Inhaltsverzeichnis

1. Vorwort ... 2
2. Das Projekt ... 3
3. Vorarbeiten ... 5
3.1. Android Studio ... 5
3.2. Die Datenbasis ... 13
4. Projekt „KennzeichenDE“ ... 14
4.1. Die Klasse `Kennzeichen.java` ... 15
4.2. Die Klasse `DBHelper.java` ... 17
4.3. Die drei Fenster der Anwendung ... 20
4.3.1. Anzeige der Auswahl ... 21
4.3.1.1. Die layout-Datei `activity_lesen_auswahl.xml` ... 22
4.3.1.2. Die Klasse `LesenAuswahlActivity.java` ... 29
4.3.2. Anzeige aller Datensätze ... 31
4.3.2.1. Layout `activity_alles_lesen.xml` ... 32
4.3.2.2. Activity `AllesLesenActivity.java` ... 33
4.3.3. Die `MainActivity` ... 34
4.3.3.1. Das Layout-File `activity_main.xml` ... 34
4.3.3.2. Die Activity `MainActivity.java` ... 35
4.3.4. Die Datei `kennzeichen.xml` ... 40
4.3.5. Ein schönes Bild abgeben ... 43
4.3.6. Die Manifest-Datei ... 45
4.4. Test der Anwendung ... 46
4.5. Die APK-Datei `app-release.apk` ... 49
4.6. Abschluss Projekt ... 52

1. Vorwort

Nach dem JavaFX Projekt, habe ich mich der App-Entwicklung zugewandt. Einfach, um auch das mal ausprobiert zu haben. Eine konkrete Idee kam mir, als ich wieder einmal ein Autokennzeichen gesehen habe, das ich nicht kannte. Ich wollte also eine App zur Abfrage von deutschen Auto-Kennzeichen erstellen und diese dann auf meinem Smartphone aufspielen.

Damit war der Funktionsumfang der App klar, Eingabe eines Kennzeichens, also ein bis drei Buchstaben, dann einen Button drücken um das gewünschte Kennzeichen in einer internen Datenbasis zu suchen. Ausgegeben werden soll zusätzlich zum Kennzeichen auch die Stadt oder der Landkreis, wovon leitet sich das Kennzeichen ab und in welchem Bundesland liegt die Stadt oder der Kreis. Zusätzlich noch einen Button, über den dann *alle* Kennzeichen hintereinander angezeigt werden sollen.

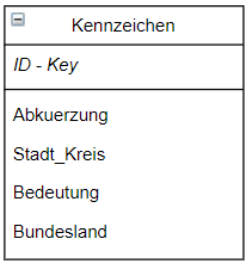
Für die Umsetzung bietet sich das „Android Studio“ an. Das ist eine komplette Workbench für die Android-Entwicklung und zwingt einen in ein enges aber passendes Korsett. Wie ich das meine, wird hoffentlich in den nachfolgenden Kapiteln klar. Die Datenbasis ist wieder eine SQLite Datenbank und als Programmiersprache habe ich mich wieder für Java entschieden, obwohl Kotlin wohl der kommende Stern ist.

Nun noch etwas redaktionelles, in den nachfolgenden Kapiteln streue ich immer wieder Dinge ein, die mir wichtig sind oder zumindest erwähnenswert scheinen. Sie haben oft nur Informationscharakter und bringen das Projekt technisch nicht unbedingt weiter. Diese Notizen sind in Lila gehalten und können überlesen werden, ohne den Projekterfolg zu gefährden. Fachliche Vorgaben, die zur Klärung der Anforderung dienen sind in blau gehalten.

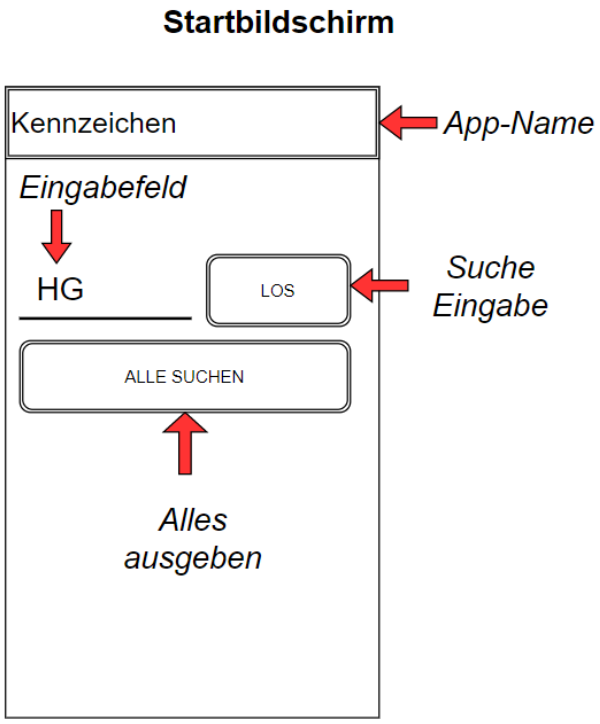
Bei der Namensgebung für Dateien, Variablen usw. habe ich mich bemüht, Begriffe in deutscher Sprache zu verwenden. Dies hauptsächlich um eine Abgrenzung zu dem in englischer Sprache gehaltenen Java-Vokabular zu erreichen. Das gestaltet sich immer dann als schwierig, wenn es um eingedeutschte englische Begriffe wie „upgraden“ oder „Button“ geht. Den Button habe ich tatsächlich „Button“ gelassen, der deutsche „Knopf“ erscheint mir zu weit hergeholt. Sollten mir also englische oder sogar „denglische“ Begriffe durchgerutscht sein, bitte ich um Nachsicht.

2. Das Projekt

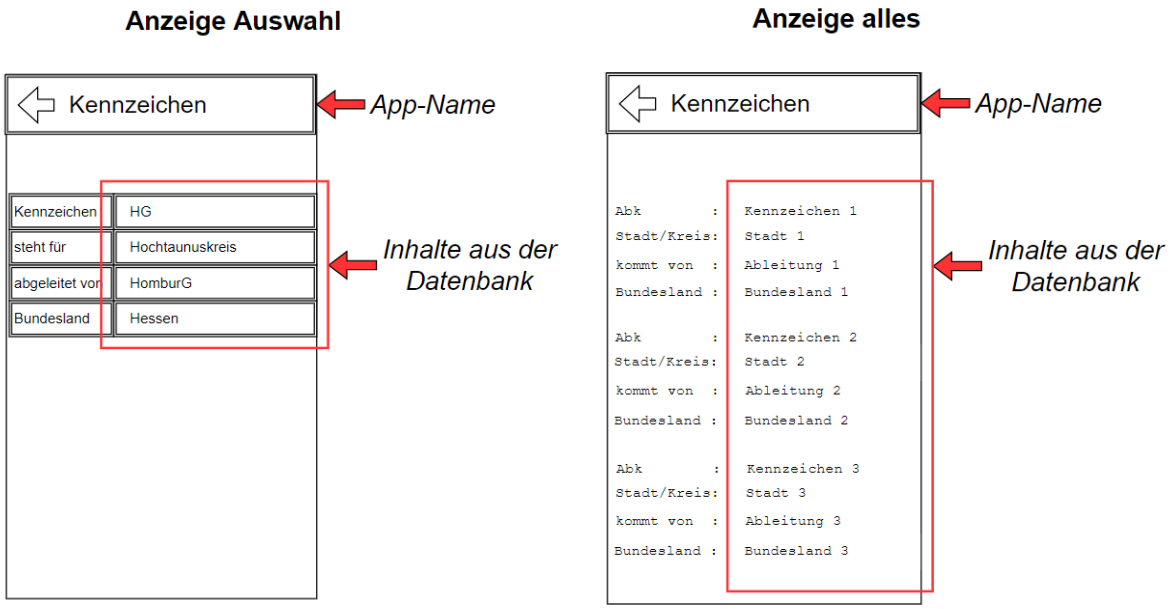
Wie im Vorwort erwähnt, geht es um die Entwicklung einer App, die nach Eingabe einer ein- bis dreistelligen Buchstabenkombination aus einer Datenbank den zugehörigen Datensatz ermittelt und anzeigt. Die Datenbasis ist eine SQLite Datenbank. Es gibt diesmal nur eine Tabelle also keine weiteren Schwierigkeiten. Die Tabelle „Kennzeichen“ hat die nachfolgenden Attribute:



Das Aussehen der App soll sein:



Auf der nächsten Seite sind die beiden Bildschirme schematisch dargestellt, die jeweils bei Klick auf einen der Button erscheinen.



Damit ist das Projekt erklärt, in den nächsten Kapiteln nähern wir uns Schritt für Schritt dem Ziel, nämlich der Umsetzung in der App. Viel Spaß dabei!

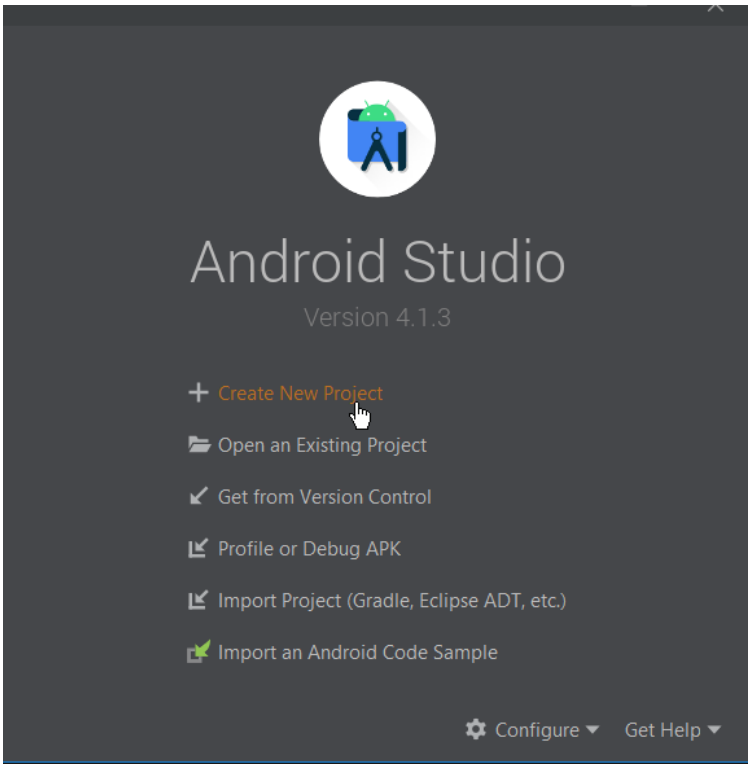
3. Vorarbeiten

Bevor es mit dem Projekt losgeht, sind noch ein paar technische Vorarbeiten notwendig. Den Anfang macht die Installation und der erste Aufruf von Android Studio.

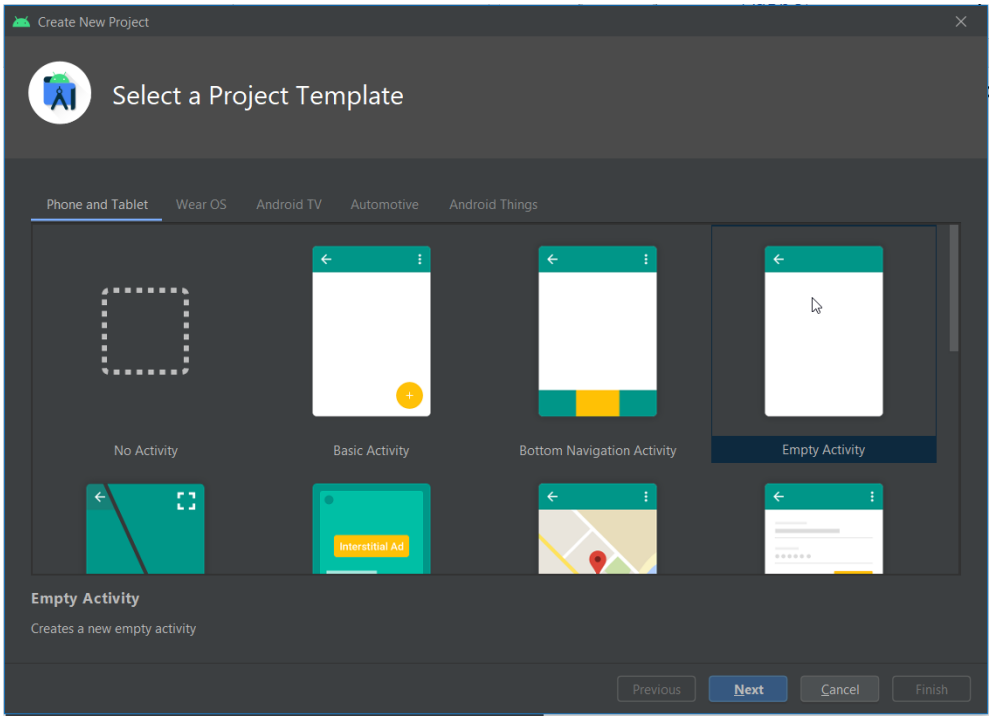
3.1. Android Studio

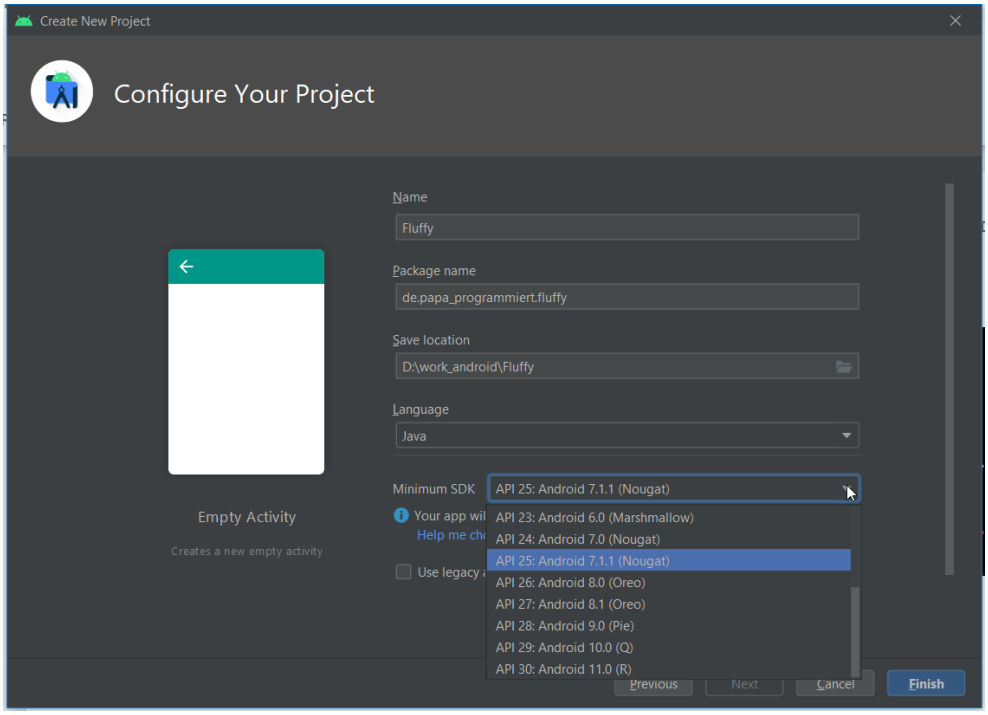
Das Android beinhaltet alles, was es zum Programmieren für eine Android App braucht, das sehen wir später. Als erstes laden wir uns das Paket aus dem Internet. Über die Adresse https://developer.android.com/studio/ gelangt man zur Startseite. Für das Herunterladen ist eine Anmeldung mit einem Google-Account erforderlich. Falls noch nicht vorhanden, kann man das neu erstellen, ich habe mir für diese Zwecke eine eigene Mailadresse besorgt.

Der Installationsprozess ist gewöhnlich, man wird durch das Menu geführt. Die Ersteinrichtung ist ebenfalls keine große Sache, im „Welcome“-Fenster auf Create New Project“ klicken



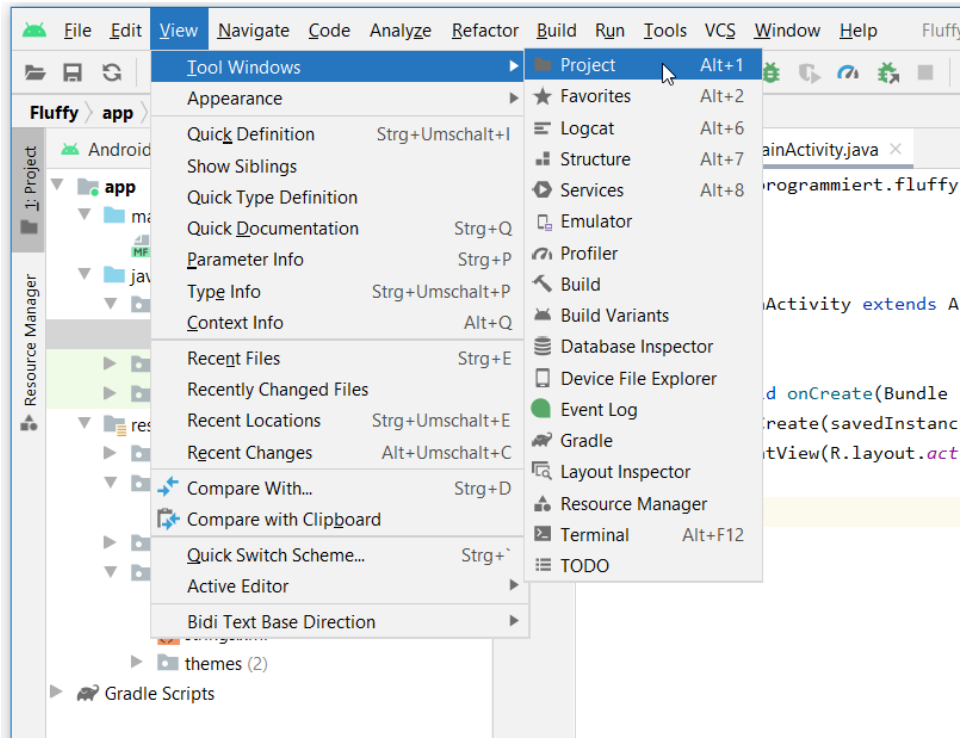
Im nachfolgenden Bildschirm „Empty Activity“ mit Doppelklick wählen:



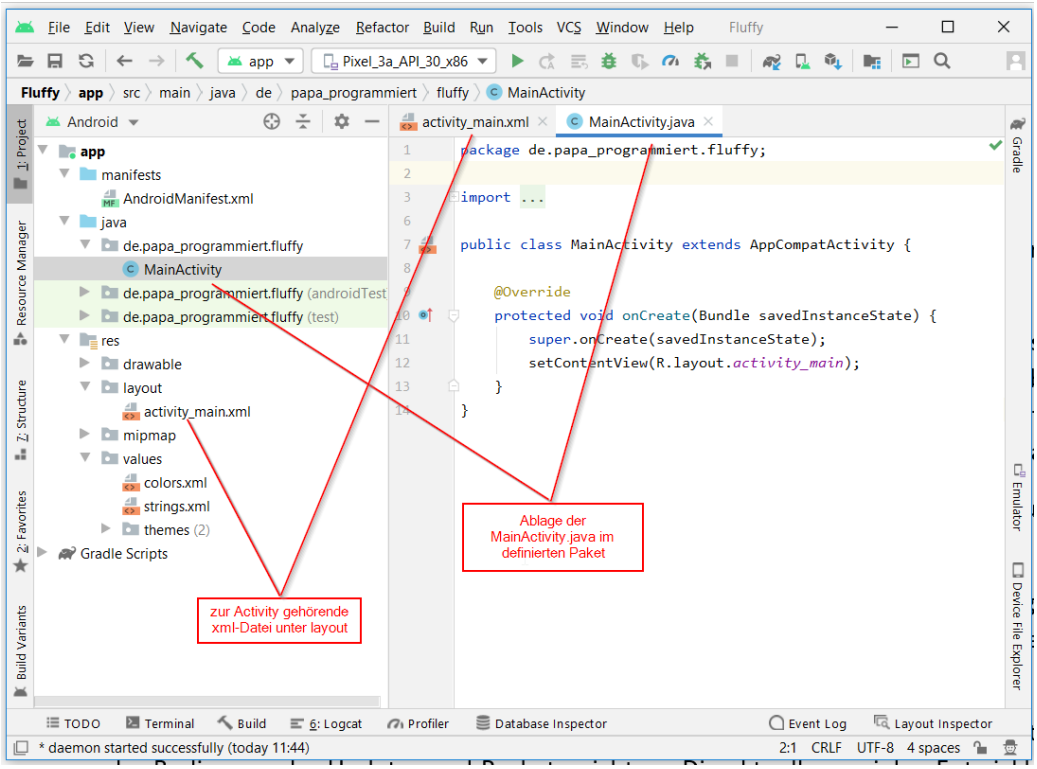


Dem Projekt einen Namen geben und dann das „Minimum SDK“ auswählen. Nehmt am besten das von Eurem Android Smartphone, dann seid Ihr auf der sicheren Seite, was die zur Verfügung stehenden Komponenten betrifft. Wäre ja blöd, wenn Ihr Funktionalitäten nutzen wollt, die Euer Smartphone nicht kann. Mit Klick auf „Finish“ beenden.

