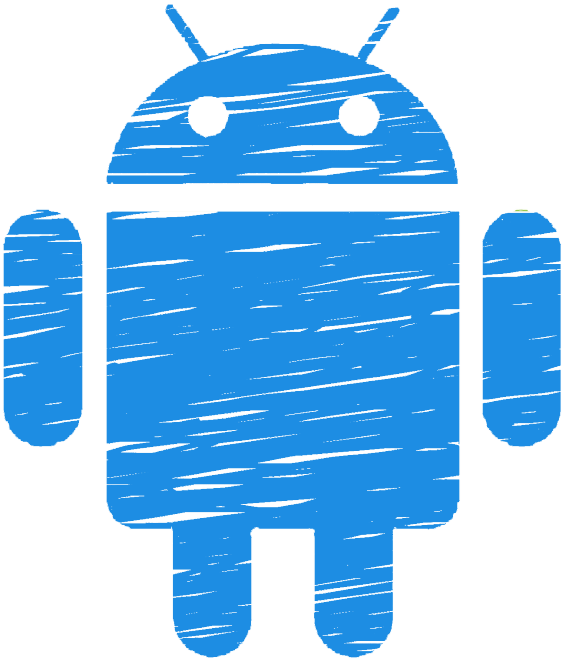
Programmierung von Apps

Skript zum Kurs



Matthias Ruf, Benno Hölz

Kurs: November 2020
Version: 0.3

sfz®
schülerforschungszentrum
südwürttemberg

Inhaltsverzeichnis

1. Java und Objektorientierung - Was ist das?

Android-Apps können in verschiedensten Programmiersprachen programmiert werden, darunter unter anderem Kotlin, C, C++, C# . Am gängigsten ist aber die Programmiersprache **Java**. [1] Da diese Programmiersprache in unseren Apps verwenden wollen, ist es für uns wichtig grundlegende Kenntnisse der Programmiersprache Java zu haben, um unsere Android Apps programmieren zu können.

Achtung: Bei den Programmen in Kapitel 1 handelt es sich um Java-Programme, die so nicht in Android Studio ausführbar sind.

1.1. Hello World

Das einfachste Programm, mit dem man Programmiersprachen einführt ist das **HelloWorld**-Programm. Auch wir wollen mit diesem Programm einsteigen. Dazu ist das Programm im folgenden Codeschnipsel dargestellt.

```
1 public class HelloWorld{
2     public static void main(String[] args){
3         System.out.println("Hello World");
4     }
5 }
```

Die grobe Struktur eines Java-Programms erkennen wir anhand der Einrückungen gut. Jedes Java-Programm ist in einer Datei abgespeichert, die den Namen der Klasse trägt. In unserem Fall ist das die Klasse `HelloWorld`, die in der Datei `HelloWorld.java` gespeichert ist. Dass die Klasse `HelloWorld` heißt, erkennen wir dabei am `public class HelloWorld` in der ersten Zeile.

Anschließend erkennen wir eine Methode (Zeile 2). In unserem Fall ist das die `Main`-Methode. Sie steht innerhalb einer Umgebung (der Klasse), die mit geschwungenen Klammern {} begrenzt wird und eröffnet ebenfalls eine Umgebung. [1] In der inneren Umgebung bzw. dem inneren Block steht der Programmcode, der aus Befehlen entsteht, die nacheinander von oben nach unten abgearbeitet werden.

In unserem Beispiel ist das genau ein Befehl: `System.out.println("Hello World");`. Dieser Befehl ruft eine Methode auf, die uns Java bereitstellt und die den Text *Hello*

[1] Informatiker würden anstelle des Wortes Umgebung das Wort Block verwenden.

World auf der Kommandozeile ausgibt. Bei der App-Programmierung arbeitet man sehr viel mit den Ausgaben auf dem Handydisplay, sodass für uns hauptsächlich die grobe Struktur des Programms und Befehle, die uns das Betriebssystem Android zur Verfügung stellt relevant sind.

1.2. Variablen und Datentypen

Grundlegend für die meisten Programmiersprache ist es, dass Werte gespeichert werden können. In Java nutzt man dazu Variablen. Variablen können deklariert (angelegt) und initialisiert (erste Wertzuweisung) werden. Dabei hat jede Variable einen Datentyp, dieser legt fest was in der Variable gespeichert wird. Beispiele hierfür sind Integer für Ganzzahlen, Char für Zeichen, String für Zeichenketten oder Double für Gleitkommazahlen (Kommazahlen). Ein Beispiel für das Deklarieren und Initialisieren einer Variable ist im folgenden Codeausschnitt dargestellt. Ein paar der angegebenen Datentypen sind so grundlegend, dass sie nicht ausgeschrieben sondern abgekürzt werden. [2]

```
1  public class Variablen{
2      public static void main(String[] args){
3          int a;  // Deklariert eine Integer-Variable mit dem Namen a
4                  // Legt eine Integer-Variable mit dem Namen a an
5          a = 5;  // Initialisiert die vorher erzeugte Integer-Variable mit dem Wert 5
6                  // Weist der vorher erzeugten Variable a den Wert 5 zu
7          String str = "Hallo Welt"; //Deklarieren und Initialisieren eines Strings in
               einem Befehl
8
9
10     }
11 }
```

1.2.1. Datentypen

Eine Übersicht weiterer Datentypen mit vereinfachter Schreibweise und beispielhaften Werten ist in Tabelle 1 im Anhang auf Seite 37 dargestellt. Für uns sind nicht alle der dort dargestellten Datentypen relevant, es kann jedoch von Vorteil sein, ein paar der Datentypen schon einmal gesehen zu haben.

1.2.2. Rechnen mit Datentypen

Die meisten Operatoren, wie wir sie aus der Mathematik kennen, sind auch in Java vertreten. Ein paar Beispiele sind im folgenden Codeausschnitt dargestellt.

```
1  public class Operatoren{
2      public static void main(String[] args){
3          int a = 40 + 2; // klassische Addition
4          int b = a * 2; // klassische Multiplikation
```

[2] // kann verwendet werden um Kommentare zu verfassen. Alles was in der Zeile hinter dem // steht wird nicht als Programmcode gewertet und entsprechend ignoriert.

```
5           c = b - 42; // klassische Subtraktion
6           int d = 42 / 21; // ganzzahlige Division (Ergebnis hat keine
                Nachkommastellen)
7           double e = 42.0 / 4 ; // klassische Division (Ergebnis hat Nachkommastellen)
8       }
9   }
```

Sollte später im Programm ein Fehler auftreten, empfiehlt es sich, die Rechnungen auf Seiteneffekte zu untersuchen. Es kann sein, dass sich Java bei Berechnungen etwas anders verhält als wir es aus der Mathematik kennen und Ergebnisse so vom erwarteten, mathematisch richtigen Wert abweichen.

