren
19       e_ps=(EditText)this.findViewById(R.id.editText_ps);
20       e_kw=(EditText)this.findViewById(R.id.editText_kw);
21     }
22
23     public void onClickUmrechnen(View v)
24     {
25         // Texte einlesen
26         String s_ps = e_ps.getText().toString();
27         String s_kw = e_kw.getText().toString();
28         // Berechnung und Ergebnisanzeige
29         float f_ps; float f_kw;
30         if (!s_ps.equals("") && s_kw.equals(""))
31         {
32             f_ps = Float.valueOf(s_ps).floatValue();
33             f_kw = PS_TO_KW * f_ps;
34             s_kw = String.valueOf(f_kw);
35             e_kw.setText(s_kw);
36         }
37         if (s_ps.equals("") && !s_kw.equals(""))
38         {
39             f_kw = Float.valueOf(s_kw).floatValue();
40             f_ps = KW_TO_PS * f_kw;
41             s_ps = String.valueOf(f_ps);
42             e_ps.setText(s_ps);
43             e_kw.setText(s_kw);
44         }
45     }
46
47     public void onClickLoeschen(View v)
48     {
49         e_ps.setText("");
50         e_kw.setText("");
51     }
52
53     public void onClickBeenden(View v)
54     {
55         this.finish();
56     }
57 }
```

Auch die **MainActivity.java**-Datei wird bei der Projekterzeugung automatisch angelegt und ist vom Entwickler lediglich anzupassen. Im konkreten Beispiel ergänzen die gelb hervorgehobenen Programmteile den automatisch erzeugten Programmrumpf. Die Methoden der Beispiel-**MainActivity.java** sind allesamt ereignisgesteuert. So wird beispielsweise die Methode **onClickUmrechnen()** durchlaufen, wenn der Schalter UMRECHNEN gedrückt wurde.

| Methode | Ereignis / Auswirkung |
|---|---|
| onCreate | Die Activity wird erzeugt / Eingabefelder werden identifiziert. |
| onClick-Umrechnen | Schalter UMRECHNEN wurde gedrückt / die PS-KW-Umrechnung erfolgt. |
| onClick-Loeschen | Schalter LÖSCHEN wurde gedrückt / die Eingabefelder werden geleert. |
| onClick-Beenden | Schalter BEENDEN wurde gedrückt / Activity und Programm werden beendet. |

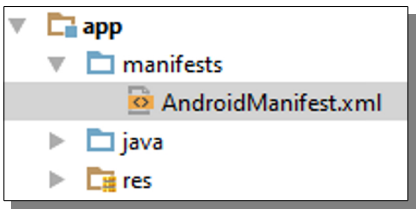
*Anmerkungen*

Die eigentliche Programmlogik (Zeile 25-43) korrespondiert mit dem Beispielprogramm aus Kapitel 11.4.2 und sollte nachvollziehbar sein.

*Kapitel 11.4.2*

Bereits nach diesen Änderungen ist das Programm lauffähig. Weitere Änderungen und Anpassungen in anderen Projektdateien sind nicht notwendig. Trotzdem lohnt es sich noch, einen Blick auf die **AndroidMainManifest.xml** zu werfen, die als zentraler Einstiegspunkt und Workflowsteuerung fungiert.

### 21.1.7 Die Datei AndroidMainManifest.xml




Die XML-Datei **AndroidMainManifest.xml** enthält zentrale Programmparameter. Neben dem Paketnamen, dem Programmicon, den Versions- und Pfadangaben existiert in dieser Datei für jede Activity ein eigener Abschnitt. Da das Beispielprogramm nur eine einzige Activity besitzt, ist in der Datei **AndroidMainManifest.xml** des Beispiels auch nur ein einziger **activity-**

Abschnitt (gelb hervorgehoben) zu finden. Der Name der Activity **MainActivity** muss mit dem Namen der zugehörigen Java-Steuerdatei übereinander passen.

**AndroidMainManifest.xml**

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest
3  xmlns:android="http://schemas.android.com/apk/res/android"
4    package="com.kw_ps">
5    <application
6      android:allowBackup="true"
7      android:icon="@mipmap/ic_launcher"
8      android:label="@string/app_name"
9      android:supportsRtl="true"
10     android:theme="@style/AppTheme">
11     <activity android:name=".MainActivity">
12       <intent-filter>
13         <action
14     android:name="android.intent.action.MAIN" />
15         <category
16     android:name="android.intent.category.LAUNCHER" />
17       </intent-filter>
18    </activity>
19   </application>
20 </manifest>
```

*Anmerkung*

Auch wenn die **AndroidMainManifest.xml**-Datei zunächst automatisch generiert wird, kommt es immer wieder vor, dass Anpassungen an dieser Datei vorzunehmen sind, insbesondere wenn man aufwendige Anwendungen mit mehreren Activitys realisiert.