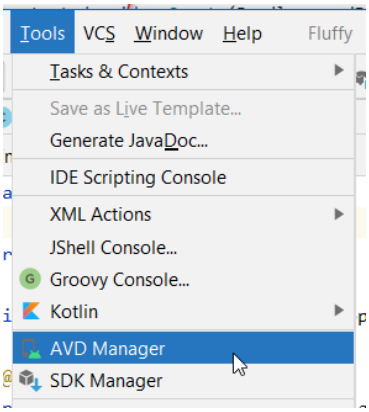
Dann fängt das Studio schon an zu generieren und wir sehen die ersten Elemente. Geöffnet sollten jetzt sein, die Dateien „MainActivity.java“ und „activity_main.xml“. Schauen wir uns das in der Projekt-Ansicht genauer an. Falls das bei Euch nicht so aussieht, die Projektansicht erreicht Ihr über View/Tool Window/Project.



Dann sehen wir hier folgendes:



Damit wir unseren Code testen können, rufen wir zunächst den AVD Manager auf. Der ist zu finden über Tools/AVD Manager:

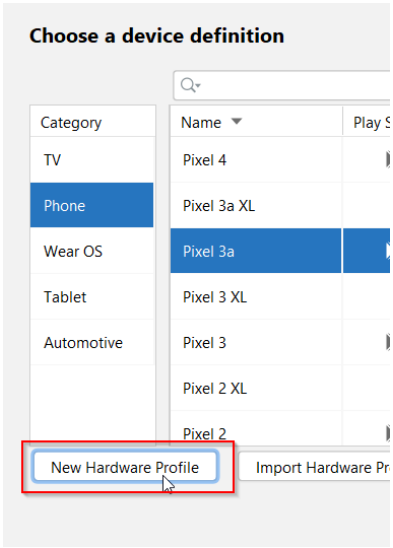


AVD steht für „Android Virtual Device“ und ist ein Tool, das ein Smartphone simuliert, das sehen wir gleich. Da ich nicht weiß was Ihr für Smartphones habt, richte ich jetzt mal das Pixel 3a ein.

Die Schritte sind:

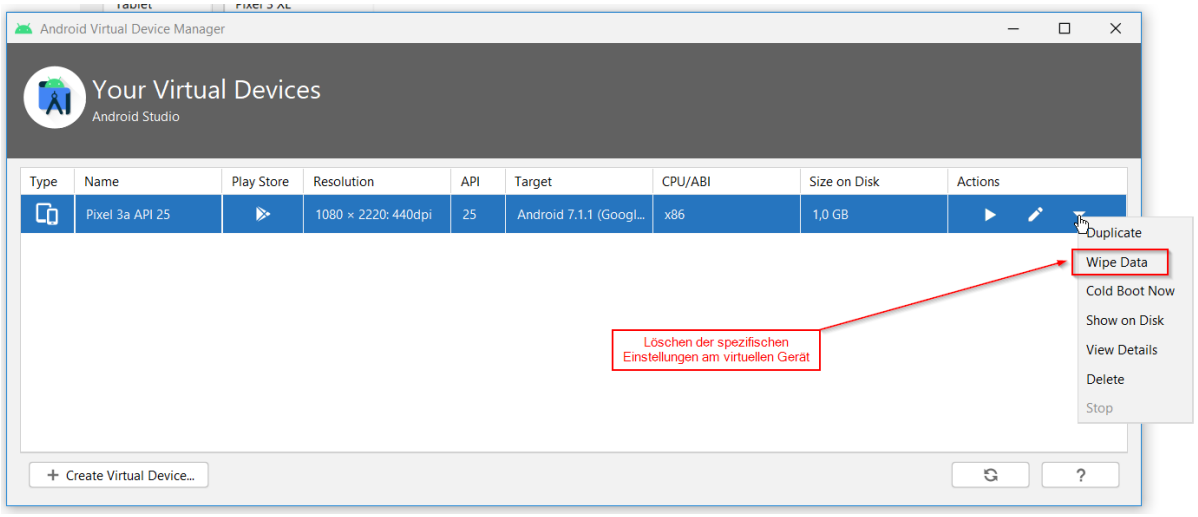
- „Create Virtual Device…“ klicken,
- „Phone” auswählen und „Pixel 3a“ anklicken, dann „Next“
- Im nächsten Fenster die Android-Version wählen, die wird dann heruntergeladen

Ihr könnt aber auch Euer Smartphone simulieren, dazu nach „Create Virtual Device…“ den Button „New Hardware Profile“ klicken, dann öffnet sich ein Fenster und dort gebt Ihr die Spezifikationen von Eurem Gerät ein.



Unten mit „Next“ geht es weiter, dort folgt ein Fenster für weitere Einstellungen, das lassen wir alles so wie vorgeschlagen und schließen mit Klick auf „Finish“ ab.

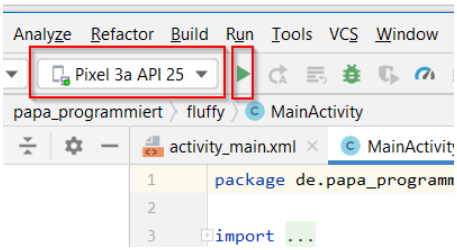
Das nachfolgende Fenster sollte dann etwa so aussehen:



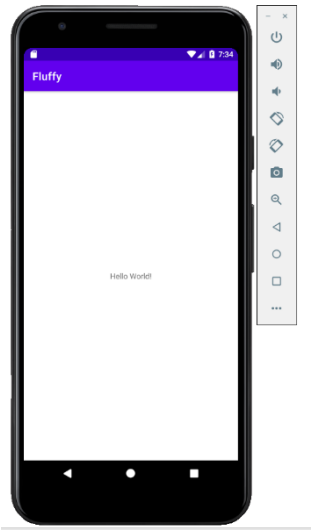
Der Hinweis hier, wenn wir uns mal verrannt haben, können wir mit Klick auf das Dreieck die Option „Wipe Data“ auswählen und alle von uns durch die Applikation gemachten Änderungen im Smartphone werden gelöscht. Kann manchmal ganz praktisch sein.

Damit können wir den AVD Manager verlassen, Fenster einfach mit dem Kreuz oben schließen.

Zurück im Android Studio sollte jetzt oben in der Mitte das Pixel 3a sichtbar sein:



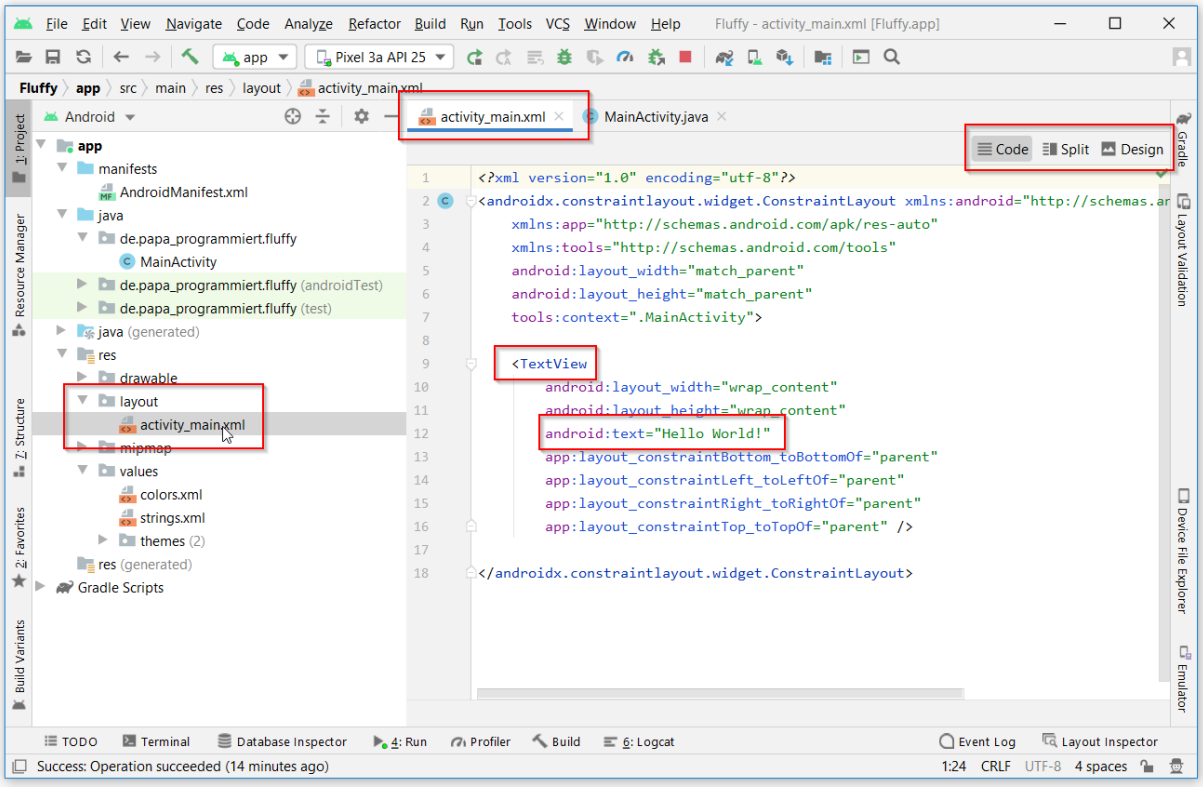
Wenn wir jetzt den grünen Pfeil neben dem Pixel klicken, wird das virtuelle Smartphone gestartet und die App wird direkt installiert. Das ist der Wahnsinn, oder?



Mit Klick auf die Pfeile auf der rechten Seite können wir das virtuelle Smartphone so steuern, als ob wir es in der Hand hätten. Cool, oder?

Okay, aber wo kommt jetzt das „Hello World“ her?

Schauen wir uns die `activity_main.xml`-Datei genauer an:



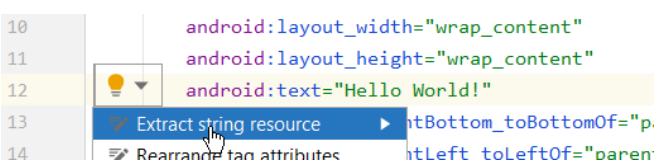
Sie befindet sich im Ordner „layout“. Für die Betrachtung der xml-Dateien gibt es oben rechts 3 verschiedene Ansicht, „Code“, „Split“ und „Design“. Ich habe hier erst einmal „Code“ ausgewählt, über „Design“ können wir per Drag and Drop Komponenten ins Fenster ziehen und sofort sehen, wie das Aussehen ist. „Split“ verbindet beide Sichten. Probiert es aus.

In der Code-Ansicht sehen wir ein Tag vom Typ `TextView`, das kommt immer dann zum Einsatz, wenn Texte angezeigt werden sollen. Das Layout selbst ist vom Typ `ConstraintLayout`. Es gibt viele unterschiedliche Typen von Layouts, jeder hat seinen eigenen Funktionsumfang. Wir bleiben erstmal beim `ConstraintLayout`.

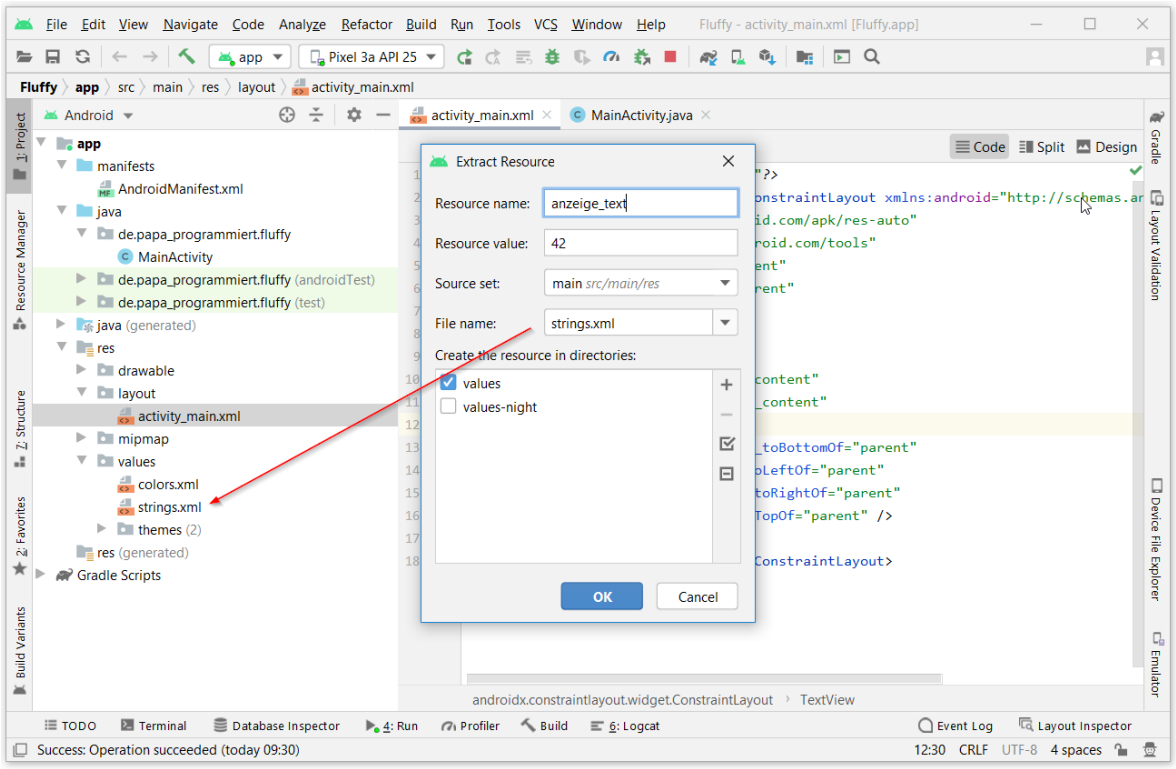
In der Mitte etwa steht die Zeile `android:text="Hello World!"`

Tauschen wir „Hello World!“ Jetzt gegen „42“ aus, wird dort natürlich 42 angezeigt.

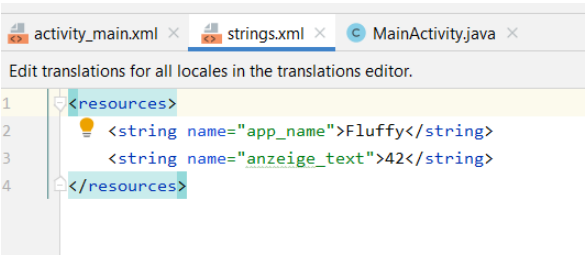
Wir können nun auch schon die erste Verbesserung machen. Ein einfacher Klick irgendwo zwischen den Anführungsstrichen nach `text` lässt auf der linken Seite die Glühbirne erscheinen. Wir wählen den ersten Punkt aus:



Dann geht ein Fenster auf:



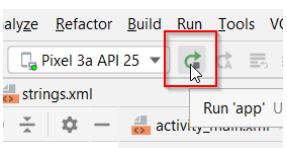
Jetzt können wir dem Feld einen Namen und einen Wert geben. Bei Klick auf „OK“ wird der Wert in die Datei `strings.xml` aufgenommen, die liegt im Ordner „values“. Wir schauen uns den Inhalt mit Doppelklick darauf auch gleich an:



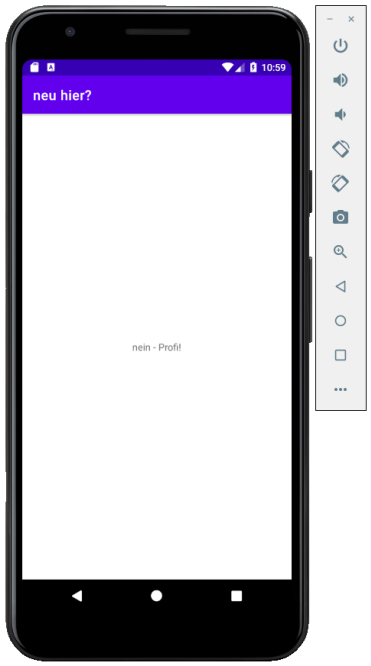
Hier ist also auch der App-Name abgelegt (in unserem Falle „Fluffy“), das ist der String, der im Startbildschirm angezeigt wird. Wir können beides zur Übung mal ändern:

```
<resources>
    <string name="app_name">neu hier?</string>
    <string name="anzeige_text">nein - Profi!</string>
</resources>
```

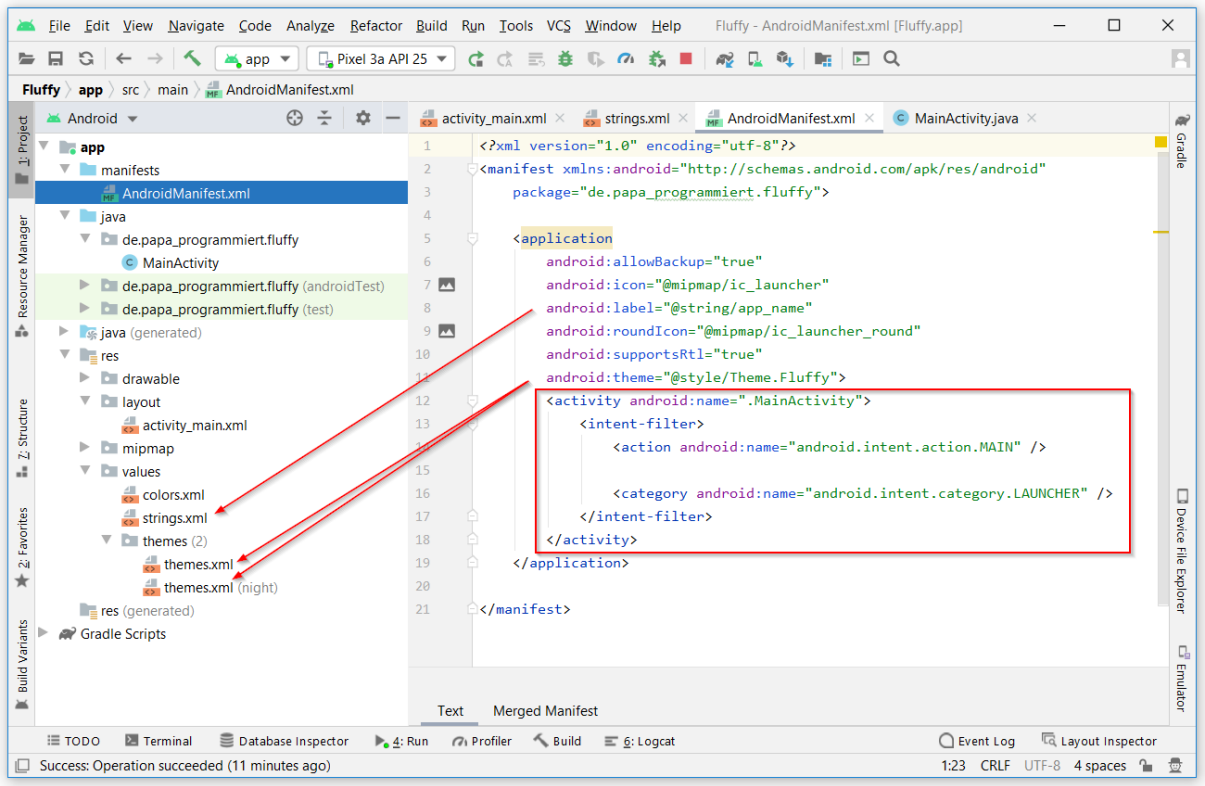
Das Speichern passiert automatisch, man muss also keinen separaten Knopf drücken. Wir müssen also nichts weiter tun, als wieder den Pfeil neben dem Pixel 3a zu drücken. Wenn der AVD Manager noch läuft, sieht das Symbol so aus:



Dann sieht das so aus:



Damit sind wir für die Entwicklung erst einmal gut gewappnet. Eine Datei will ich noch zeigen, das ist die Manifest-Datei. Sie liegt direkt im Hauptverzeichnis unter „manifest“:



Hier laufen alle Fäden zusammen, Alle Bildschirme und das Aussehen werden hier abgelegt. Die Manifest-Datei verändert sich je nach Umfang unserer App. Damit sie korrekt gepflegt werden kann, ist es wichtig, neue Bildschirme generieren zu lassen. Das sehen wir gleich.