1.3. Schleifen und Kontrollstrukturen

Ebenfalls wichtig in der Programmierung von Anwendungen sind die sogenannten Kontrollstrukturen. Kontrollstrukturen sind dazu da, abhängig von Bedingungen den Programmablauf zu regeln. Die bekanntesten Kontrollstrukturen sind die If-Abfragen sowie die Schleifen.

1.3.1. If-Statment bzw. If-Abfrage

Eine If-Abfrage verwenden wir, um eine Befehlsfolge abhängig von einer Bedingung auszuführen oder nicht. Eine Abfrage unterscheidet sich von einer Schleife dadurch, dass die Abfrage nicht noch einmal durchlaufen wird, selbst dann nicht, wenn die Bedingung nach dem Durchlauf noch immer erfüllt sein sollte. Die Syntax[3] ist im folgenden Codeschnipsel dargestellt.

```
1  public class Abfragen{
2      public static void main(String[] args){
3
4          boolean bob = true;
5          if(bob){
6              System.out.println("Wahr");
7          }else{
8              System.out.println("Falsch");
9          }
10
11     }
12 }
```

In unserem Beispiel würde nun *Wahr* ausgegeben werden. Würde man die Variable bob mit `false` belegen wäre die Ausgabe *Falsch*. Der Trick ist eine Boolean-Variable bspw. durch einen Vergleich zweier Integer-Variablen zu generieren und so die Bedingung abhängig von einem Variablenwert zu gestalten.

[3]Syntax bedeutet in der Informatik soviel wie Schreibweise, Semantik soviel wie Bedeutung

1.3.2. For- und While-Schleife

Um etwas wiederholt ausführen zu können, gibt es in den meisten Programmiersprachen[4] Schleifen. Schleifen wiederholen bestimmte Befehle solange, bis ihre Bedingung nicht mehr erfüllt ist. In Java gibt es verschiedene Typen von Schleifen. Die zwei gängigsten Schleifen in Java sind die For- und die While-Schleife.

While-Schleife

Die einfachste Schleife ist die While-Schleife. Sie wiederholt wie bereits erwähnt ihren Schleifenrumpf solange, bis ihre Bedingung nicht mehr erfüllt ist. Die Syntax ist im folgenden Codeschnipsel dargestellt.

```
1 public class Schleifen1{
2     public static void main(String[] args){
3         int i = 0;
4         while(i < 5){
5             System.out.println( i ); // gibt die Variable aus
6             i = i + 1; // erhoeht die Variable um 1
7         }
8     }
9 }
```

Das dargestellte Programm prüft vor jedem Schleifendurchlauf die Bedingung und führt dann die Befehle (den Befehl) in den geschweiften Klammern aus. In diesem Beispiel führt das dazu, dass die Zahlen von 0 bis 4 ausgegeben werden.

For-Schleife

Die For-Schleife ist insbesondere für das Hochzählen von Variablen geeignet. Sie ist in diesen Situationen etwas kompakter als die While-Schleife.[5] Die Syntax ist im folgenden Codeschnipsel dargestellt.

```
1 public class Schleifen2{
2     public static void main(String[] args){
3         for(int i = 0;i < 5; i = i + 1){
4             System.out.println( i ); // gibt die Variable aus
5         }
6     }
7 }
```

1.4. Methoden

Methoden werden für uns in der App-Entwicklung besonders wichtig. Eine Methode[6] ist in gewisser Weise ein **Unterprogramm**. Methoden werden verwendet um Codeabschnitte auszulagern und den Code besser zu strukturieren. Außerdem kann es sein, dass

[4] Aber bei Weitem nicht in allen Programmiersprachen
[5] Bei Arten von Schleifen können jedoch das selbe, es handelt sich lediglich um eine andere Syntax
[6] in anderen Programmiersprachen auch Funktion genannt

man später exakt den selben Code noch einmal benötigt. Damit man diesen dann nicht kopieren und so in manchen Fällen tausende Male schreiben muss, lagert man ihn einmal aus und verwendet ihn an bestimmten Stellen im Programm wieder.

Ein Beispiel, das die Fakultät von 3 verschiedenen Zahlen berechnen soll ist im folgenden Codeschnipsel dargestellt.

```
1  public class Fak{
2      public static void main(String[] args){
3          int fak5 = fak(5);
4          int fak6 = fak(6);
5          int fak9 = fak(9);
6      }
7
8      /*
9       * num nennt man Parameter, er wird der Methode "uebergeben"
10      */
11     public static int fak(int in){
12
13         // negative Zahlen und die 0 abfangen
14         if(in <= 1){
15             return in;
16         }
17         // Fakultaet berechnen
18         for( int i = 1; i <= in; i=i+1){
19             out = out * i;
20         }
21         //Ergebnis zurueckgeben
22         return out;
23
24     }
25 }
```

Wir erkennen einen Methodenaufruf in den Zeilen 3, 4 und 5. An diesen Stellen soll jeweils die Fakultät für die Zahlen 5, 6 und 9 berechnet werden.[7] In Zeile 11 steht der Methodenkopf. Er besteht aus den Wörtern `public` und `static`, die wir beide erst einmal hinnehmen dürfen. Direkt gefolgt werden die beiden Wörter in unserem Beispiel von einem `int`. Dieses `int` steht für den Rückgabedatentyp. Also den Datentyp der den Rückgabewert (das Ergebnis) beschreibt. In unserem Fall wollen wir eine Ganzzahl erhalten und haben deshalb Integer gewählt. Direkt danach steht der Name der Methode. Mithilfe ihres Namens können wir die Methode aufrufen.[8] In den darauf folgenden Klammern `()` stehen die Parameter, die beim Methodenaufruf mit angegeben werden müssen. Diese können im Methodenrumpf verwendet werden. Die restlichen Variablen, die in der vorherigen Methode verwendet wurden, können sofern sie nicht als Parameter übergeben werden nicht in der Methode verwendet werden. Als Letztes muss nun noch das Schlüsselwort `return` besprochen werden. Das Schlüsselwort `return` wird gefolgt von einem Wert oder einer Variable verwendet um diese zurückzugeben. `return` kann mehrfach beim Implementieren[9] einer Methode verwendet werden, jedoch beendet das erste erreichte `return` die Ausführung der Methode und das Programm läuft an der Stelle im Programmcode weiter, von dem aus die Methode aufgerufen wurde.