*Aufgabe 22.21.2*

### 21.1.8 Ausblick

Das Beispiel eines einfachen Android-Projektes aus dem vorherigen Abschnitt zeigt, wie die fünf Dateien

- **AndroidMainManifest.xml**,
- **MainActivity.java**,
- **activity_main.xml**,
- **R.java**,
- **string.xpm**

zusammenspielen und ineinandergreifen. Damit sind Sie in der Lage eigene einfache Android-Anwendungen zu schreiben bzw. automatisch generierten Quellcode Ihren Wünschen entsprechend anzupassen.
Mehr kann und soll das Buch an dieser Stelle nicht leisten. Gerade wenn eine Anwendung mehr als eine Activity besitzt oder Menüstrukturen hinzukommen sollen, steigt die Komplexität der Anwendung schnell an. Trotzdem ist das Erarbeiten weiterer Themen relativ einfach möglich, wenn der grundlegende Aufbau und die prinzipielle Arbeitsweise von Android-Applikationen erst einmal verstanden ist. Daher folgen im nächsten Abschnitt noch einige grundlegende Hintergrundinformationen.

*Aufgabe 22.21.3*

## 21.2 Hintergrundinformationen

Es folgen weiterführende Informationen zu Activitys und deren Kommunikation sowie deren Lebenszyklus.

### 21.2.1 Activity

Die Activity (Bildschirmseite) spielt bei Android-Anwendugen eine zentrale Rolle. Android-Apps bestehen aus nur einer oder aber auch mehreren Bildschirmseiten. Dabei sollte jede Bildschirmseite eine bestimmte Aufgabe oder Aktivität erfüllen. Daraus leitet sich auch der Begriff Activity ab. Zu jeder Activity gehören eine Layoutdatei und untrennbar damit verbunden eine eigene Steuerungsdatei. In dem vorherigen Beispiel gehören etwa die Layoutdatei (View) **activity_main.xml** und die Steuerungsdatei (Controll) **MainActivity.java** untrennbar zusammen und bilden gemeinsam die Definition der Activity. Besitzt ein Android-Projekt mehrere Activities, so behalten die einzelnen Activities auf diese Weise trotzdem ihre Eigenständigkeit. Überspitzt ausgedrückt, ist eine Android-App also nichts andres als eine Ansammlung eigenständiger Activities, die miteinander kommunizieren. Die Kommunikation zwischen unterschiedlichen Activities erfolgt über Intent[1]-Objekte.

### 21.2.2 Intent

Ein Intent ist eine lose Verbindung zwischen aufrufender und aufgerufener Activity. Die Bildschirmseite, die gerade angezeigt wird, ist dabei die aufrufende Activity. Sie muss das Feld (den Bildschirm) räumen, damit nun die aufgerufene Activity angezeigt und somit aktiviert werden kann. Die Verbindung zwischen aufrufender und aufgerufener Activity erfolgt in drei Schritten.

1. Die aufrufende Activity erzeugt ein Intent-Objekt.
2. Sie befüllt das Intent-Objekt mit Informationen.
   ***Informationen im Intent-Objekt sind:***
   a) welche Activity soll aufgerufen werden.
   b) welche Daten sind an die Activity zu übertragen.
3. Anschließend versendet die aufrufende Activity das Intent.

Das System wird das Intent entgegennehmen, die gewünschte Activity starten und die im Intent enthaltenen Daten an die aufgerufene Activity übergeben.

*Anmerkung*

Obwohl in der Beispiel-App des vorigen Kapitels nur eine einzige Activity verwendet wird, finden sich bereits **<intent-filter>**-Tags (Zeile 12-17) in der Datei **AndroidMainManifest.xml**. Über diese XML-Tags wird gesteuert, wie und ob Activities aus anderen Anwendungen mit der selbstgeschriebenen Activity mittels Intent-Objekten in Verbindung treten und diese aufrufen dürfen.

1 Intent, dt. Absicht, Ziel