Was gibt es sonst noch zu sagen, bevor es losgehen kann?

Wir werden 5 Dateien erstellen. Die Funktionalitäten rund um die Datenbank lagern wir in eine **Helper Klasse** aus. Für die Kommunikation zwischen den Klassen müssen wir eine **Kennzeichen-Objekt-Klasse** anlegen und wir brauchen **3 Bildschirme**, den Startbildschirm, die Anzeige des gesuchten Kennzeichens und die Anzeige aller Kennzeichen. Zum Schluss erstellen wir uns noch ein eigenes Symbol für den Zugriff auf die App.

3.2. Die Datenbasis

Unsere Daten für die Anzeige müssen wir in die Applikation importieren. Als Grundlage dafür dient ein xml-File, in unserem Fall `kennzeichen.xml`. Ich habe dieses File selber erstellt, tatsächlich mit Hilfe eines Cobol-Programms, das ich selber geschrieben habe. In den nächsten Wochen werde ich versuchen auch das Projekt zu beschreiben und online zu stellen. Ihr könnt aber auch ein csv-File nehmen und es in einem Textverarbeitungsprogramm in ein xml-File umwandeln. Alternativ kann auch der Import aus einem csv gemacht werden, das habe ich aber hier nicht beschrieben.

Der Aufbau der Datei ist:

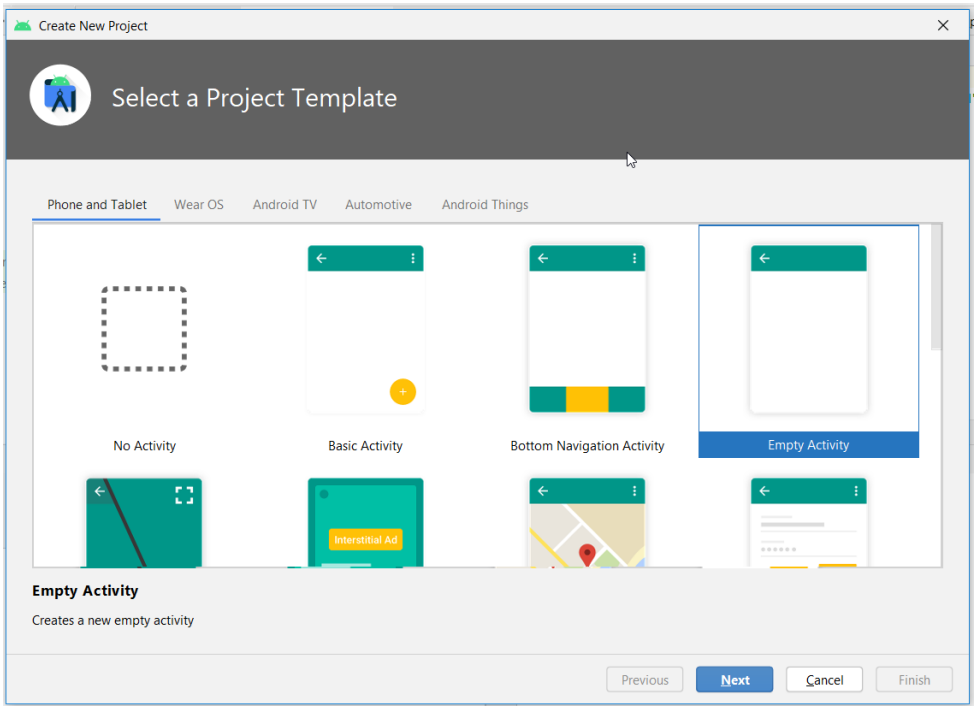
```
<?xml version="1.0" ?>
<kennzeichen>
    <record>
        <Abk>A</Abk>
        <Stadt_Landkreis>Stadt und Landkreis Augsburg</Stadt_Landkreis>
        <abgeleitet_von>Augsburg</abgeleitet_von>
        <Bundesland>Bayern</Bundesland>
    </record>
    <record>
        <Abk>AA</Abk>
        <Stadt_Landkreis>Ostalbkreis</Stadt_Landkreis>
        <abgeleitet_von>AAlen</abgeleitet_von>
        <Bundesland>Baden-Württemberg</Bundesland>
    </record>
    <record>
        <Abk>AB</Abk>
        <Stadt_Landkreis>Stadt und Landkreis Aschaffenburg</Stadt_Landkreis>
        <abgeleitet_von>AschaffenBurg</abgeleitet_von>
        <Bundesland>Bayern</Bundesland>
    </record>
…
</kennzeichen>
```

Damit genug der Vorreden, lasset die Spiele beginnen. Oder so...

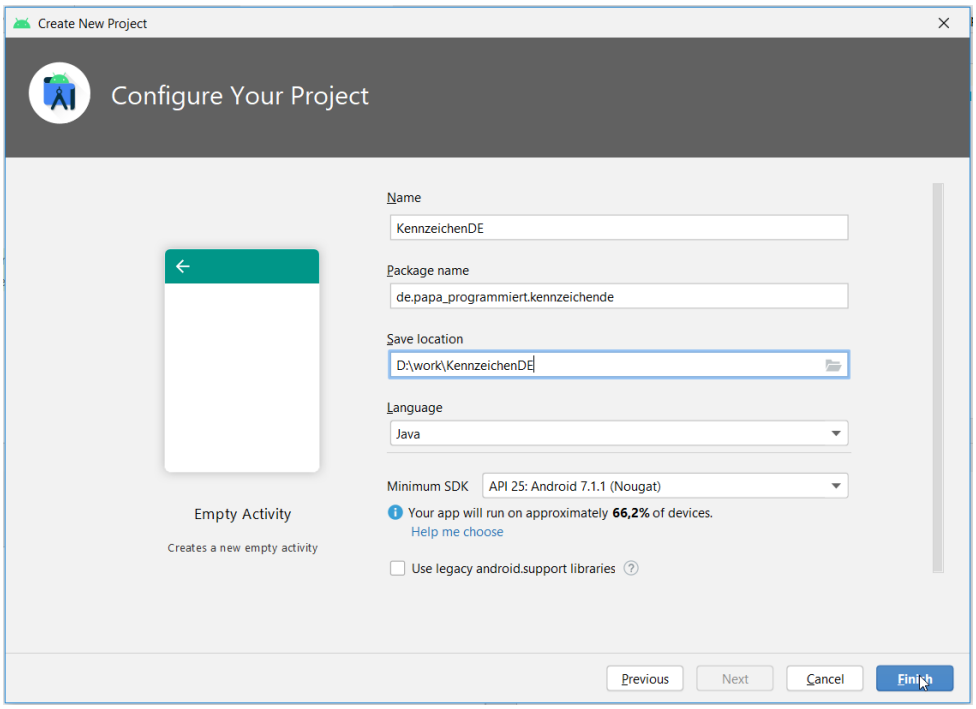
4. Projekt „KennzeichenDE"

Nachfolgende Kapitel beinhalten alle Komponenten zur App, jedes Kapitel sollte einzeln lauffähig sein.

Zunächst starten wir mit einem neuen Projekt. Über File/New/New Project… starten wir den Prozess und geben im nachfolgenden Fenster Empty Activity auswählen und mit Next bestätigen:



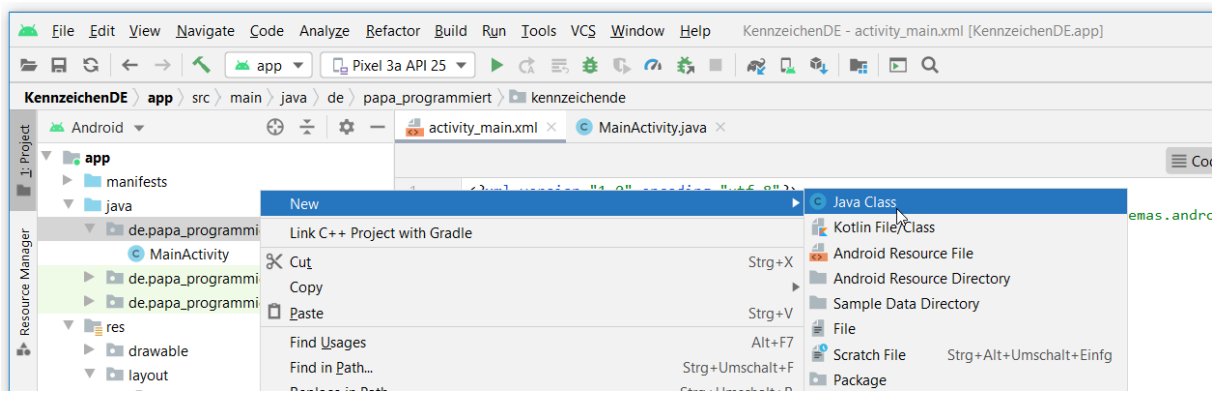
Name, Ablageort und SDK auswählen und mit „Finish" bestätigen:



Zunächst werden wir die Objekt-Klasse `Kennzeichen.java` anlegen.

4.1. Die Klasse `Kennzeichen.java`

Über Rechtsklick auf dem package in dem auch die MainActivity liegt, den Kontext öffnen und über „New/Java Class“ eine neue Klassendatei mit Namen `Kennzeichen` erstellen.



Für die Kennzeichen Klasse nutzen wir die 4 Strings und ein Long als ID, das ist der Datentyp, den die SQLite DB für die Schlüssel der Tabellen braucht. Dann jeweils die Getter und Setter, eine toString-Methode und fertig ist das Gartenhäuschen:

```
1  package de.papa_programmiert.kennzeichende;
2
3  public class Kennzeichen {
4
5      private long id;
6      private String abk;
7      private String text;
8      private String abgel;
9      private String buLand;
10
11     public Kennzeichen(long id, String abk, String text,
12                        String abgel, String buLand) {
13         this.id = id;
14         this.abk = abk;
15         this.text = text;
16         this.abgel = abgel;
17         this.buLand = buLand;
18     }
19
20     public Kennzeichen() {
21         this(0, null, null, null, null);
22     }
23
24     public long getId() {
25         return id;
26     }
27
28     public void setId(long id) {
29         this.id = id;
30     }
31
32     public String getAbk() {
33         return abk;
34     }
35
36     public void setAbk(String abk) {
```

```
37          this.abk = abk;
38      }
39
40      public String getText() {
41          return text;
42      }
43
44      public void setText(String text) {
45          this.text = text;
46      }
47
48      public String getAbgel() {
49          return abgel;
50      }
51
52      public void setAbgel(String abgel) {
53          this.abgel = abgel;
54      }
55
56      public String getBuLand() {
57          return buLand;
58      }
59
60      public void setBuLand(String buLand) {
61          this.buLand = buLand;
62      }
63      @Override
64      public String toString() {
65          String output = id + " + " + abk + " + " + text
66          + " + " + abgel + " + " + buLand;
67          return output;
68      }
69
70  }
```

Ihr könnt Euch das auch generieren lassen, das geht dann entweder über Kontext-Menu (Rechtsklick in der Klasse) und dann „Generate…“ oder Tastenkombination „Alt+Einfg“.

4.2. Die Klasse `DBHelper.java`

Der Zugriff auf eine SQLite Datenbank ist im Android Studio besonders einfach geregelt. Wir erzeugen und eine Klasse, die von `SQLiteOpenHelper` erbt.

Die fertige Klasse sieht so aus:

```
1  package de.papa_programmiert.kennzeichende;
2
3  import android.content.ContentValues;
4  import android.content.Context;
5  import android.database.Cursor;
6  import android.database.sqlite.SQLiteDatabase;
7  import android.database.sqlite.SQLiteOpenHelper;
8  import android.util.Log;
9
10 import java.util.ArrayList;
11 import java.util.List;
12
13 public class DBHelper extends SQLiteOpenHelper {
14     private String eingabe;
15
16     private static final String LOG_TAG = DBHelper.class.getSimpleName();
17     public static final String DB_NAME = "kennzeichen.db";
18     public static final int DB_VERSION = 1;
19     public static final String TABELLE_KENNZEICHEN = "kennzeichen";
20
21     public static final String FELD_ID = "_id";
22     public static final String FELD_ABK = "abk";
23     public static final String FELD_TEXT = "text";
24     public static final String FELD_ABGEL = "abgel";
25     public static final String FELD_BULAND = "buLand";
26
27     public static final String SQL_CREATE =
28             "CREATE TABLE " + TABELLE_KENNZEICHEN +
29                     "(" + FELD_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
30                     FELD_ABK + " TEXT NOT NULL, " +
31                     FELD_TEXT + " TEXT, " +
32                     FELD_ABGEL + " TEXT, " +
33                     FELD_BULAND + " TEXT);";
34
35     public static final String SQL_DROP =
36             "DROP TABLE IF EXISTS " + TABELLE_KENNZEICHEN;
37     public static final String SQL_PRUEFUNG =
38             "SELECT _id FROM " + TABELLE_KENNZEICHEN + " WHERE _id = 1 ;";
39     public static final String SQL_LESE_AUSWAHL =
40             "SELECT _id, abk, text, abgel, buland " +
41                     "FROM kennzeichen WHERE abk = ? ;";
42     public static final String SQL_LESE_ALLES =
43             "SELECT _id, abk, text, abgel, buland FROM kennzeichen;";
```

```
44
45      // Konstruktor
46      public DBHelper(Context context) {
47          super(context, DB_NAME, null, DB_VERSION);
48          Log.d(LOG_TAG, "Datenbank  " + getDatabaseName() + " wurde erzeugt.");
49      }
50
51      @Override
52      public void onCreate(SQLiteDatabase db) {
53          try {
54              Log.d(LOG_TAG, "Anlegen der Tabelle mit: " + SQL_CREATE);
55              db.execSQL(SQL_CREATE);
56          }
57          catch (Exception ex) {
58              Log.e(LOG_TAG, "Fehler beim Anlegen der Tabelle: " + ex.getMessage());
59          }
60      }
61
62      @Override
63      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
64          Log.d(LOG_TAG, "Löschen der Tabelle mit Versionsnummer " + oldVersion );
65          db.execSQL(SQL_DROP);
66          onCreate(db);
67      }
68
69      // Kennzeichen hinzufügen
70      void anlegenKennzeichen(Kennzeichen kennzeichen) {
71          SQLiteDatabase db = this.getWritableDatabase();
72
73          ContentValues values = new ContentValues();
74          values.put(FELD_ABK, kennzeichen.getAbk());
75          values.put(FELD_TEXT, kennzeichen.getText());
76          values.put(FELD_ABGEL, kennzeichen.getAbgel());
77          values.put(FELD_BULAND, kennzeichen.getBuLand());
78
79          // Zeile in die Tabelle schreiben
80          db.insert(TABELLE_KENNZEICHEN, null, values);
81          //Tabelle schliessen
82          db.close();
83      }
84
85      // Alle Kennzeichen lesen
86      public List<Kennzeichen> lesenAlleKennzeichen() {
87          List<Kennzeichen> kennzeichenList = new ArrayList<Kennzeichen>();
88          String selectQuery = SQL_LESE_ALLES;
89
90          SQLiteDatabase db = this.getWritableDatabase();
91          Cursor cursor = db.rawQuery(selectQuery, null);
92
93          // Schleife um alle Datenbankinhalte
94          if (cursor.moveToFirst()) {
```

```
95              do {
96                  //neue Kennzeichen-Instanz
97                  Kennzeichen kennzeichen = new Kennzeichen();
98                  kennzeichen.setId(Long.parseLong(String.valueOf(cursor.getLong(0))));
99                  kennzeichen.setAbk(cursor.getString(1));
100                 kennzeichen.setText(cursor.getString(2));
101                 kennzeichen.setAbgel(cursor.getString(3));
102                 kennzeichen.setBuLand(cursor.getString(4));
103                 // Objekt-Eintrag in Liste erzeugen
104                 kennzeichenList.add(kennzeichen);
105             } while (cursor.moveToNext());
106         }
107         db.close();
108         return kennzeichenList;
109     }
110
111     // Lesen der Auswahl
112     public List<Kennzeichen> lesenEinKennzeichen(String eingabe) {
113         this.eingabe = eingabe;
114         List<Kennzeichen> kennzeichenList = new ArrayList<Kennzeichen>();
115
116         SQLiteDatabase db = this.getWritableDatabase();
117         Cursor cursor = db.rawQuery(SQL_LESE_AUSWAHL, new String[]{eingabe});
118
119         // Lesen der Auswahl
120         if (cursor.moveToFirst()) {
121             do {
122                 //neue Kennzeichen-Instanz
123                 Kennzeichen kennzeichen = new Kennzeichen();
124                 kennzeichen.setId(Long.parseLong(String.valueOf(cursor.getLong(0))));
125                 kennzeichen.setAbk(cursor.getString(1));
126                 kennzeichen.setText(cursor.getString(2));
127                 kennzeichen.setAbgel(cursor.getString(3));
128                 kennzeichen.setBuLand(cursor.getString(4));
129                 // Objekt-Eintrag in Liste erzeugen
130                 kennzeichenList.add(kennzeichen);
131             } while (cursor.moveToNext());
132         }
133         db.close();
134         return kennzeichenList;
135     }
136
137     public Boolean pruefenTabelle() {
138         boolean tabelleVorhanden = false;
139         SQLiteDatabase db = getWritableDatabase();
140         Log.d(LOG_TAG, "Die Tabelle wird erfragt.");
141         Cursor cursor = db.rawQuery(SQL_PRUEFUNG, null);
142
143         while (cursor.moveToNext()) {
144             String foo = cursor.getString(0);
145             if (cursor.getLong(0) == 1) {
146                 tabelleVorhanden = true;
```

```
147             } else {
148                 tabelleVorhanden = false;
149             }
150         }
151         db.close();
152         return tabelleVorhanden;
153     }
154 }
155         return tabelleVorhanden;
156     }
157 }
```

Gehen wir es durch.