[7] Die Fakultät ist in der Mathematik für positive Zahlen definiert als $\text{fak}(n) = 1 \cdot 2 \cdot \ldots \cdot n$.
[8] So geschehen in den Zeilen 3, 4 und 5
[9] Programmieren

1.5. Objektorientierung

Als letzte noch fehlende Grundlage für den Einstieg in die App-Programmierung fehlt noch das Konzept der Objektorientierung. Objektorientierung bedeutet, dass wir eigene Datentypen definieren können.

Um nun einen eigenen Datentyp zu definieren benötigen wir ein Art Rezept (oder Stempel), der uns vorgibt wie der entsprechende Datentyp aussehen soll. Die Rolle dieses Stempels übernimmt eine Klasse für uns.

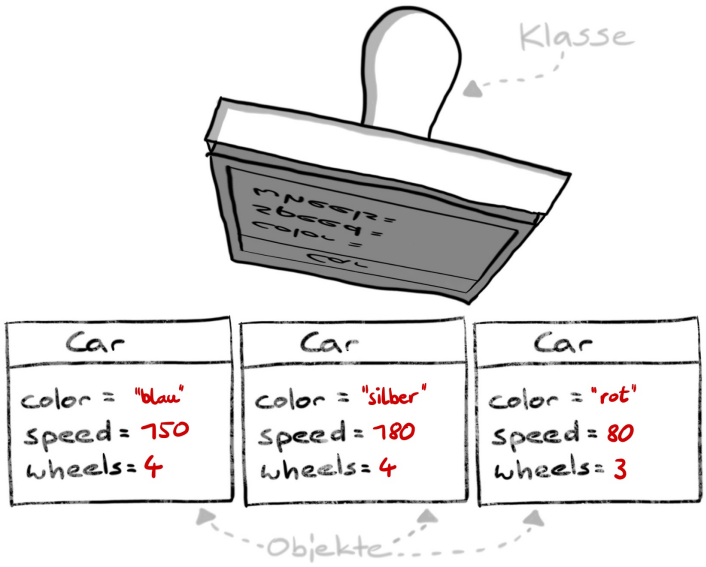


Abbildung 1.: Übersicht zur Objektorientierung [2]

Der Begriff Klasse wurde bereits beim Hello-World-Programm angesprochen. Es handelt sich dabei um eine Datei mit der Endung `.java`, die das bekannte Schema erfüllt. Das bekannte Schema ist im Folgenden dargestellt.

```
1 public class <Name des Datentyps> {
2
3 }
```

Um nun einen Datentyp zu definieren können diesem Schema Attribute hinzugefügt werden. Als Attribute werden Datentypen verstanden, die Teil des neu zu definierenden Datentyps sind. Beispielsweise kann bei einem Datentyp `Auto` mit einer Bytevariable die Tankfüllung angeben oder eine Booleanvariable dazu verwenden, um anzuzeigen, ob das Auto fahrbereit ist oder nicht. Auch hierzu gibt es ein Beispiel:

```
1 public class Auto{
2
3     // Attribute
4     byte tank;
5     boolean fahrbereit;
6
7     // Main-Methode falls gewuenscht
```

```
8       public static void main(String[] args){
9           // Hier koennte Programmcode stehen
10      }
11  }
```

Nun haben wir eine Datentyp definiert, damit wir nun Objekte[10], also Werte dieses Datentyps erzeugen können, benötigen wir noch eine Methode, die das für uns übernimmt. Diese Methode heißt Konstruktor. Sie wurde im folgenden Codeabschnitt hinzugefügt.

```
1   public class Auto{
2
3       // Attribute
4       byte tank;
5       boolean fahrbereit;
6
7       // Konstruktor
8       public Auto(){
9           tank = 0;
10          fahrbereit = false;
11      }
12
13      // Main-Methode falls gewuenscht
14      public static void main(String[] args){
15          // Hier koennte Programmcode stehen
16      }
17  }
```

Was uns als letztes jetzt noch fehlt sind Methoden, die speziell auf ein Objekt angewendet werden können. Sie kennzeichnen sich dadurch, dass kein `static` im Methodenkopf verwendet wird. In diesen Methoden können wir die Werte des Objekts ansprechen, ohne das diese zusätzlich übergeben werden mussten. Dabei handelt es sich immer um die Attribute des Objekts, mit dem die Methode aufgerufen wurde.[11]

```
1   public class Auto{
2
3       // Attribute
4       byte tank;
5       boolean fahrbereit;
6
7       // Konstruktor
8       public Auto(){
9           tank = 0;
10          fahrbereit = false;
11      }
12
13      // Main-Methode falls gewuenscht
14      public static void main(String[] args){
15          // Objekt erzeugen
16          Auto audi = new Auto();
17          // Methode auf das Objekt anwenden
18          audi.tanken();
19          audi.fahren();
20      }
21
22      // tank-Methode
23      public void tanken(){
24          tank = 127;
25          fahrbereit = true;
```

[10] Nun ist auch klar warum das Kapitel Objektorientierung heißt.

[11] Wir unterscheiden also zwischen zwei Arten von Methoden den statischen (werden **ohne** Objekt aufgerufen) und den nicht statischen (werden **mit** Objekten aufgerufen) Methoden.

```
26      }
27
28      // fahr-Methode
29      public void fahren(){
30          if(tank <= 0){
31              System.out.println("Bitte tanken!");
32              fahrbereit = false;
33          }
34          if(fahrbereit){
35              System.out.println("Brumm Brumm!");
36              tank = tank - 1;
37          }
38      }
39
40  }
```

Fazit In diesem Kapitel haben wir die wichtigsten Konzepte der Programmiersprache Java besprochen. Sie müssen nicht alles davon im Detail verstanden haben, um Ihre erste App programmieren zu können. Sie werden jedoch merken, dass es von Vorteil sein kann, die wichtigsten Konzepte zumindest schon mal angerissen zu haben.

2. Installation von Android Studio

Um Android-Apps zu programmieren verwenden wir **Android Studio**. Dabei handelt es sich um eine sogenannte **IDE**, also eine Entwicklungsumgebung die uns beim Programmieren z. B. mit Autovervollständigung unterstützt und mit der wir später auch unsere Apps auf dem Smartphone installieren können. Die Installation von Android Studio wir im Folgenden erklärt.

2.1. Windows

Der Installer für Android-Studio unter Windows (64-bit) kann auf folgender Website heruntergeladen werden:

https://developer.android.com/studio

Während der Installation werden wir gefragt, ob wir `Android Virtual Device` installieren wollen:

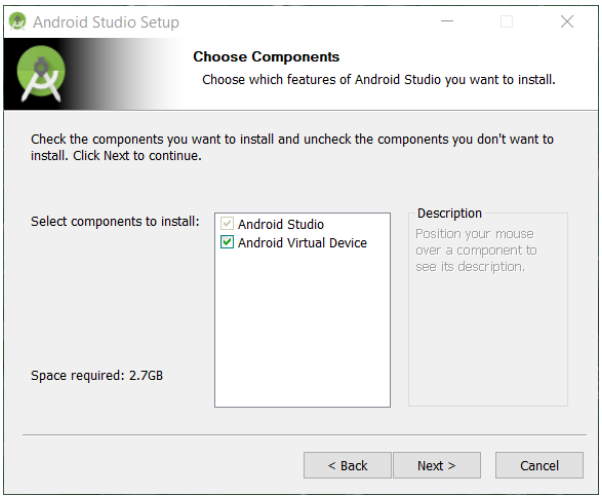


Abbildung 2.: Android Studio-Installer

Dabei handelt es sich um ein virtuelles Gerät, auf dem wir vom PC aus Apps testen können ohne sie auf einem echten Smartphone installieren zu müssen. Das kann praktisch sein, wenn wir gerade kein Android-Smartphone zur Hand haben oder unsere App auf anderen Gerätetypen testen wollen. Wir setzen also einen Haken bei

`Android Virtual Device`. Im Anschluss wird Android Studio installiert.

2.2. macOS

Möchten wir Android Studio auf einem Mac installieren, so müssen wir uns auch dafür zunächst den Installer herunterladen:

https://developer.android.com/studio

Nach dem Download öffnen wir die `.dmg`-Datei, woraufhin sich folgendes Fenster öffnet:

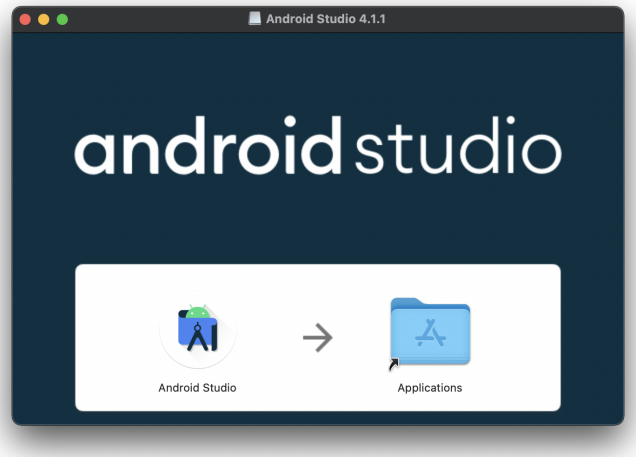


Abbildung 3.: Mac-Installer

Hier ziehen wir nun das Android-Studio-Symbol einfach mit der Maus auf den Applications-Ordner. Ist der Kopiervorgang abgeschlossen, können wir Android Studio auch schon beispielsweise über das Launchpad ausführen. Beim ersten Öffnen werden wir noch gefragt, ob wir die Anwendung auch wirklich ausführen möchten, da sie von einem Drittanbieter stammt:

Abbildung 4.: Meldung beim ersten Start

Wir bestätigen dies mit einem Klick auf `Öffnen`.

2.3. Linux - Ubuntu

Um die Installation von Android Studio unter Ubuntu (ab Version 18.04) durchzuführen reicht es einen Befehl im Terminal auszuführen. Man öffnet also mit der Tastenkombination `Strg+Alt+T` ein neues Terminal und tippt die folgende Zeile ab.[3]

```
1 snap install android-studio --classic
```

Alternativ ist auch eine Installation über das Softwarecenter möglich. Hier reicht es nach *Android Studio* zu suchen, es auszuwählen und den `installieren`-Button zu klicken.

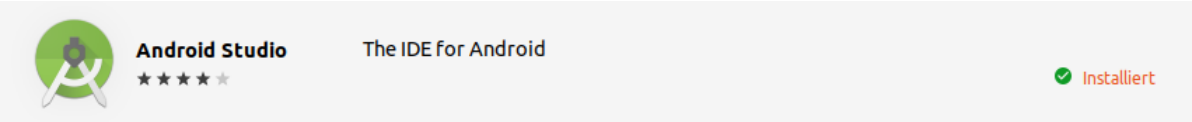


Abbildung 5.: „Ubuntu Software“ nach der Installation

3. Rundgang durch Android Studio

3.1. Ersteinrichtung

Wenn wir Android Studio nach der Installation das erste Mal öffnen, müssen wir noch ein paar Einrichtungsschritte durchlaufen.

Direkt zu Beginn öffnet sich ein **Welcome**-Fenster. Um ins nächste Fenster zu gelangen klicken wir auf `Next`.