Die Imports übergehe ich an dieser Stelle.

Zeile 13 – unsere Klasse erbt von SQLiteOpenHelper

Die Deklarationen sind nicht sonderlich spannend, bis auf Zeile 18. Möchten wir eine neue Version der Datenbank implementieren, erhöhen wir die Version um 1, wodurch die alte Version der Datenbank gelöscht und eine neue angelegt wird.

In Zeilen 36 und 38 habe ich die Konstante „*TABELLE_KENNZEICHEN*“ verwendet, in Zeile 41 und 43 nicht, es geht beides, je nach Geschmack.

Nach dem Konstruktor in Zeile 46 geht es mit den beiden Methoden `onCreate()` und `onUpgrade()` weiter. Sie werden standardmäßig aufgerufen, `onUpgrade` immer dann, wenn sich die Version der Datenbank geändert hat. Probiert das am besten selber mal aus.

In Zeile 70 übergeben wir ein Kennzeichen und lassen es uns in der Tabelle speichern.

Ab Zeile 86 füllen wir die Kennzeichen `List` mit den Inhalten der Tabelle, die wir in einer Schleife um den `cursor` lesen.

Der Einfachheit halber habe ich den Code für das Lesen des Kennzeichens zur Eingabe ab Zeile 112 kopiert. Das geht sicher auch eleganter, tobt Euch aus!

Die Methode `pruefenTabelle()` in Zeile 137 kommt bei Programmstart zum Einsatz. Hier wird geprüft, ob bereits eine Tabelle angelegt wurde, oder ob diese noch aus einem dem csv-file ermittelt werden muss.

Das war die Klasse DBHelper.

4.3. Die drei Fenster der Anwendung

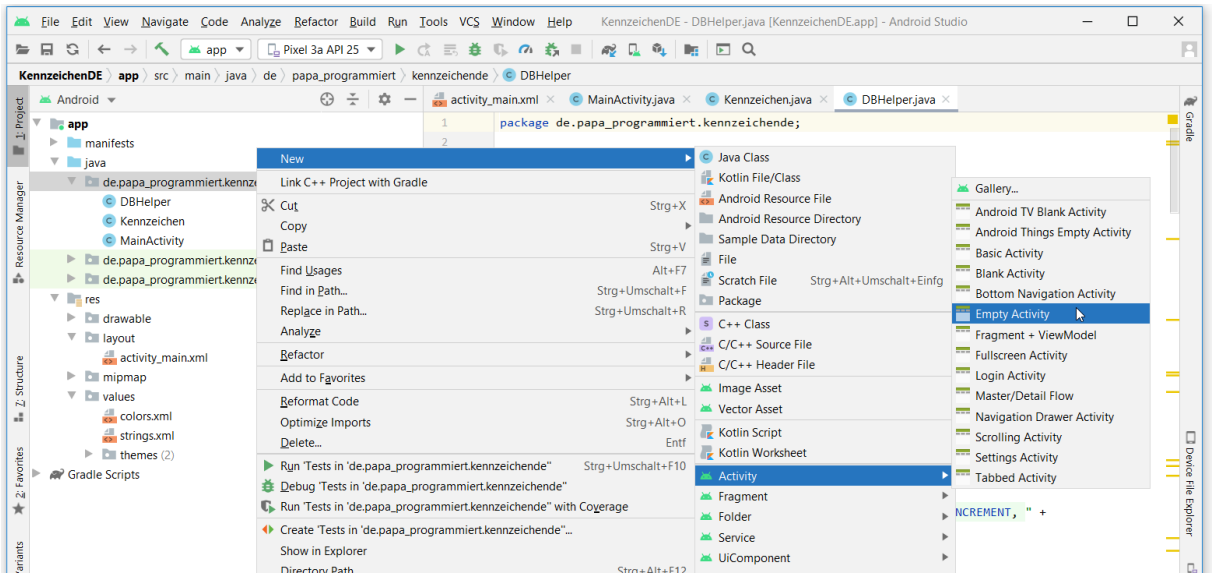
Wie eingangs erwähnt, brauchen wir für unsere Ausgabe 3 Fenster.

Android Studio nimmt uns die Wahl des Ablageortes für die Fenster ab. Wir haben es bei der Generierung des Projektes gesehen, die `MainActivity` wurde im Hauptpaket angelegt, dazu kommt die `layout`-Datei und der Eintrag im `Manifest`.

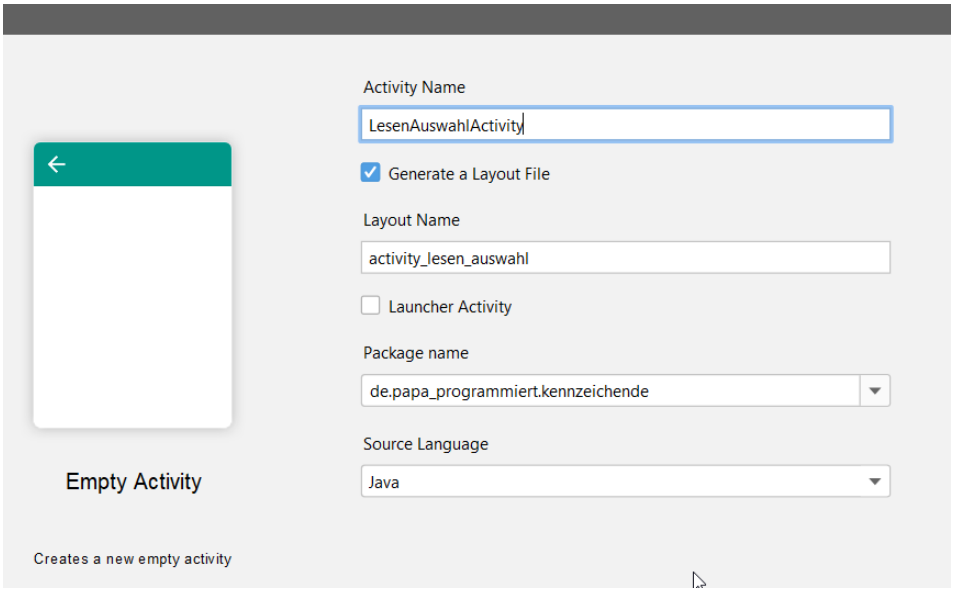
Bei der Anlage einer leeren Activity erzeugt bzw. pflegt Android Studio automatisiert die benötigten Dateien, das sehen wir gleich. Damit das erfolgen kann, müssen wir darauf achten, dass wir das richtige bestellen.

4.3.1. Anzeige der Auswahl

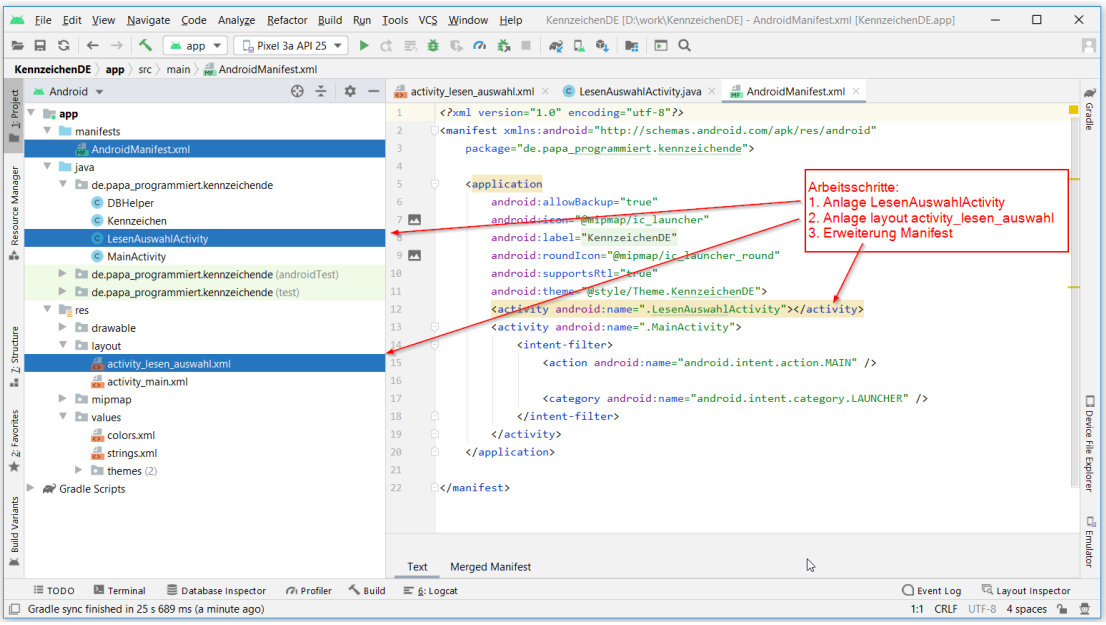
Um das zu demonstrieren, fangen wir mit der Anzeige der Auswahl an. Als Vorbereitung legen wir eine neue „Empty Activity“ an. Das machen wir über Rechtsklick auf das Paket, dann „New/Activity/Empty Activity“:



Der Name soll `LesenAuswahlActivity` heißen;

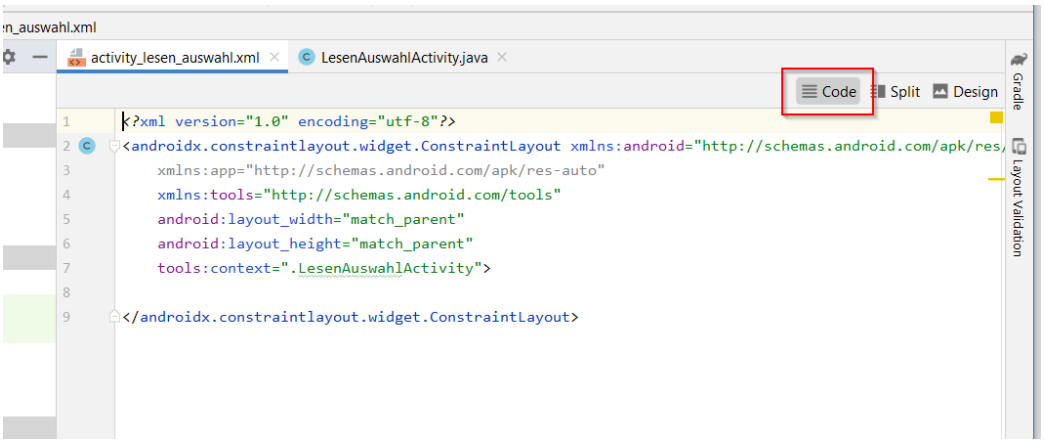


Das Layout-File lassen wir uns gleich mitgenerieren. Wie sieht unser Projekt danach aus?

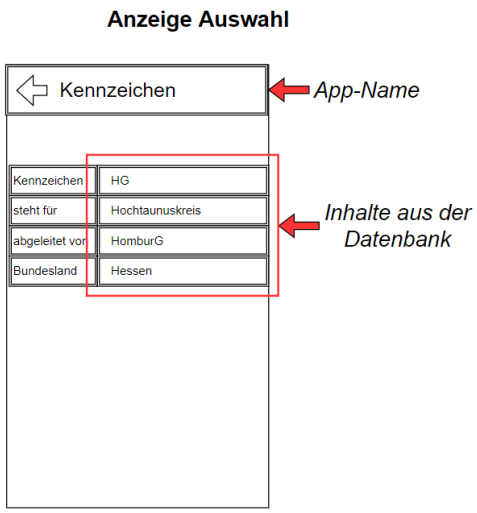


4.3.1.1. Die layout-Datei `activity_lesen_auswahl.xml`

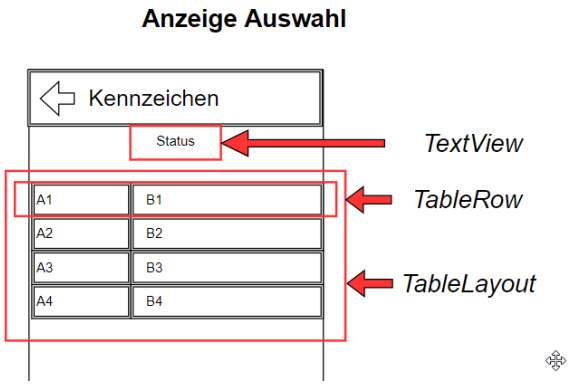
Schauen wir zunächst in die layout-Datei `activity_lesen_auswahl.xml` in der Code-Ansicht an:



Das ist auto-generierter Text, den wir jetzt erweitern wollen. Wir erinnern uns, die Anzeige soll so aussehen:



Bei der Umsetzung bedienen wir uns der Elemente `TextView` (hier habe ich ein Statusfeld vorgesehen, falls es zu der Eingabe keinen Eintrag gibt) und einem `TableLayout`. Die Zeilen der Tabelle heißen `TableRow`, ich habe sie wie in einer Tabellenkalkulation A1 bis B4 benannt:



Der Quellcode sieht so aus:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:nestedScrollingEnabled="true"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/Status"
        android:layout_width="0dp"
        android:layout_height="38dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="8dp"
        android:gravity="center_horizontal"
        android:textSize="15sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TableLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:stretchColumns="2"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
```

```
36          app:layout_constraintTop_toBottomOf="@+id/Status">
37
38          <TableRow
39              android:layout_width="match_parent"
40              android:layout_height="match_parent"
41              android:background="#BABABA">
42
43              <TextView
44                  android:id="@+id/A1"
45                  android:layout_width="wrap_content"
46                  android:layout_height="wrap_content"
47                  android:layout_weight="0.3"
48                  android:padding="10dp"
49                  android:text="Kennzeichen"
50                  android:textSize="15sp"
51                  android:textStyle="bold" />
52
53              <TextView
54                  android:id="@+id/B1"
55                  android:layout_width="wrap_content"
56                  android:layout_height="wrap_content"
57                  android:layout_weight="0.7"
58                  android:padding="10dp"
59                  android:textSize="15sp" />
60
61          </TableRow>
62
63          <TableRow
64              android:layout_width="match_parent"
65              android:layout_height="match_parent"
66              android:background="#E4E4E4">
67
68              <TextView
69                  android:id="@+id/A2"
70                  android:layout_width="wrap_content"
71                  android:layout_height="wrap_content"
72                  android:layout_weight="0.3"
73                  android:padding="10dp"
74                  android:text="steht für"
75                  android:textSize="15sp"
76                  android:textStyle="bold" />
77
78              <TextView
79                  android:id="@+id/B2"
80                  android:layout_width="wrap_content"
81                  android:layout_height="wrap_content"
82                  android:layout_weight="0.7"
83                  android:padding="10dp"
84                  android:textSize="15sp" />
85
86          </TableRow>
```