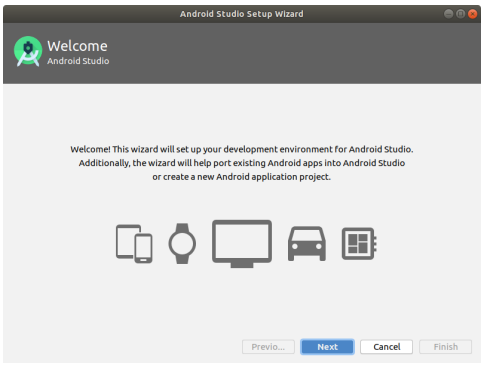


Abbildung 6.: Welcome - Fenster

Wir gelagen dadurch ins nächste Fenster. Hier wählen wir das Standardsetup aus und klicken wieder auf `Next`.

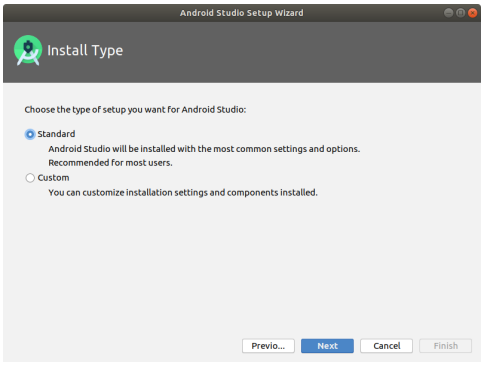


Abbildung 7.: Auswahlfenster zum Setup

Als nächstes steht die Wahl des Farbschemas an. Im Beispiel habe ich das Darcula Design ausgewählt. Für welches Sie sich entscheiden bleibt Ihnen überlassen.

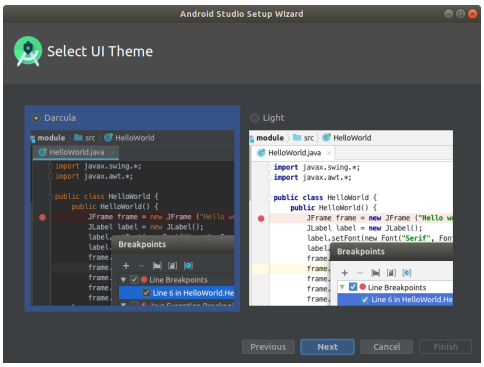


Abbildung 8.: Wahl des Farbschemas

Danach gelangen wir ins nächste Fenster. In diesem Fenster können wir die gewählten Einstellungen überprüfen. Wir verlässt das Fenster durch einen Klick auf `Next`. Es öffnet sich ein weiteres Fenster. In diesem Fenster stehen nun Informationen zum Virtual Device (siehe Abb. 9).

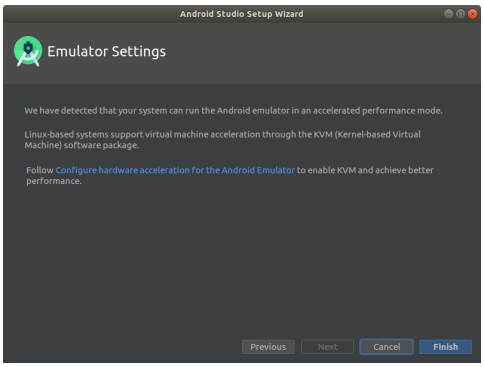


Abbildung 9.: Überprüfen der Virtual-Device-Einstellungen

Um das Setup abzuschließen klicken wir auf `Finish`. Es beginnt ein Download. Nach diesem Download und der Installation einiger zusätzlicher Elemente können wir wieder mit einem Klick auf `Finish` das Setup endgültig abschließen. Es öffnet sich Android Studio und man erhält das Fenster in Abb. 10 .

Abbildung 10.: Startbildschirm Android Studio

Der Vorgang wie er hier dargestellt wurde bezieht sich auf das Betriebssystem Ubuntu (18.04), ist aber in Windows (10) quasi identisch.

3.2. Neues Projekt anlegen

Um ein neues Projekt[1] anzulegen, klicken wir auf `File > New > New Project` bzw. auf dem Willkommensbildschirm auf `Create New Project`. Es öffnet sich ein Fenster, in dem wir unser neues Projekt konfigurieren können:

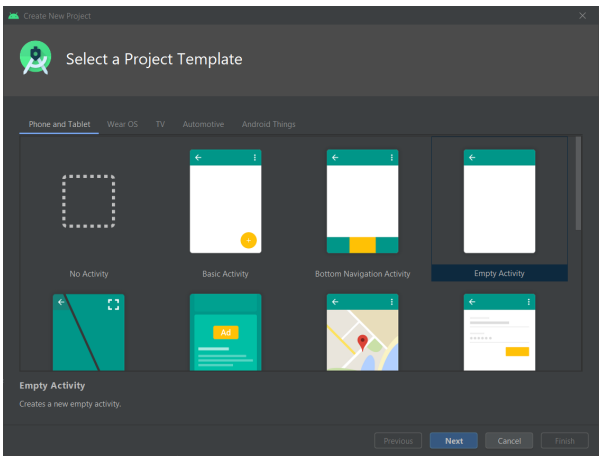


Abbildung 11.: Anlegen eines neuen Projekts

Hier können wir eine Vorlage für unser Projekt auswählen. Wir beginnen mit der Vorlage `Empty Activity` unter `Phone and Tablet`. Dabei handelt es sich um einen leeren Bild-

[1] Eine neue App

schirm, die nötigen GUI-Elemente werden wir später selbst hinzufügen. Anschließend klicken wir auf `Next`. Nun müssen wir noch einige Projekteigenschaften festlegen:

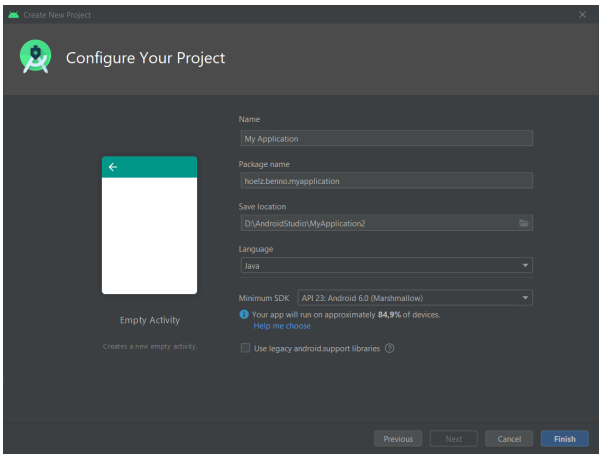


Abbildung 12.: Eigenschaften des neuen Projekts

Auf diesem Bildschirm können wir der App einen Namen geben und eine minimal nötige Android-Version auswählen. Wir verwenden hier Android 6.0. Für den Anfang genügt diese Version. Wenn unsere Projekte komplexer werden müssen wir hier bei Bedarf eine größere Version wählen, da mache Features in früheren Versionen von Android noch nicht unterstützt werden[2]. Welche Android-Version auf dem eigenen Smartphone installiert ist, lässt sich in den Einstellungen des Handys unter `System > Über das Telefon`[3] herausfinden:

[2] z.B. kann erst ab Version 8.0 der Minimalwert eines Sliders definiert werden
[3] Die Bezeichnungen der Menüpunkte können je nach Hersteller unterschiedlich sein!

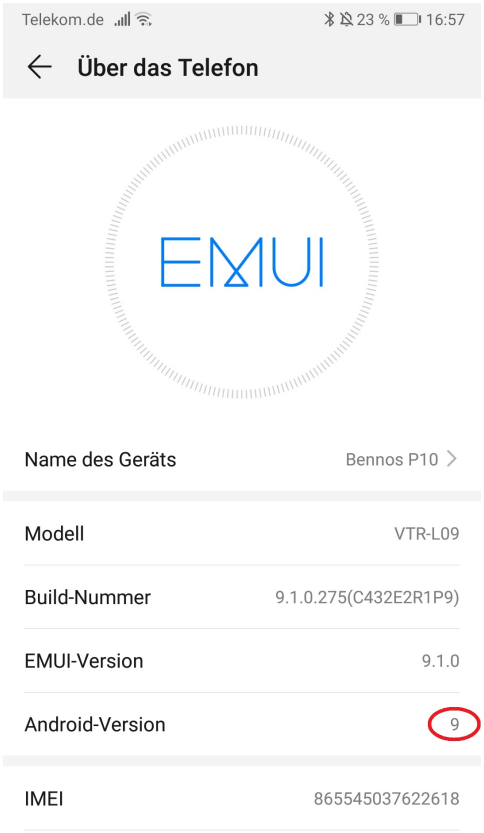


Abbildung 13.: Screenshot der Einstellungen eines Android-9-Smartphones

Beim `Package name` handelt es sich gewissermaßen um die Ordnerstruktur des Projektes, wobei Unterordner durch . getrennt sind. Am besten überlegt man sich hier ein einheitliches Schema oder verwendet die Standardeinstellung. Schließlich müssen wir noch einen Speicherort für unser Projekt angeben und als Programmiersprache Java auswählen. Mit einem Klick auf `Finish` wird das Projekt erstellt. Das kann eine Weile dauern.

3.3. Projektstruktur

Schauen wir uns nun die Struktur unseres neu angelegten Projekts an. Falls nicht bereits geschehen, öffnen wir dazu auf der linken Seite den Tab `1:Project` und achten auf die Darstellungsform `Android` (oben links im Bild):

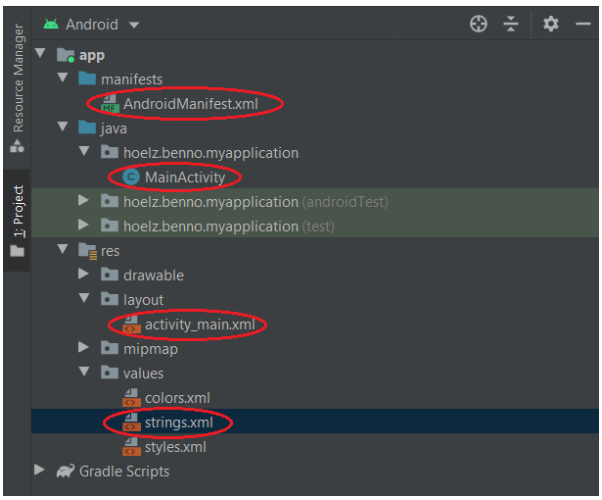


Abbildung 14.: Projektstruktur in Android Studio

Wir finden unter anderem folgende fünf Dateien, die für uns wichtig sind:

1. `activity_main.xml`
   In dieser XML-Datei wird die graphische Oberfläche (GUI) der App definiert[4].
2. `strings.xml`
   Hier können verschiedene Texte abgespeichert und später referenziert werden. Dies ist z.B. praktisch, wenn man seine App zusätzlich in einer anderen Sprache anbieten möchte, in diesem Fall muss nur auf eine andere `strings.xml`-Datei verwiesen werden um den gesamten Text in der App zu ändern[5].
3. `AndroidManifest.xml`
   Diese Datei enthält einige Informationen zu unserer App, wie etwa den App-Namen oder das App-Symbol.
4. `MainActivity.java`
   Die für uns wichtigste Datei. Hier schreiben wir Code, um auf Nutzereingaben zu reagieren, Berechnungen oder andere Aktionen auzuführen und die GUI entsprechend anzupassen.

3.4. Der GUI-Editor

Ein zentrales Element einer Android-Application ist ihre **GUI**[6]. Sie ist für die Darstellung aller graphischen Elemente, sowie für das Entgegennehmen von Benutzereingaben verantwortlich. Alle Informationen, die Android Studio für das anfängliche Design unserer GUI benötigt, werden in der Datei `activity_main.xml` gespeichert. Der Ort, an

[4] mehr dazu in Abschnitt 4.2
[5] Auch dazu mehr in Abschnitt 4.2
[6] GUI, engl. für graphical user interface, bedeutet übersetzt graphische Benutzeroberfläche

dem die Datei zu finden ist, ist in Abb. 14 dargestellt. Mit einem Doppelklick auf die Datei öffnet sich der GUI-Editor und die Datei lässt sich bearbeiten. Dies ist in Abb. 15 dargestellt.