```
87
88          <TableRow
89              android:layout_width="match_parent"
90              android:layout_height="match_parent"
91              android:background="#BABABA">
92
93              <TextView
94                  android:id="@+id/A3"
95                  android:layout_width="wrap_content"
96                  android:layout_height="wrap_content"
97                  android:layout_weight="0.3"
98                  android:padding="10dp"
99                  android:text="abgeleitet von"
100                 android:textSize="15sp"
101                 android:textStyle="bold" />
102
103             <TextView
104                 android:id="@+id/B3"
105                 android:layout_width="wrap_content"
106                 android:layout_height="wrap_content"
107                 android:layout_weight="0.7"
108                 android:padding="10dp"
109                 android:textSize="15sp" />
110
111         </TableRow>
112
113         <TableRow
114             android:layout_width="match_parent"
115             android:layout_height="match_parent"
116             android:background="#E4E4E4">
117
118             <TextView
119                 android:id="@+id/A4"
120                 android:layout_width="wrap_content"
121                 android:layout_height="wrap_content"
122                 android:layout_weight="0.3"
123                 android:padding="10dp"
124                 android:text="Bundesland"
125                 android:textSize="15sp"
126                 android:textStyle="bold" />
127
128             <TextView
129                 android:id="@+id/B4"
130                 android:layout_width="wrap_content"
131                 android:layout_height="wrap_content"
132                 android:layout_weight="0.7"
133                 android:padding="10dp"
134                 android:textSize="15sp" />
135         </TableRow>
136     </TableLayout>
137 </androidx.constraintlayout.widget.ConstraintLayout>
```

Gehen wir auch das kurz durch.

Zeilen 1 bis 9 kennen wir schon, in Zeile 9 ist der `Context` jetzt der der `MainActivity`.

Zeilen 11 bis 24 bilden den Block für das Status-Feld als `TextView`. Hier wollen wir anzeigen, wenn es zu einer Eingabe kein Kennzeichen gibt.

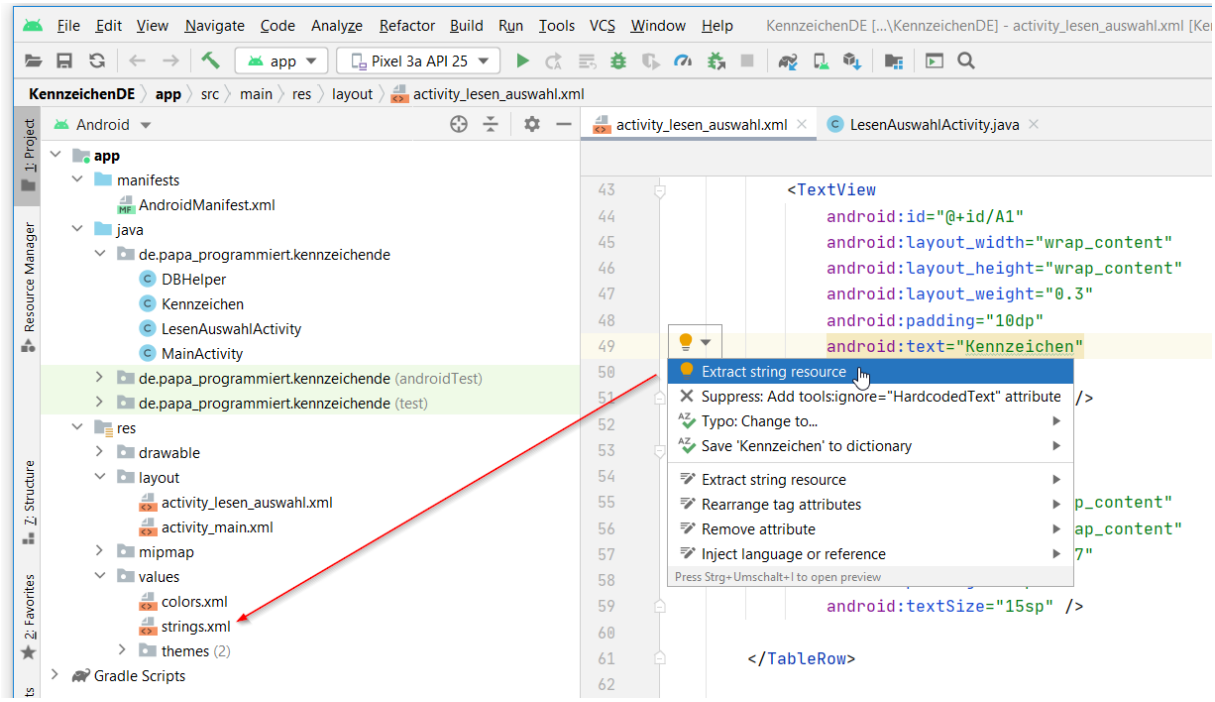
Zeile 26 macht das `TableLayout` auf, geschlossen wird es in Zeile 136.

Die erste Zeile im `TableLayout` fängt Quellcode-Zeile 38 an und geht bis Zeile 61. Die Zelle A1 beginnt Zeile 43 und endet Zeile 51. Die Zelle B1 beginnt Zeile 53 und endet Zeile 59.

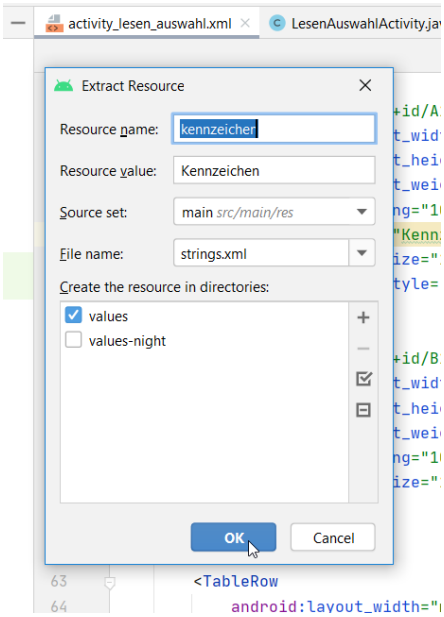
Die anderen Zeilen sind analog.

Wenn wir uns den Quellcode im Android Studio anschauen, sind einige Zeilen gelb unterlegt, was bedeutet, dass es hier einen Hinweis, aber keinen Fehler gibt.

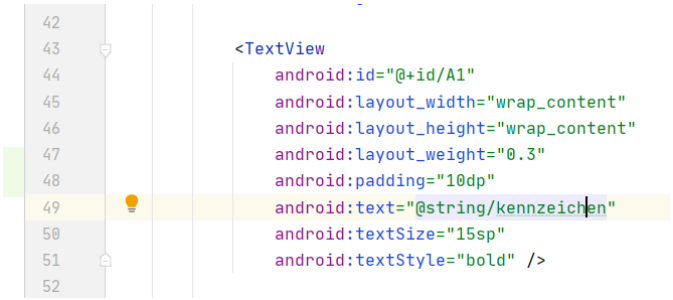
Schauen wir uns das am Beispiel des Literal „`Kennzeichen`“ an. Mit einfachem Klick irgendwo im Wort „Kennzeichen“ geht auf der linken Seite die gelbe „Hinweislampe“ an. Wenn wir auf das Dreieck rechts daneben klicken, wird uns unter anderem der Vorschlag „Extract string resource“ angezeigt. Das bewirkt, dass wir in der value-Datei `strings.xml` einen Wert für den String „Kennzeichen“ hinterlegen, der dann statt des fest codierten Wertes genommen wird.



Klicken wir das an, geht ein neues Fenster auf und wir können die Angaben zum String eingeben, wobei wir darauf achten müssen, dass hier nur Kleinbuchstaben verwendet werden dürfen:

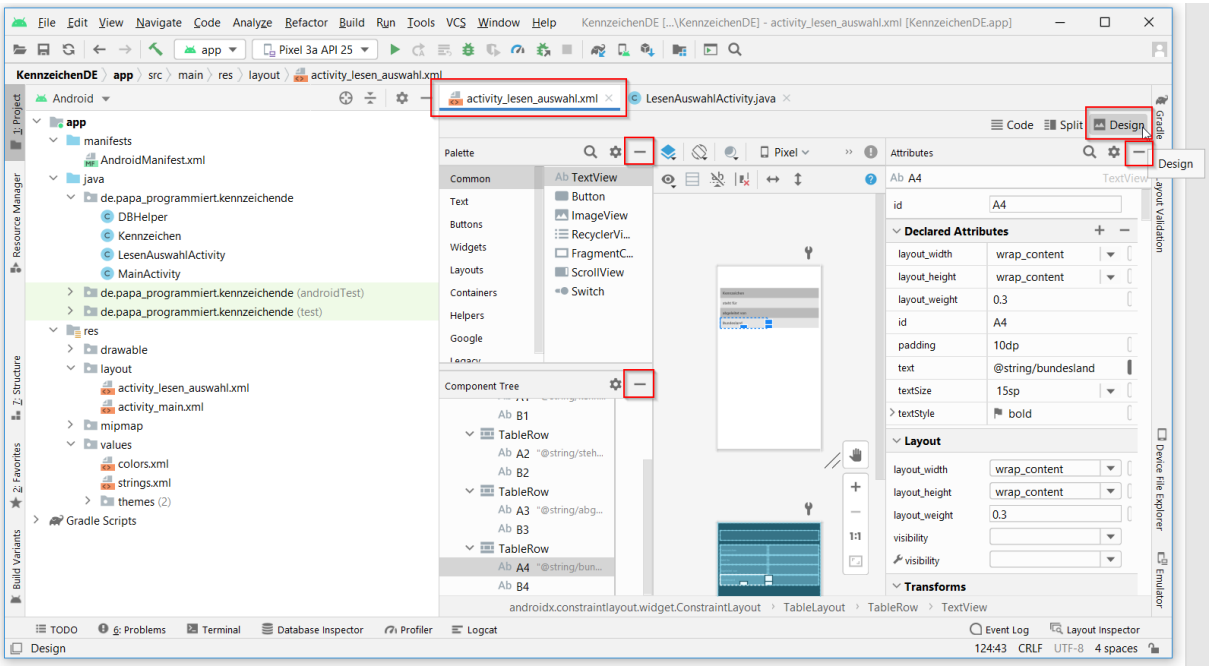


Der Ausschnitt des neuen Quellcode sieht dann so aus:

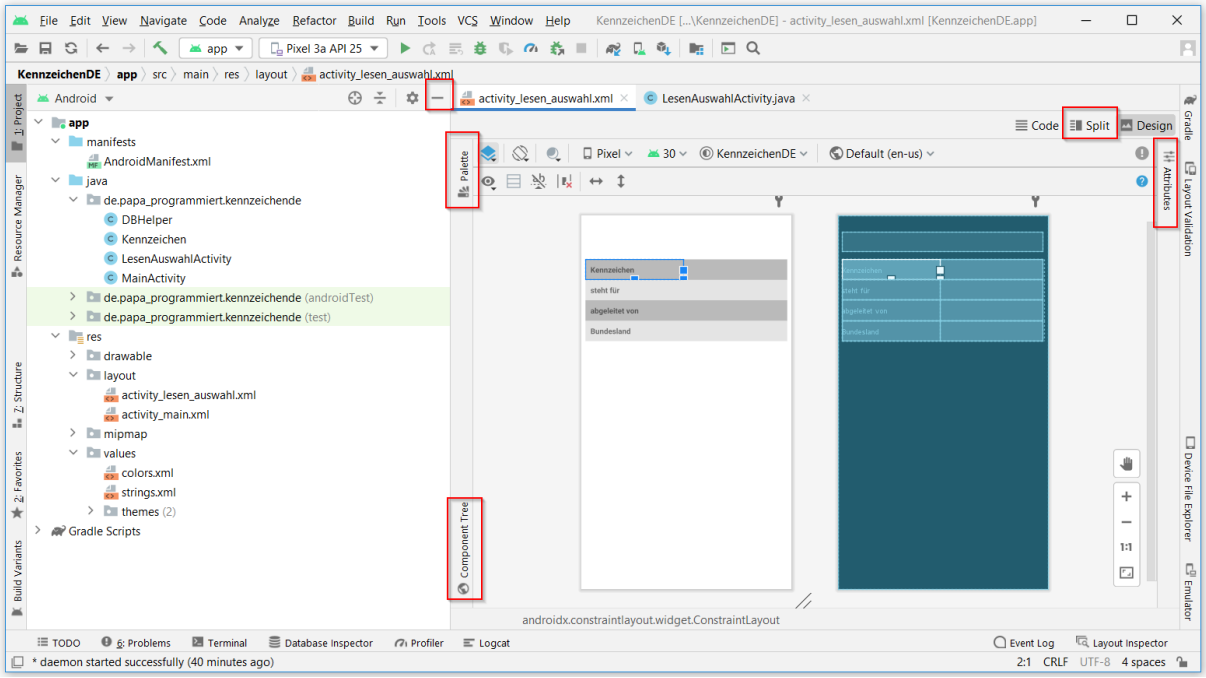


Das machen wir analog mit den anderen Werten der Zellen A2, A3 und A4.

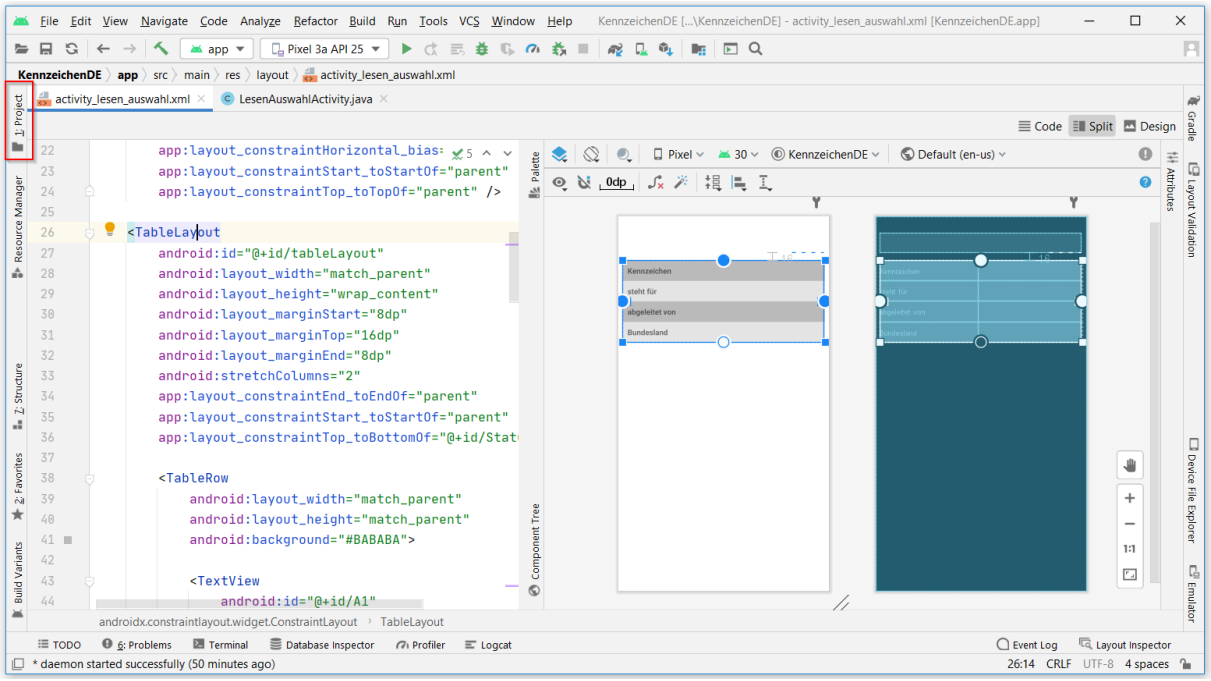
Mit der Erstellung des Aussehens sind wir jetzt erst einmal durch. Wenn Ihr in die Ansicht „Design“ wechselt, werden Euch vermutlich ganz viele Fenster angezeigt:



Die könnt Ihr mit den kleinen Minus-Zeichen erstmal klein machen, sodass nur das mittlere Fenster offen bleibt. Die Fenster sind über den Klick auf den Rand der Ansicht wieder herstellbar:



Damit sehen wir dann schon das Ergebnis unserer Programmierkunst. Wenn man auch das Projekt-Fenster minimiert und in die Ansicht „Split“ wechselt, sieht man beides, den Code im xml-Format und das Aussehen in der „Design“-Ansicht:



Wiederherstellung des Projekt-Fensters wieder über Klick auf das linke Menu-Band.

Damit haben wir das Aussehen definiert, was jetzt noch fehlt ist die Programmierung der Activity.

4.3.1.2. Die Klasse `LesenAuswahlActivity.java`

Der auto-generierte Code sieht so aus:

```
1  package de.papa_programmiert.kennzeichende;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  public class LesenAuswahlActivity extends AppCompatActivity {
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         setContentView(R.layout.activity_lesen_auswahl);
13     }
14 }
```

Alles verständlich, oder? Unsere Klasse erbt von `AppCompatActivity`, die `onCreate()`-Methode wird immer aufgerufen, hier werden sich die die Inhalte aus `activity_lesen_auswahl.xml` geholt.

Unser Code sieht nach der Anpassung so aus:

```
1  package de.papa_programmiert.kennzeichende;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.widget.TextView;
8
9  import java.util.List;
10
11 public class LesenAuswahlActivity extends AppCompatActivity {
12
13     TextView tvB1, tvB2, tvB3, tvB4, status;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_lesen_auswahl);
19
20         Intent intent = getIntent();
21         String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
22         DBHelper db = new DBHelper(this);
23         List<Kennzeichen> kennzeichen = db.lesenEinKennzeichen(message);
24
25         for (Kennzeichen ke : kennzeichen) {
26             tvB1 = findViewById(R.id.B1);
27             tvB1.setText(ke.getAbk());
```

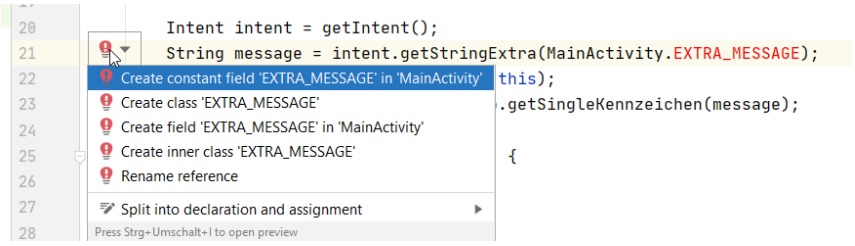
```
28              tvB2 = findViewById(R.id.B2);
29              tvB2.setText(ke.getText());
30              tvB3 = findViewById(R.id.B3);
31              tvB3.setText(ke.getAbgel());
32              tvB4 = findViewById(R.id.B4);
33              tvB4.setText(ke.getBuLand());
34          }
35          if (tvB1 == null) {
36              status = findViewById(R.id.Status);
37              status.setText(R.string.keine_daten);
38          } else {
39              status = findViewById(R.id.Status);
40              status.setText(R.string.alles_klar);
41          }
42          db.close();
43      }
44  }
```

Gehen wir es durch:

Die Imports übergehe ich hier elegant.

Zeile 13 definiert die 4 TextViews der Zellen B1 bis B4, sowie unsere Status-Meldung.

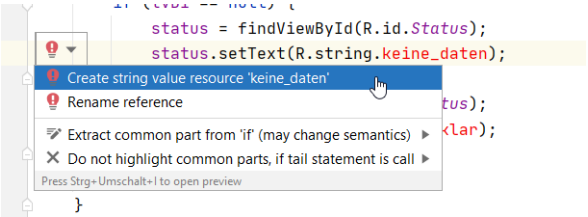
In Zeile 20/21 holen wir uns den Inhalt der Eingabe aus der MainActivity mittels `Intend`. Das schauen wir uns noch genauer an, zumal wir hier noch eine Fehlermeldung haben

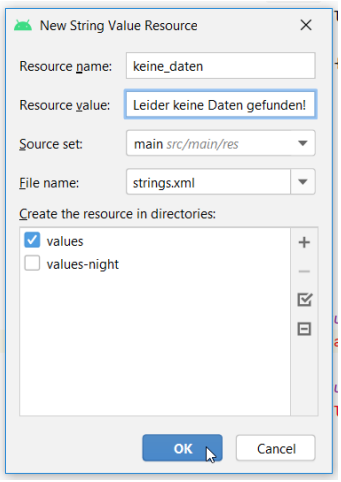


Zeile 23 übergeben wir die Eingabe an die Methode `lesenEinKennzeichen()` des DBHelper und lassen uns das Ergebnis in unsere List schreiben.

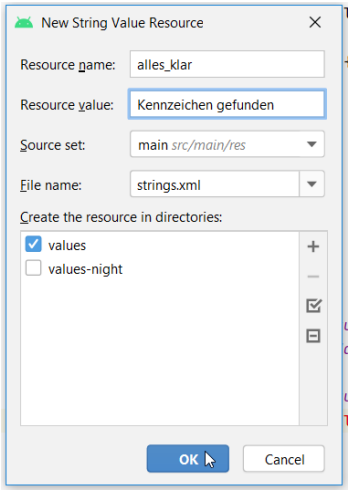
Die gehen wir dann ab Zeile 25 durch und schreiben den Wert auf die `TextView` der Zellen B1 bis B4 im `TableLayout`.

In Zeile 25 fragen wir noch, ob die Abkürzung leer ist, dann haben wir zur Eingabe nichts gefunden. Das sagen wir mit einem String, den wir noch in der string.xml eintragen müssen, aktuell gibt es da auch noch eine Fehlermeldung:





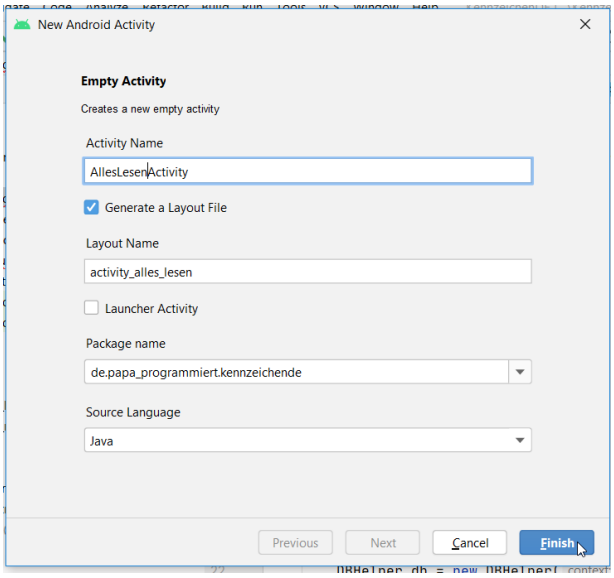
Der `else`-Zweig ist eigentlich überflüssig, ich habe mal die Ausgabe „Kennzeichen gefunden“ definiert. Wenn Ihr mögt, löscht den Teil gerne.



4.3.2. Anzeige aller Datensätze

Bevor wir uns des Fehlers der fehlenden EXTRA_MESSAGE annehmen, der seine Ursache in der MainActivity-Klasse hat, machen wir schnell noch die Anzeige aller Datensätze. Das Zusammenspiel Activity-Klasse und xml kennen wir ja nun schon.

Zunächst legen wir analog dem `LesenAuswahlActivity` eine neue leere Activity an (File/New/ Activity/Empty Activity) und nennen sie `AllesLesenActivity`. Die Layout-Datei lassen wir uns auch gleich generieren.



4.3.2.1. Layout `activity_alles_lesen.xml`

In diesem Fall ist es relativ einfach. Unser Code sieht so aus:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="wrap_content"
4      android:layout_height="match_parent"
5      android:fillViewport="false">
6
7      <LinearLayout
8          android:layout_width="match_parent"
9          android:layout_height="wrap_content"
10         android:orientation="vertical">
11
12         <TextView
13             android:id="@+id/ausgabe"
14             android:layout_width="wrap_content"
15             android:layout_height="wrap_content"
16             android:textSize="15sp"
17             android:typeface="monospace" />
18
19     </LinearLayout>
20 </ScrollView>
```

Gehen wir es wieder durch. In der Datenbank werden am Schluss etwa 700 Kennzeichen landen. Da diese nicht alle auf eine Seite passen, brauchen wir einen Bildschirm, den man blättern kann. Dazu dient `ScrollView`.