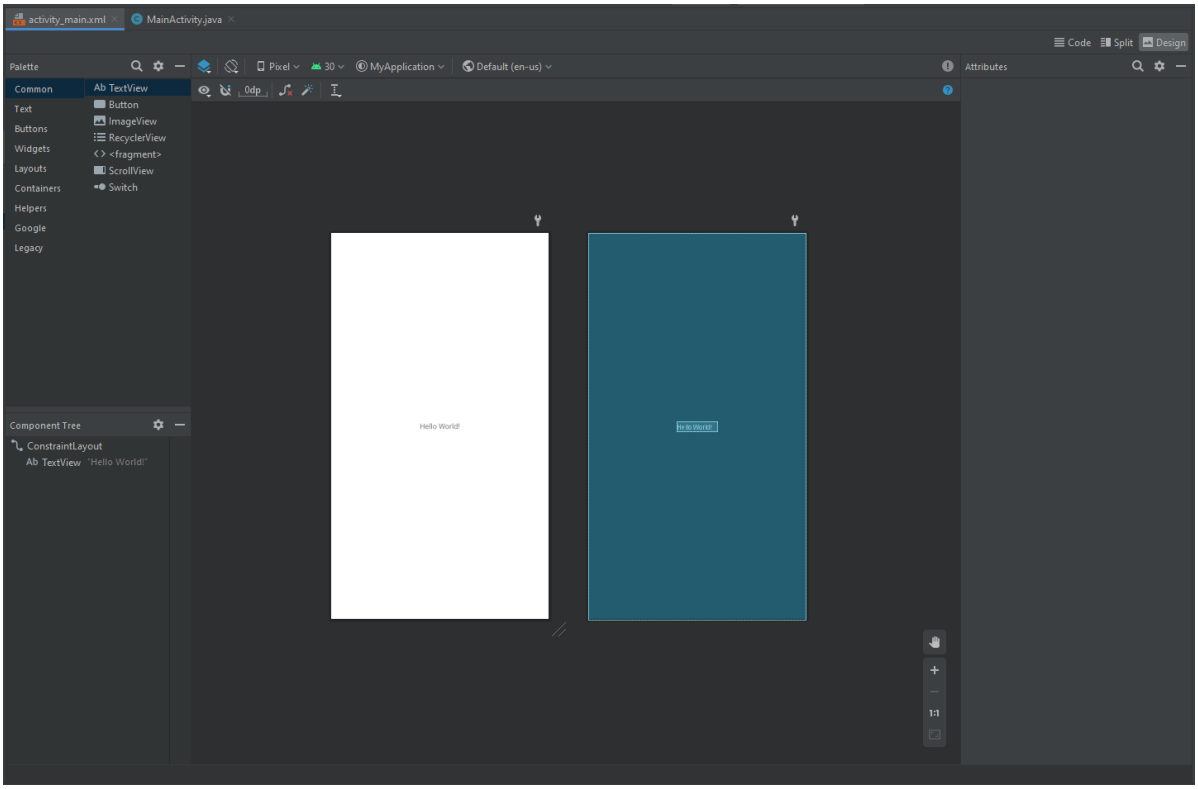


Abbildung 15.: GUI-Editor am Beispiel der Datei *activity_main.xml*

Sollte der Bildschirm nach dem Doppelklick nicht wie in der Abbildung aussehen, sondern nur ein Text angezeigt werden, kann das an der Darstellung durch Android-Studio liegen. In diesem Fall können wir durch Anklicken der Schaltfläche `Design` (oben rechts in der Abb. 15) die Darstellungsform ändern. Anschließend müsste die Datei wie in der Abbildung dargestellt werden.

Der Umgang mit dem GUI-Editor wird in Kapitel 4.2 genauer beschrieben. Hier werden wir auch Änderungen an der Datei vornehmen.

3.5. Der Programmcode

Noch wichtiger als die GUI ist in einer Android-App der **Programmcode**. Beim Programmcode handelt es sich um das eigentliche Programm, d.h. die Befehlsfolge die vom

Betriebssystem[7] des Smartphones ausgeführt wird.

Im vorliegenden Beispiel ist für den Programmcode die Datei `MainActivity.java` zuständig. Bei einem Doppelklick auf die Datei[8] öffnet sich ein Texteditor innerhalb der Entwicklungsoberfläche[9]. In der Datei ist das folgende Programm zu finden.

```
1  package hoelz.benno.myapplication;
2
3  import androidx.appcompat.app.AppCompatActivity;
4
5  import android.os.Bundle;
6
7  // Klasse mit dem Hauptprogramm (MainActivity)
8  public class MainActivity extends AppCompatActivity {
9
10     @Override
11     protected void onCreate(Bundle savedInstanceState) { // Methodenkopf
12         // Wird beim Start der App ausgefuehrt (Methodenrumpf)
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_main);
15     }
16 }
```

Im Programm erkennen wir eine klare Struktur. Der wichtigste Teil der Datei beginnt in Zeile 8. Hier wird eine Java-Klasse, wie wir sie bereits aus Kapitel 1.1 kennen, erstellt. Sie enthält auch bei Android-Apps eine Art Main-Methode[10]. Diese Art Main-Methode trägt den Methodenkopf `protected void onCreate(Bundle savedInstanceState)` und wird beim Start der Applikation aufgerufen. D.h. die Befehlsfolge in ihrem Rumpf[11] wird von oben nach unten ausgeführt.

Die `onCreate`-Methode enthält zu Beginn unserer App zwei Befehle. Der erste Befehl ruft eine Methode weiter oben in der Vererbungshierarchie auf [12]. Deutlich interessanter ist aber der zweite Befehl. Dieser generiert die Graphische Oberfläche nach den Vorgaben aus der Datei `activity_main.xml`. Fehlt dieser Befehl ist es egal was in der Datei `activity_main.xml` gespeichert ist, der Bildschirm bleibt weitestgehend leer.

[7] Android - Das Betriebssystem bezeichnet die Schnittstelle zwischen Hardware (Prozessor, Display, etc) und Software (Programmcode).
[8] Man findet sie in der Projektstruktur in Abb. 14
[9] IDE, Android-Studio
[10] vgl. Seite 5 Kapitel 1.1; In der Main-Methode befindet sich in den meisten gängigen Programmiersprachen der Programmstart
[11] Bereich zwischen den beiden geschwungnen Klammmern { }
[12] Was das im Detail bedeutet ist für den Moment irrelevant, wichtig ist nur das die Befehlszeile in dieser Form existiert

4. Die erste App - Hello World

4.1. App installieren

Wenn wir ein neues Projekt anlegen, ist dieses nicht ganz leer. Die GUI enthält bereits einen Schriftzug: *Hello World!* Wir haben also schon eine einfache erste App, die nichts anderes tut als diesen Text anzuzeigen.