Wir haben ab Zeile 12 nur ein `TextView`-Feld definiert, das lassen wir gleich in der `AllesLesenActivity` bestücken.

4.3.2.2. Activity `AllesLesenActivity.java`

Auch diese Geschichte ist schnell erzählt. Der Quellcode sieht so aus:

```
1  package de.papa_programmiert.kennzeichende;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6  import android.widget.TextView;
7
8  import org.w3c.dom.Element;
9  import org.w3c.dom.Node;
10 import org.w3c.dom.NodeList;
11
12 import java.util.List;
13
14 public class AllesLesenActivity extends AppCompatActivity {
15
16     TextView tv1;
17
18     @Override
19     protected void onCreate(Bundle savedInstanceState) {
20         super.onCreate(savedInstanceState);
21         setContentView(R.layout.activity_alles_lesen);
22
23         DBHelper db = new DBHelper(this);
24         List<Kennzeichen> kennzeichen = db.lesenAlleKennzeichen();
25
26         for (Kennzeichen ke : kennzeichen) {
27             tv1= findViewById(R.id.ausgabe);
28             tv1.setText(tv1.getText()+"\n"+"Kennzeichen : " + ke.getAbk()+"\n");
29             tv1.setText(tv1.getText()+"Stadt/Kreis : " + ke.getText()+"\n");
30             tv1.setText(tv1.getText()+"kommt von   : " + ke.getAbgel()+"\n");
31             tv1.setText(tv1.getText()+"Bundesland  : " + ke.getBuLand()+"\n");
32         }
33         db.close();
34     }
35 }
```

Nichts Neues, oder? Über die Methode `lesenAlleKennzeichen()` im DBHelper holen wir uns alle Kennzeichen in die `List` und iterieren ab Zeile 26 darüber.

4.3.3. Die `MainActivity`

So, jetzt wird es nochmal spannend. Wir setzen die Teile jetzt zusammen und dazu müssen wir in die `MainActivity`. Da wir das bei Projektstart schon haben generieren lassen, ist das hier nicht mehr notwendig. Wir gehen daher direkt zum Quellcode.

4.3.3.1. Das Layout-File `activity_main.xml`

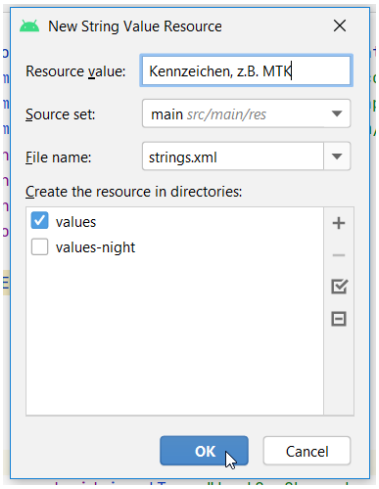
Der Code dafür ist wieder einfach gehalten:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <androidx.constraintlayout.widget.ConstraintLayout
3      xmlns:android="http://schemas.android.com/apk/res/android"
4      xmlns:app="http://schemas.android.com/apk/res-auto"
5      xmlns:tools="http://schemas.android.com/tools"
6      android:orientation="vertical"
7      android:layout_width="match_parent"
8      android:layout_height="match_parent"
9      tools:context=".MainActivity">
10
11     <EditText
12         android:id="@+id/eingabe"
13         android:layout_width="0dp"
14         android:layout_height="64dp"
15         android:layout_marginStart="16dp"
16         android:layout_marginTop="32dp"
17         android:ems="10"
18         android:hint="@string/hinweis_text"
19         android:inputType="textCapCharacters"
20         app:layout_constraintStart_toStartOf="parent"
21         app:layout_constraintTop_toTopOf="parent" />
22
23     <Button
24         android:id="@+id/allesLesen"
25         android:layout_width="0dp"
26         android:layout_height="64dp"
27         android:layout_marginStart="16dp"
28         android:layout_marginTop="32dp"
29         android:layout_marginEnd="16dp"
30         android:onClick="allesLesen"
31         android:text="@string/button_alle"
32         app:layout_constraintEnd_toEndOf="parent"
33         app:layout_constraintHorizontal_bias="0.0"
34         app:layout_constraintStart_toStartOf="parent"
35         app:layout_constraintTop_toBottomOf="@+id/eingabe" />
36
37     <Button
38         android:id="@+id/lesenAuswahl"
39         android:layout_width="0dp"
40         android:layout_height="64dp"
41         android:layout_marginStart="16dp"
42         android:layout_marginTop="32dp"
```

```
43          android:layout_marginEnd="16dp"
44          android:onClick="lesenAuswahl"
45          android:text="@string/button_lesen_auswahl"
46          app:layout_constraintEnd_toEndOf="parent"
47          app:layout_constraintStart_toEndOf="@+id/eingabe"
48          app:layout_constraintTop_toTopOf="parent" />
49
50  </androidx.constraintlayout.widget.ConstraintLayout>
```

Wenn wir den Code in das xml einfügen, gibt es wieder einige Fehlermeldungen.

Die erste ist im `EditText` Zeile 18. Hier müssen wir den String in der `string.xml` aufnehmen. Es handelt sich dabei um einen Hinweistext, den man bei einem Eingabefeld anzeigen lassen kann. In meinem Fall will ich anzeigen lassen „Kennzeichen, z.B. MTK“. Wie das geht, wissen wir ja schon, Feld markieren, Fehlerlampe auswählen und dann „Create string value resource…“:



Die beiden anderen Strings sind die Aufschriften auf den beiden Button, für `button_lesen_auswahl` tragen wir „Los“ ein, für `button_alle` „Alle suchen“.

Die Fehlermeldungen für `android:onClick` übergehe ich hier, die lösen sich gleich auf.

Besonderes Augenmerk noch auf die Zeile 19 `android:inputType="textCapCharacters"`, der Eintrag bewirkt, dass beim Einblenden der Tastatur diese auf Groß-Schrift gestellt wird. Das verhindert allerdings nicht, dass der Anwender auf Klein-Schrift umstellen kann. Sollte er das tun, wird der Eintrag nicht gefunden, da „M“ nicht identisch mit „m“ ist.

4.3.3.2. Die Activity `MainActivity.java`

Kommen wir nun zur `MainActivity` selbst. Der Quellcode lautet:

```
1  package de.papa_programmiert.kennzeichende;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.content.Intent;
6  import android.os.Bundle;
7  import android.util.Log;
```

```
 8 import android.view.View;
 9 import android.widget.EditText;
10
11 import org.w3c.dom.Document;
12 import org.w3c.dom.Element;
13 import org.w3c.dom.Node;
14 import org.w3c.dom.NodeList;
15
16 import java.io.InputStream;
17
18 import javax.xml.parsers.DocumentBuilder;
19 import javax.xml.parsers.DocumentBuilderFactory;
20
21 public class MainActivity extends AppCompatActivity {
22
23     public static final String EXTRA_MESSAGE =
24             "de.papa_programmiert.kennzeichende.MESSAGE";
25
26     @Override
27     protected void onCreate(Bundle savedInstanceState) {
28         super.onCreate(savedInstanceState);
29         setContentView(R.layout.activity_main);
30         tabellePruefen();
31     }
32
33     private void tabellePruefen() {
34         DBHelper db = new DBHelper(this);
35         boolean tabelleVorhanden = db.pruefenTabelle();
36         if (!tabelleVorhanden){
37             try {
38                 InputStream is = getAssets().open("kennzeichen.xml");
39                 DocumentBuilderFactory dbFactory =
40                         DocumentBuilderFactory.newInstance();
41                 DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
42                 Document doc = dBuilder.parse(is);
43                 Element element=doc.getDocumentElement();
44                 element.normalize();
45                 NodeList nList = doc.getElementsByTagName("record");
46
47                 for (int i=0; i<nList.getLength(); i++) {
48                     Node node = nList.item(i);
49                     if (node.getNodeType() == Node.ELEMENT_NODE) {
50                         Element element2 = (Element) node;
51                         String xml_abk = getValue("Abk", element2);
52                         String xml_text = getValue("Stadt_Landkreis", element2);
53                         String xml_abgel = getValue("abgeleitet_von", element2);
54                         String xml_buland = getValue("Bundesland", element2);
55                         Kennzeichen kennzeichen = new Kennzeichen(
56                                 0, xml_abk, xml_text, xml_abgel, xml_buland);
57                         db.anlegenKennzeichen(kennzeichen);
58                     }
```

```
                }
            } catch (Exception e) {e.printStackTrace();}
        } else {
            Log.d("Hinweis ", "Tabelle ist da.");
        }
        db.close();
    }

    private static String getValue(String tag, Element element) {
        NodeList nodeList =
                element.getElementsByTagName(tag).item(0).getChildNodes();
        Node node = nodeList.item(0);
        return node.getNodeValue();
    }

    // Aufruf, wenn Anwender auf "Los" klickt
    public void lesenAuswahl(View view) {
        Intent intent = new Intent(this, LesenAuswahlActivity.class);
        EditText editText = findViewById(R.id.eingabe);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }

    // Aufruf, wenn Anwender auf "alles Suchen" klickt
    public void allesLesen(View view) {
        Intent intent = new Intent(this, AllesLesenActivity.class);
        startActivity(intent);
    }
}
}
        SQLiteDatabase db = this.getWritableDatabase();
        Cursor cursor = db.rawQuery(selectQuery, null);

        // Schleife um alle Datenbankinhalte
        if (cursor.moveToFirst()) {
            do {
                //neue Kennzeichen-Instanz
                Kennzeichen kennzeichen = new Kennzeichen();
                kennzeichen.setId(Long.parseLong(String.valueOf(cursor.getLong(0))));
                kennzeichen.setAbk(cursor.getString(1));
                kennzeichen.setText(cursor.getString(2));
                kennzeichen.setAbgel(cursor.getString(3));
                kennzeichen.setBuLand(cursor.getString(4));
                // Objekt-Eintrag in Liste erzeugen
                kennzeichenList.add(kennzeichen);
            } while (cursor.moveToNext());
        }
        db.close();
        return kennzeichenList;
    }

```

```
111     // Lesen der Auswahl
112     public List<Kennzeichen> lesenEinKennzeichen(String eingabe) {
113         this.eingabe = eingabe;
114         List<Kennzeichen> kennzeichenList = new ArrayList<Kennzeichen>();
115 
116         SQLiteDatabase db = this.getWritableDatabase();
117         Cursor cursor = db.rawQuery(SQL_LESE_AUSWAHL, new String[]{eingabe});
118 
119         // Lesen der Auswahl
120         if (cursor.moveToFirst()) {
121             do {
122                 //neue Kennzeichen-Instanz
123                 Kennzeichen kennzeichen = new Kennzeichen();
124                 kennzeichen.setId(Long.parseLong(String.valueOf(cursor.getLong(0))));
125                 kennzeichen.setAbk(cursor.getString(1));
126                 kennzeichen.setText(cursor.getString(2));
127                 kennzeichen.setAbgel(cursor.getString(3));
128                 kennzeichen.setBuLand(cursor.getString(4));
129                 // Objekt-Eintrag in Liste erzeugen
130                 kennzeichenList.add(kennzeichen);
131             } while (cursor.moveToNext());
132         }
133         db.close();
134         return kennzeichenList;
135     }
136 
137     public Boolean pruefenTabelle() {
138         boolean tabelleVorhanden = false;
139         SQLiteDatabase db = getWritableDatabase();
140         Log.d(LOG_TAG, "Die Tabelle wird erfragt.");
141         Cursor cursor = db.rawQuery(SQL_PRUEFUNG, null);
142 
143         while (cursor.moveToNext()) {
144             String foo = cursor.getString(0);
145             if (cursor.getLong(0) == 1) {
146                 tabelleVorhanden = true;
147             } else {
148                 tabelleVorhanden = false;
149             }
150         }
151         db.close();
152         return tabelleVorhanden;
153     }
154 }
```

Gehen wir auch das schnell durch, die Imports übergehe ich wie gewohnt.

Unsere Klasse erbt von `AppCompatActivity`.

Zeile 23 ist der Eingabe gewidmet, die wir in Zeile 78 zusammenbauen.

Die `onCreate()`-Methode kennen wir schon aus dem auto-generierten Text. Da dies immer der Startpunkt ist, prüfen wir hier auch zuerst die Existenz einer Tabelle.

Dafür rufen wir die `pruefenTabelle()`-Methode im `DBHelper` auf. Wir erinnern uns, dort wurde versucht den ersten Datensatz zu lesen, falls der nicht da ist, wird `tabelleVorhanden` auf `false` gesetzt.

In diesem - und *nur* in diesem Fall lesen wir ab Zeile 38 alles aus dem `kennzeichen.xml`-Datei aus und schreiben es Satz für Satz in die Tabelle. Das passiert dann in Zeile 57.

Die Methode `lesenAuswahl()` in Zeile 75 wird gerufen, wenn der Anwender auf den Los-Button geklickt hat. In diesem Falle besorgen wir uns die Eingabe und starten die Activity.

Gleiches gilt für die Methode `allesLesen()` in Zeile 83. Da wir hier aber keine Eingabe haben, rufen wir direkt die Activity auf.

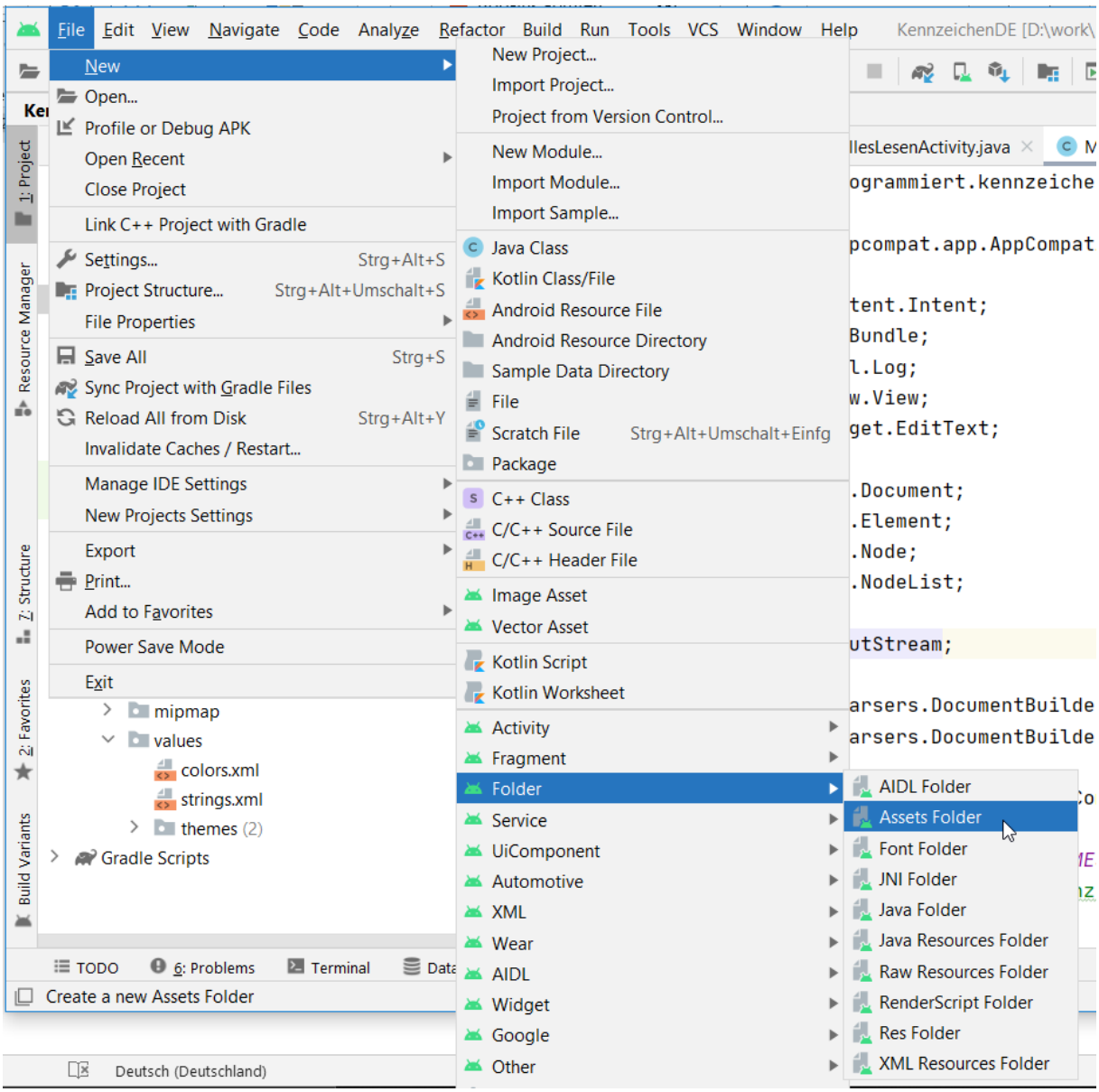
Das war es schon in der `MainActivity`, jetzt sollten sich alle Fehlermeldungen der vorangegangenen Kapitel aufgelöst haben.

Falls nicht, nochmal die Strings kontrollieren oder die Schreibweise der Aufrufe.

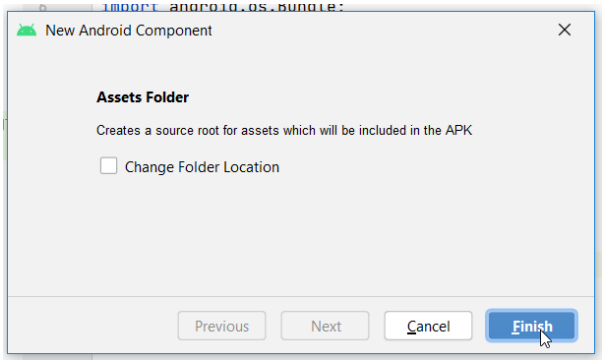
4.3.4. Die Datei `kennzeichen.xml`

Eine Sache müssen wir noch erledigen, bevor wir testen können. Bitte ladet Euch auch die Datei `kennzeichen.xml` herunter, sie enthält alle Kennzeichen, die ich gefunden habe. Diese müssen wir unserer Applikation natürlich noch mitgeben.

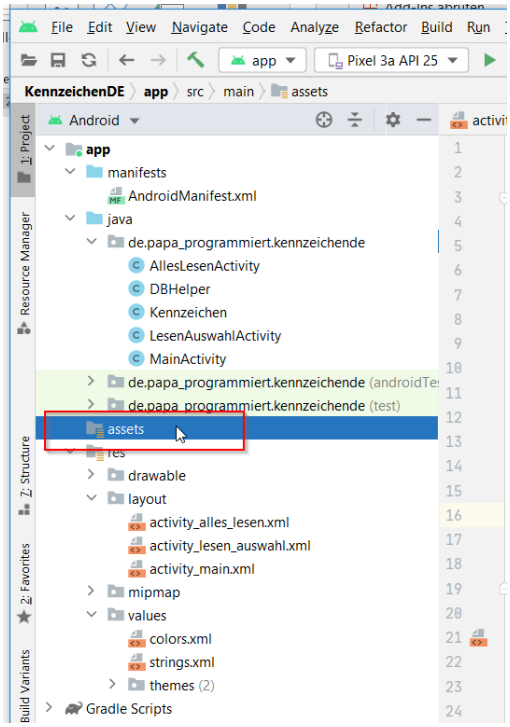
Auch für Dateien hat Android Studio einen eigenen Platz vorgesehen, hier müssen wir zunächst einen „Asset"-Ordner erstellen. Das geht über „File/New/Folder/Asset Folder":



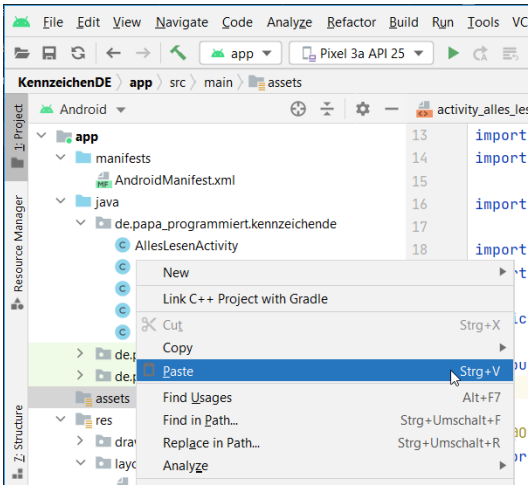
Das nachfolgende Fenster mit „Finish“ bestätigen:

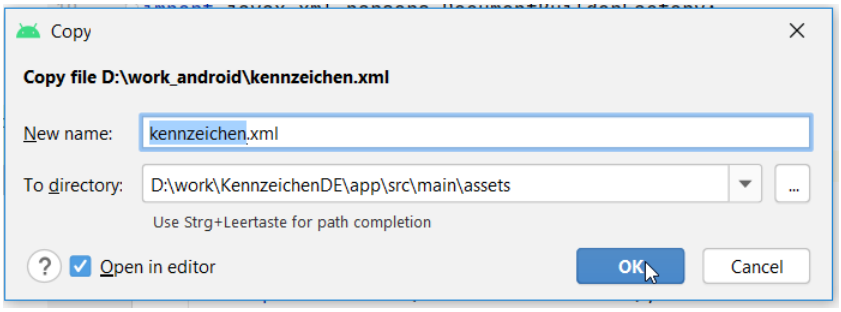


Und es wird ein neuer Ordner angelegt:

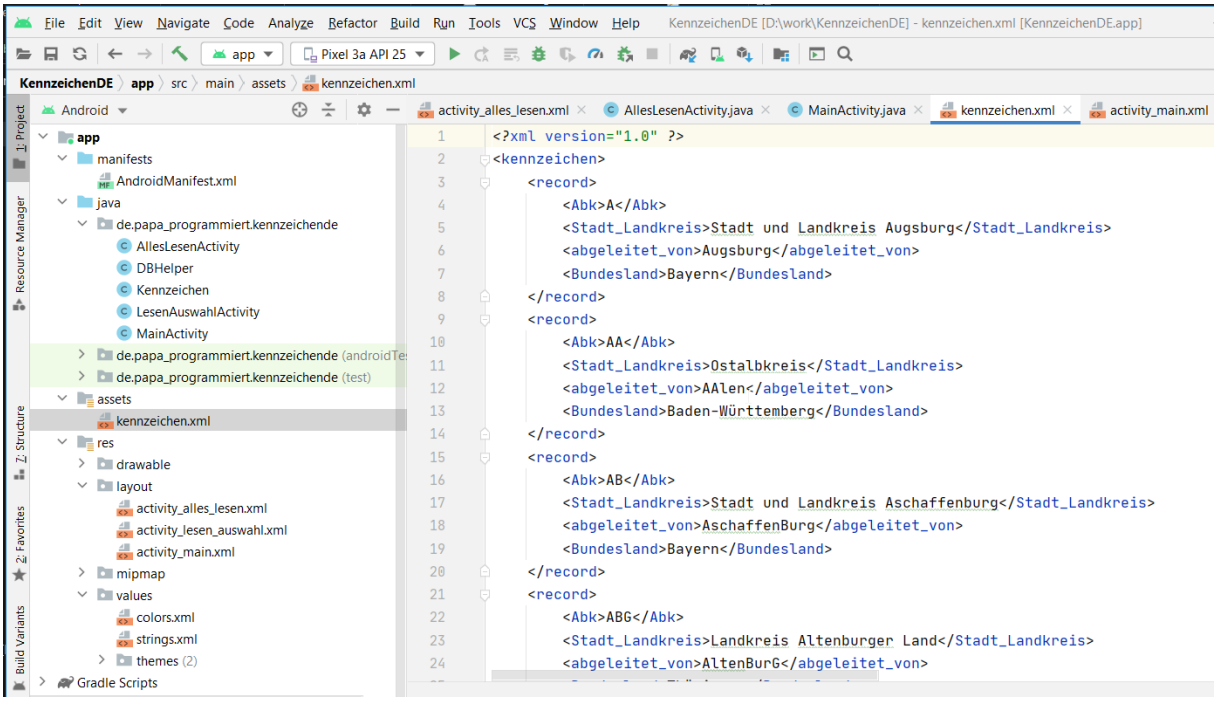


Nun kopieren wir das File per Cut and Paste in das Verzeichnis:





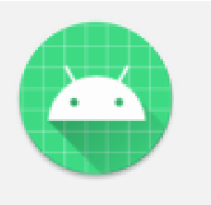
Mit „OK“ bestätigen. Die Datei taucht nun im Asset-Ordner auf und mit Doppelklick können wir und die Inhalte ansehen:



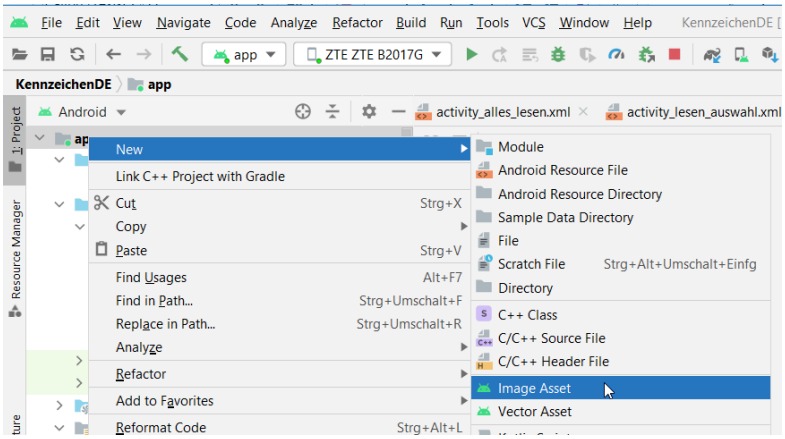
Mehr ist nicht zu beachten.

4.3.5. Ein schönes Bild abgeben

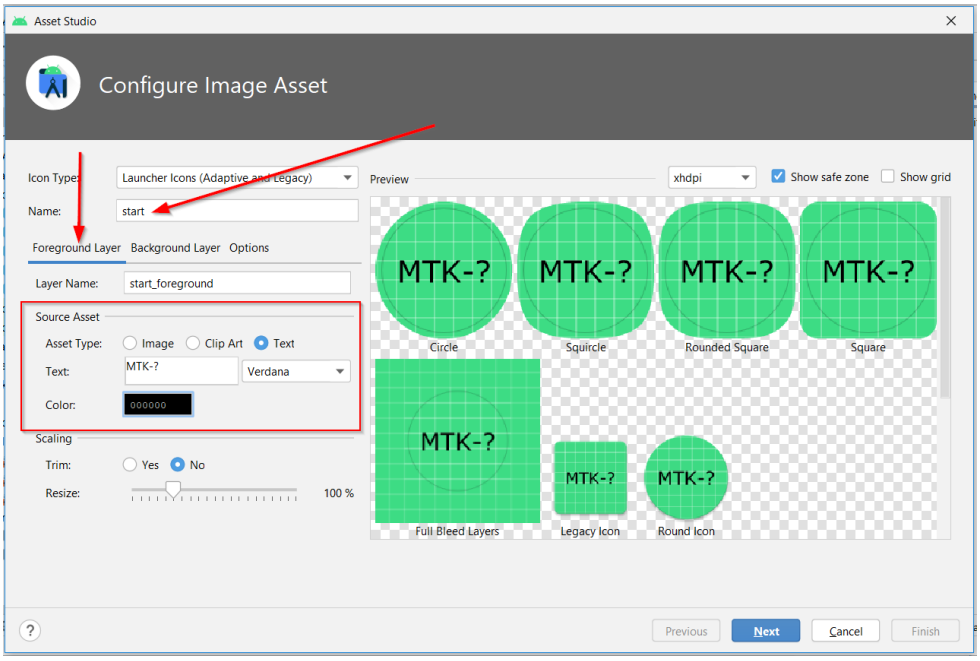
Das Startbild der Applikation sollten wir auch noch ändern, damit wir ein individuelles Design haben anstatt des allgemeinen Bildchens:



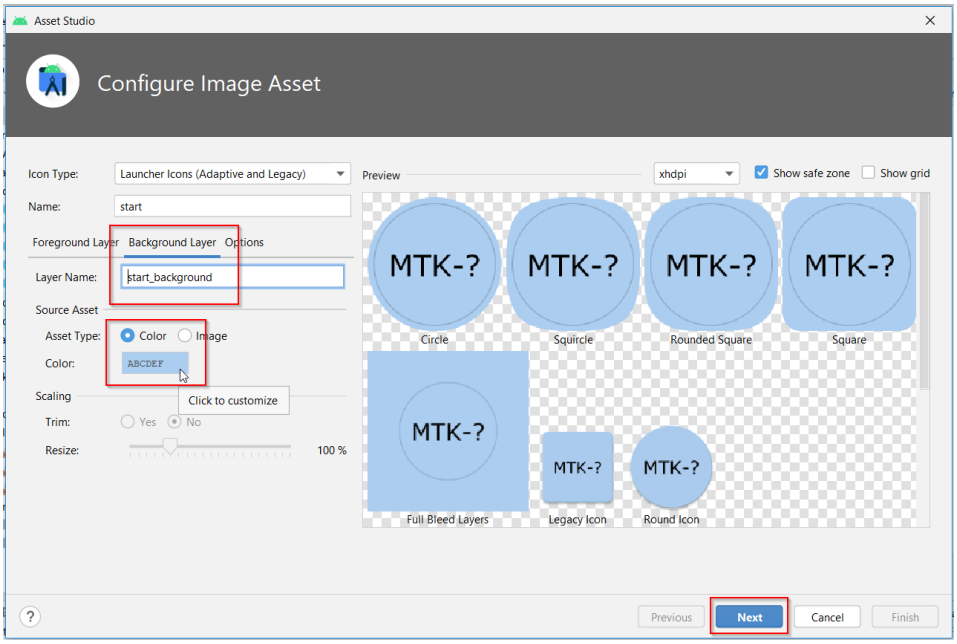
Das machen wir über Rechtsklick auf den obersten app-Knoten, dann „New/Image Asset“



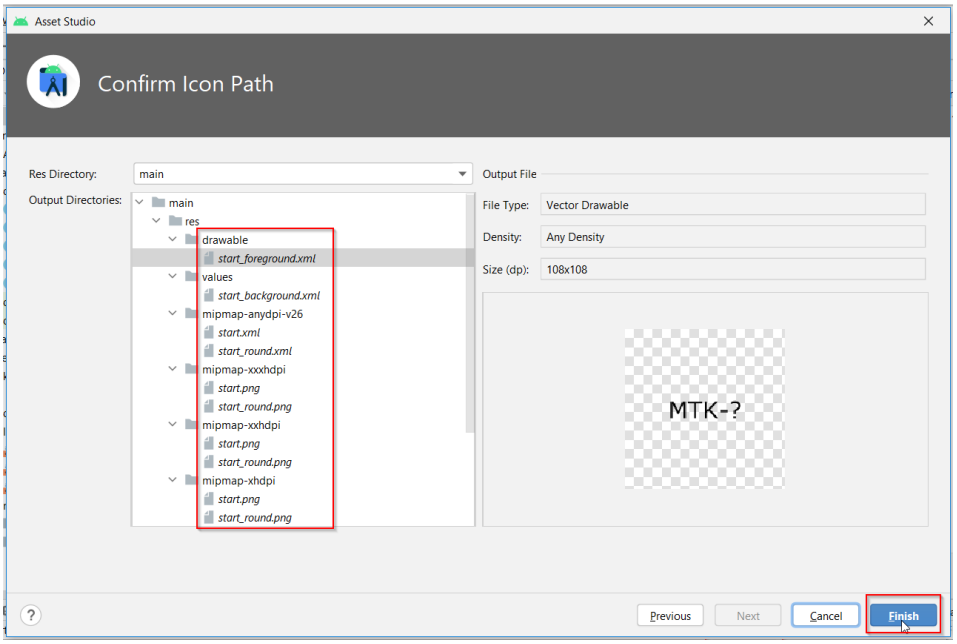
Im nachfolgenden Fenster einen Namen vergeben (bei mir „start“) im „Foreground Layer“ wird der Text bearbeitet, Ihr könnt natürlich etwas anderes festlegen.



Im Background Layout geht es demnach um den Hintergrund, ich habe mich da für ein leichtes Blau entschieden, der Hex-Code dafür ist ABCDEF.



Mit „Next“ kommt man auf die Übersicht für die Ablage der Bilder, dies kann man mit „Finish“ bestätigen:



Damit sind wir auch hiermit fertig.

4.3.6. Die Manifest-Datei

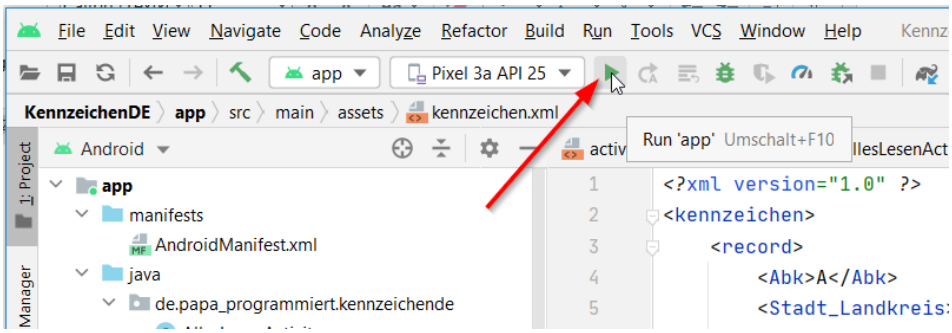
Die `AndroidManifest.xml`-Datei habe ich hier auch als Quellcode hinterlegt:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="de.papa_programmiert.kennzeichende">
4
5      <application
6          android:allowBackup="true"
7          android:icon="@mipmap/start"
8          android:label="@string/app_name"
9          android:roundIcon="@mipmap/start_round"
10         android:supportsRtl="true"
11         android:theme="@style/Theme.KennzeichenDE">
12         <activity android:name=".AllesLesenActivity"
13             android:parentActivityName=".MainActivity">
14             <meta-data
15                 android:name="android.support.PARENT_ACTIVITY"
16                 android:value=".MainActivity" />
17         </activity>
18         <activity android:name=".LesenAuswahlActivity"
19             android:parentActivityName=".MainActivity">
20             <meta-data
21                 android:name="android.support.PARENT_ACTIVITY"
22                 android:value=".MainActivity" />
23         </activity>
24         <activity android:name=".MainActivity">
25             <intent-filter>
26                 <action android:name="android.intent.action.MAIN" />
27
28                 <category android:name="android.intent.category.LAUNCHER" />
29             </intent-filter>
30         </activity>
31     </application>
32
33 </manifest>
```

In den Zeilen 7 und 9 sind die oben erstellten Icons aufgenommen. Das müsst Ihr manuell ändern, per Default sind da „ic_launcher“ und „ic_launcher_round“ hinterlegt.

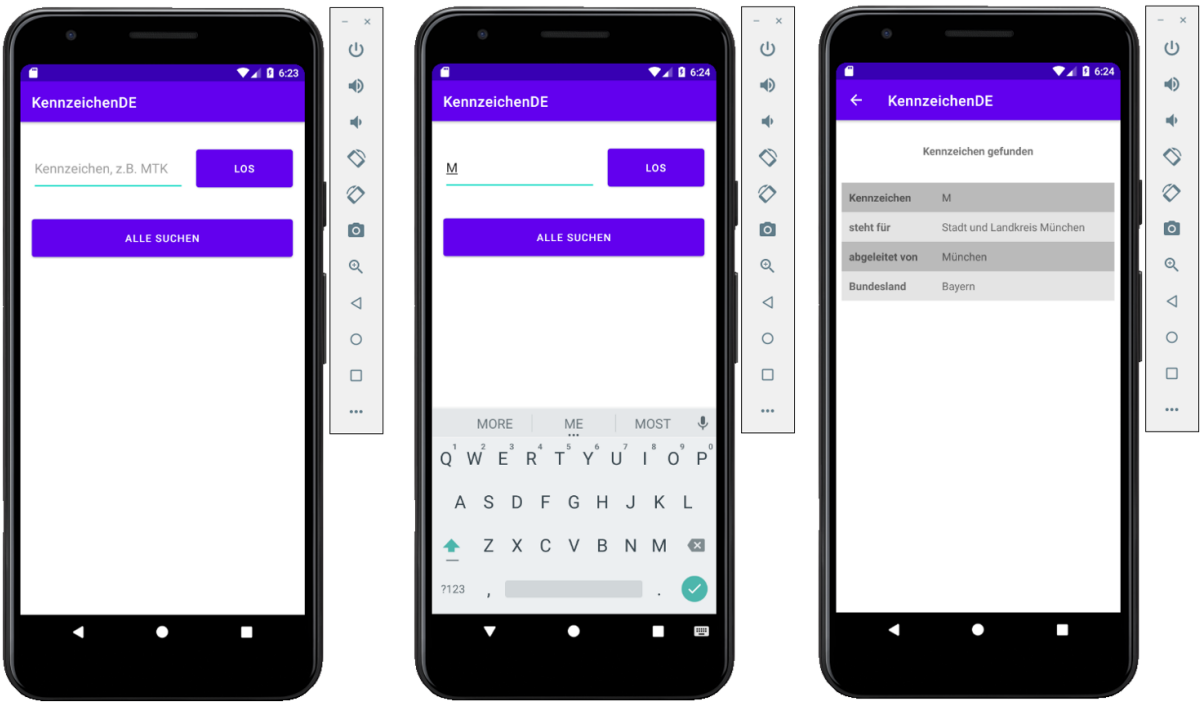
4.4. Test der Anwendung

Zuerst testen wir die App in der Emulation:



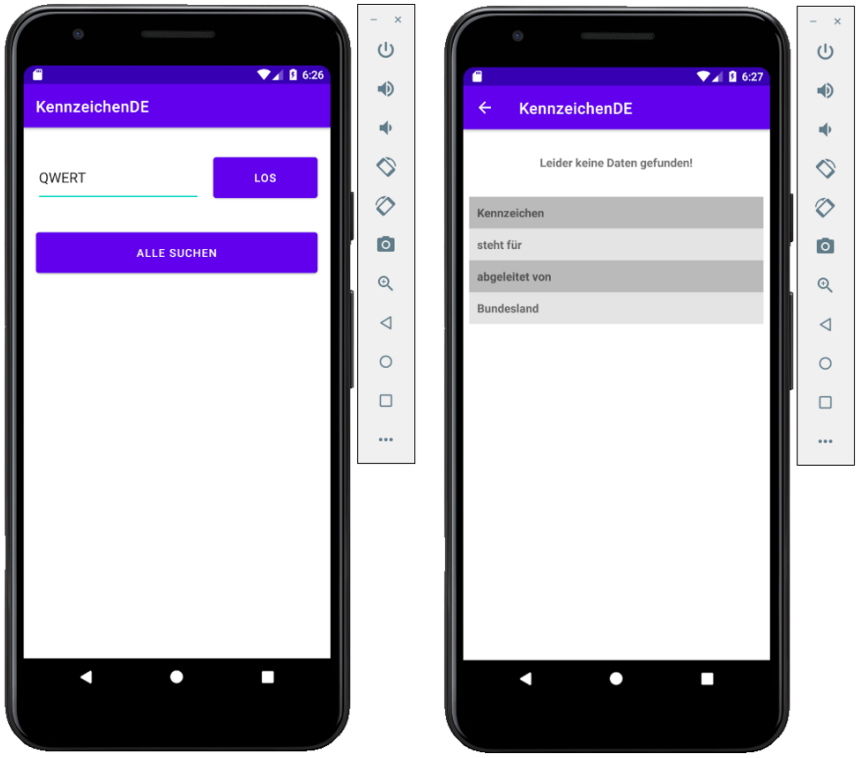
Bei mir braucht das eine Weile, bis das Pixel gestartet ist und die Applikation installiert wurde.

Ein Klick in das Eingabefeld öffnet die Texteingabe und der Klick auf „Los“ zeigt das Ergebnis:

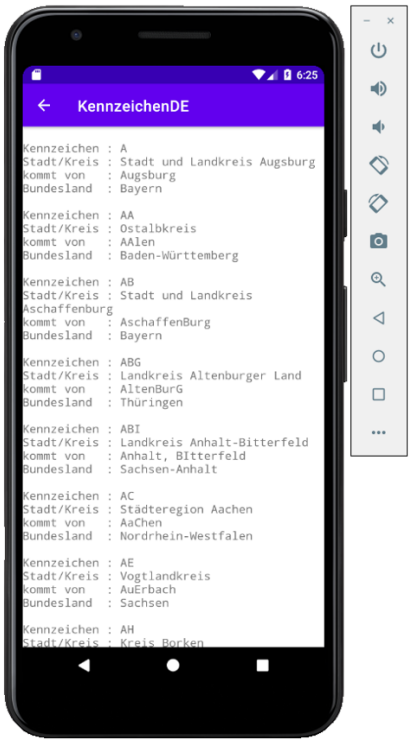


Ich bin begeistert!

Was passiert, wenn kein Eintrag vorhanden ist?

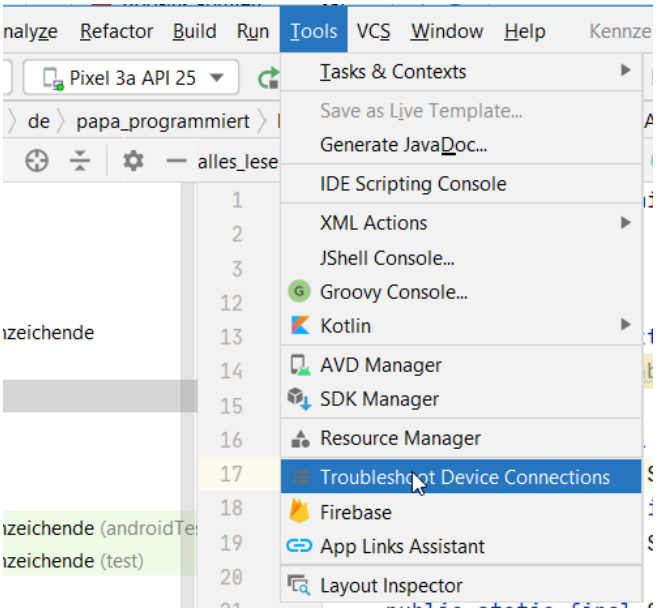


Super, klappt auch. Und alles ansehen?

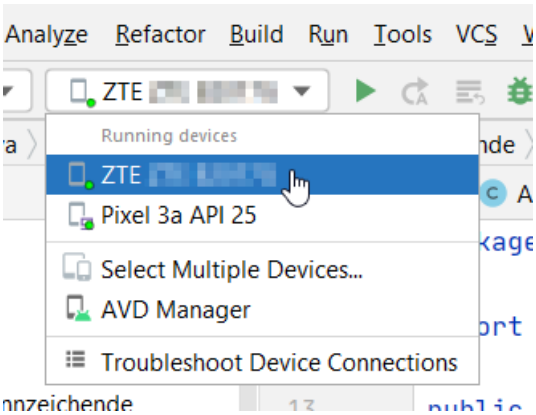


Prima, Ziel erreicht! Wie geht es jetzt weiter? Ich habe ein ZTE und das habe ich mit dem USB-Ladekabel an meinen Rechner angeschlossen. Die Entwickleroptionen sind bei mir aktiviert, was mir erlaubt, das Smartphone im Android Studio einzubinden.

Über „Tools/Troubleshooting Device Connections“ habe ich es geschafft, mein ZTE einzurichten.



Sobald das Smartphone erkannt wird, habt Ihr die Möglichkeit, die Applikationen direkt auf Euer Smartphone zu schicken. Das Gerät muss allerdings entsperrt sein. Am besten Ihr stellt in den Entwickleroptionen „aktiv bleiben“ ein.



Damit habt Ihr keine Probleme mehr mit Installation und allem Drum und Dran. Klasse, oder?

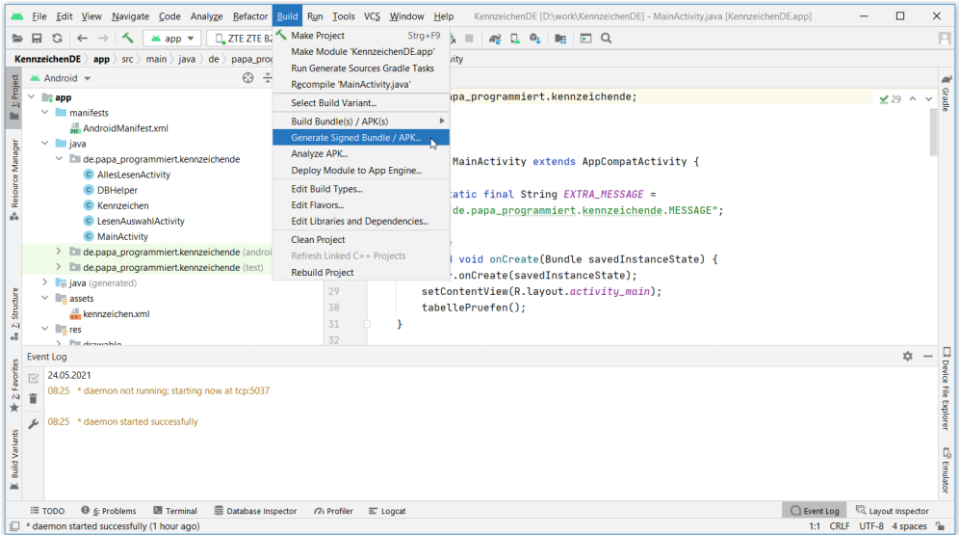
4.5. Die APK-Datei `app-release.apk`

Eine Android-App wird über einen App-Store auf das Gerät gespielt, zuerst erfolgt der Download und dann die Installation. Der gebräuchlichste App-Store ist der Google Play-Store. Damit ein Entwickler seine App dort veröffentlichen kann, muss er sich als Entwickler registrieren. Diese Registrierung ist kostenpflichtig und deshalb habe ich das noch nicht gemacht. Soweit ich das ermitteln konnte, kostet es aktuell einmalig 25 Dollar.

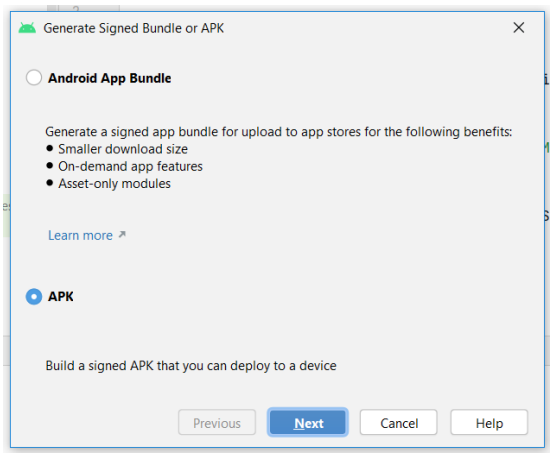
Unabhängig davon, ob Ihr registriert seid oder nicht, muss die App als APK-Datei vorliegen. APK steht für Android Package. Das Betriebssystem auf dem Smartphone braucht also eine APK-Datei um die App zu installieren.

Android Studio stellt dazu einen Prozess zur Verfügung, an dessen Ende genau so eine APK-Datei herauskommt. Wir gehen den Prozess hier durch, das Endergebnis ist auf meiner Homepage veröffentlicht.

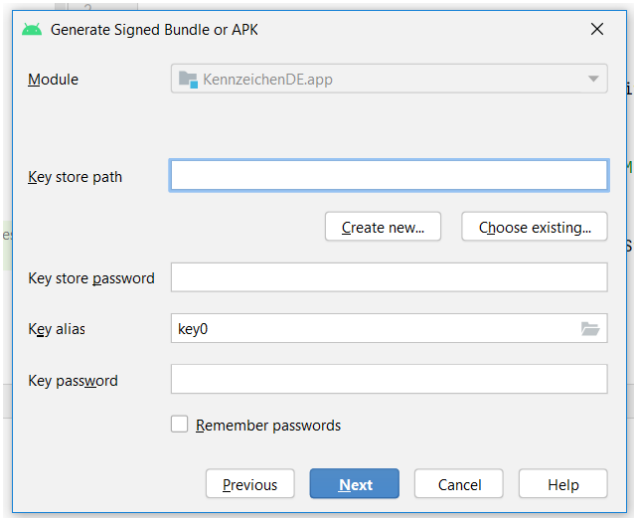
Über das Menu „Build/Generate Signed Bundle / APK…“



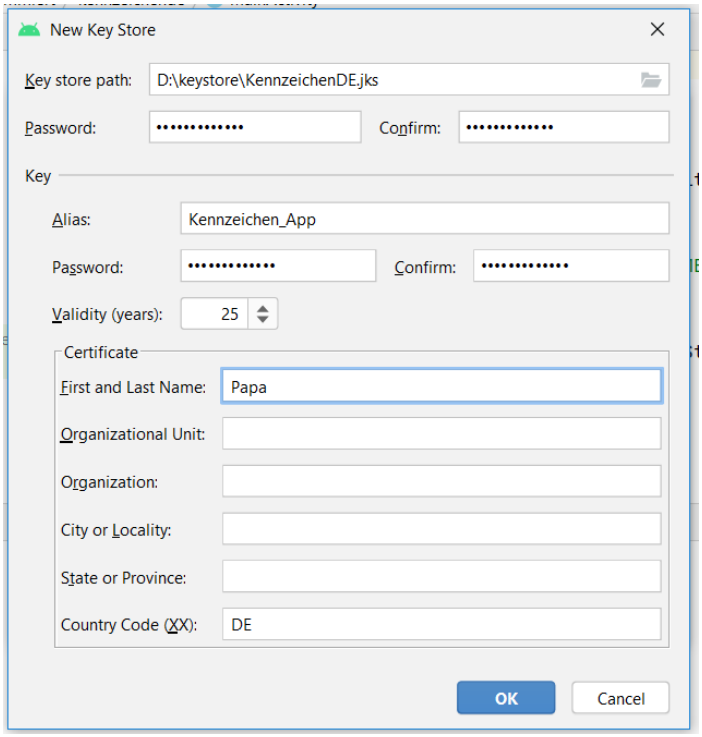
…öffnet sich ein neues Fenster:



Da wir das APK nicht auf einem App-Store vermarkten wollen, richten wir hier nur das APK ein. Mit Next bestätigen, dann gelangen wir zum nächsten Fenster:



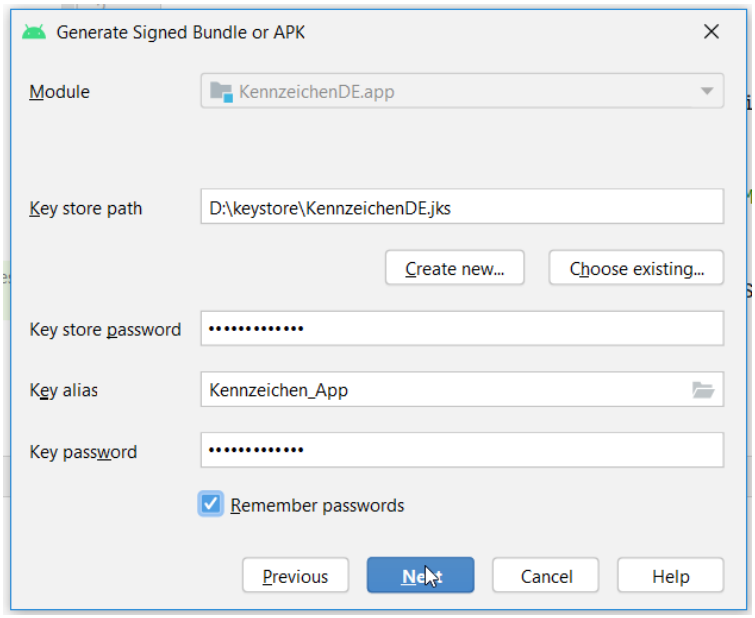
Mit „Create new…“ legen wir einen neuen Keystore an:



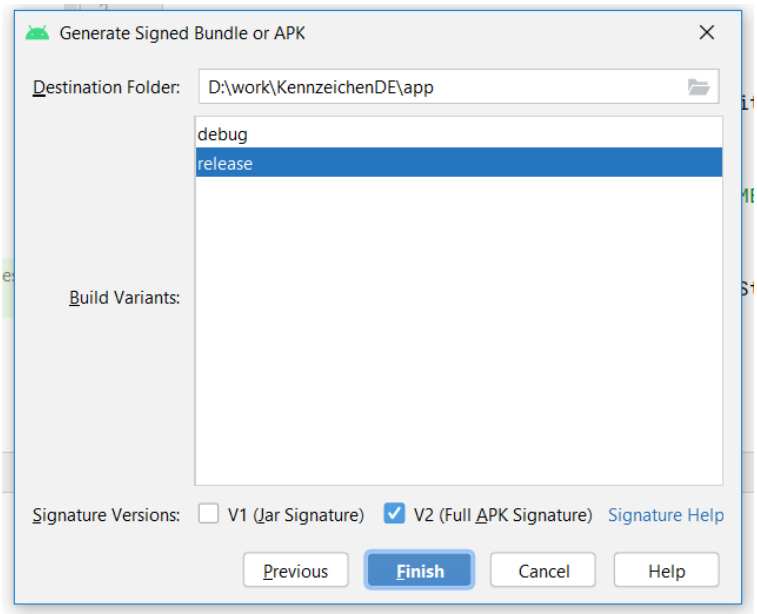
Hier müssen wir einen eigenen Signaturpfad auf unserem Rechner anlegen. Der muss nicht zwingend für jede weitere App neu eingerichtet werden, den kann man wiederverwenden.

Solltet Ihr das ganze professionell betreiben wollen, solltet Ihr die Angaben im Zertifikat vollständig machen, hier habe ich nur den Namen und das Land angegeben.

Mit „OK“ bestätigen:



Da wir uns unserer Sache sicher sind bauen wir eine release-Version:



Nachdem Android Studio gearbeitet hat, ist die fertige APK-Datei im Projekt-Ordner „KennzeichenDE/app/release“ unter `app-release.apk` zu finden. Wie oben erwähnt, habe ich diese Datei auf der Homepage zum Download zur Verfügung gestellt.

Da die Datei nicht vom App-Store kommt, muss man am Smartphone noch die Sicherheitseinstellung ändern. In den „Einstellungen“ gibt es irgendwo den Unterpunkt „Sicherheit“ und da wiederum einen Unterpunkt „Unbekannte Herkunft“ oder so ähnlich, auf jeden Fall ein Menu-Punkt wo Ihr die Installation aus unbekannten Quellen zulassen könnt. Diesen Punkt aktivieren.

Dann in das Verzeichnis wechseln, in das Ihr die APK-Datei kopiert habt und die APK anklicken. Der Rest ist selbsterklärend.

4.6. Abschluss Projekt

In der Einleitung habe ich von einem engen aber passenden Korsett gesprochen, ich hoffe, das ist deutlich geworden. Durch die Vorgaben der Ordner-Strukturen ist es relativ einfach, sich in diesem Gefüge zurecht zu finden. Da alles generiert werden kann, ist das Zusammenstecken der einzelnen Komponenten übersichtlich und gleichartig.

Wie könnte es weitergehen?

Ihr werdet es sicher gemerkt, haben, der erste Start ist ziemlich langsam, da dabei die Datenbank erst erzeugt werden muss und die Tabelle aus dem xml befüllt werden muss.

Es wäre sicher nicht schlecht, dem Nutzer einen Hinweis in Form einer Sanduhr oder ein anderes Wartesymbol anzuzeigen. Damit wären wir aber in einem anderen Thread und damit habe ich mich aktuell noch nicht beschäftigt, kann hier also keinen Erfahrungsbericht abgeben. Ihr könnt das aber gerne ausprobieren, als Stichwort für eine Internet-Recherche zu dem Punkt sei „Android Loading Spinner“ empfohlen.

Auch ist die Optik sicher noch verbesserungswürdig, da könnt Ihr Euch austoben.

Vielleicht kann man auch noch weitere Infos geben, wie den Standort der Stadt auf einer Map anzeigen.

So, das war es von meiner Seite, ich hoffe, ich konnte Euch einen kleinen Einblick geben. Fragen und Anregungen, aber auch, wenn Euch Fehler auffallen, würde ich mich über Rückmeldung über papa@papa-programmiert.de freuen.

Viel Spaß beim Programmieren!