4.1.1. Installation auf dem eigenen Android-Smartphone

Um die App zu testen, können wir sie auf unserem Smartphone installieren. Das funktioniert folgendermaßen:

1. Zunächst müssen wir auf dem Smartphone **USB-Debugging** in den **Entwickleroptionen** aktivieren. Die Entwickleroptionen sind jedoch nicht standardmäßig sichtbar und müssen deshalb zuerst freigeschaltet werden. Wir gehen dazu wieder in den Einstellungen unter `System > Über das Telefon`[1] und tippen dort so oft auf die **Build-Nummer**, bis eine Nachricht erscheint, die uns signalisiert dass wir nun ein Entwickler sind. Die Entwickleroptionen finden wir nun unter `System` und können dort das USB-Debugging aktivieren.
2. Als nächstes verbinden wir unser Smartphone via USB mit dem PC. Dabei müssen wir darauf achten, dass am Smartphone der USB-Modus auf `Dateien übertragen` o. Ä.[2] gestellt ist und das Gerät entsperrt ist.
3. Unser Smartphone sollte nun rechts oben in Android Studio angezeigt werden bzw. aus der Liste ausgewählt werden können. Mit einem Klick auf `Run`, also den grünen Pfeil, wird die App auf dem Smartphone installiert und sollte sich im Anschluss automatisch öffnen. Das kann einen Moment dauern.

[1] Die Bezeichnungen der Menüpunkte können je nach Hersteller unterschiedlich sein!
[2] Bei manchen Geräten auch **MIDI eingeben** o. Ä.

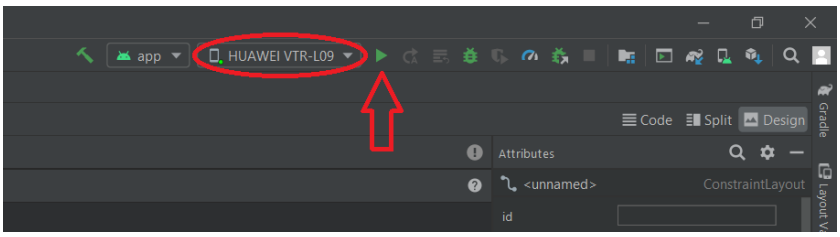


Abbildung 16.: Hier sollte unser Smartphone angezeigt werden

4.1.2. Installation auf einem Virtual Device

Eine Alternative zur Installation auf dem eigenen Android-Smartphone ist die Installation auf einem virtuellen Gerät. Android Studio bietet uns hierzu die Möglichkeit, ein Smartphone zu simulieren und die App darauf zu installieren. Im Idealfall gibt es bereits ein voreingestelltes Virtual Device. Ansonsten lohnt es sich, sich mit dem **AVD Manager**[3] zu befassen.

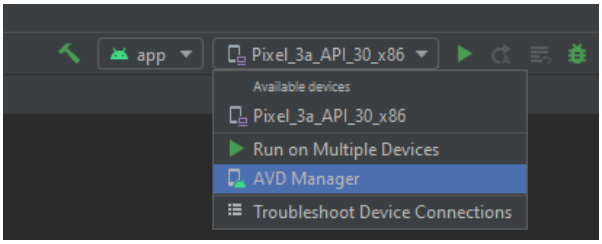


Abbildung 17.: Virtual Device

4.2. App modifizieren: GUI-Editor

Nun ist eine App die einfach nur *HelloWorld* anzeigt, noch etwas langweilig. Vielleicht würden wir ja gerne selber einen Text festlegen, der angezeigt werden soll oder die Schriftgröße ändern. Die einfachste Art und Weise das zu tun ist der GUI-Editor. Ein GUI-Element, also z. B. das Textfeld, kann durch Anklicken ausgewählt und anschließend bearbeitet werden. Schauen wir uns dafür den Bereich `Attributes` auf der rechten Seite an:[4]

[3] Android Virtual Device Manager; Ist für die Verwaltung der virtuellen (simulierten) Geräte verantwortlich

[4] Sollte dieses Menü nicht angezeigt werden, kann es durch einen Klick geöffnet werden

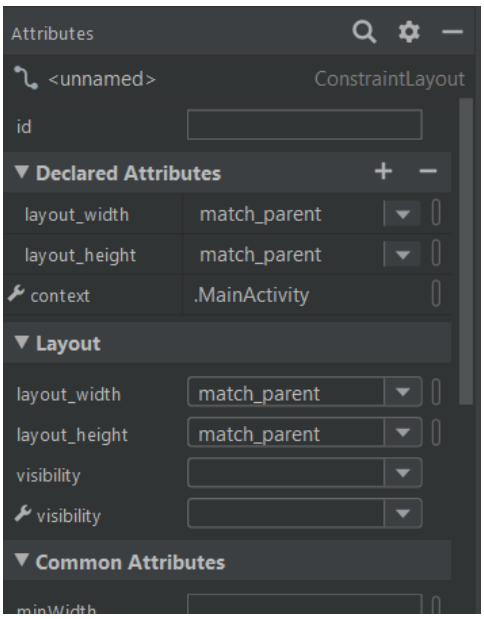


Abbildung 18.: Der Attribut-Manager (bei ausgewähltem Textfeld)

Zunächst tragen wir hier in das Feld `id` eine von uns gewählte ID ein, also einen eindeutigen Namen für das zugehörige GUI-Element. Diese ID wird später wichtig, wenn wir im Programmcode auf ein GUI-Element zugreifen bzw. dieses verändern wollen. Da es sich hier um eine TextView handelt, kann besipielsweise `tvHello` als ID verwendet werden. Es empfiehlt sich, bei den IDs immer ein festes Schema zu verwenden (also z.B. `tvXY` für TextViews oder `btnXY` für Buttons). Das macht den Code übersichtlicher und auch andere finden sich schneller wieder zurecht.

An dieser Stelle können wir auch direkt den Text unserer TextView verändern. Dazu müssen wir den von uns gewünschten Text im Feld `text` eintragen.[5]

Der Text ist der TextView nun direkt zugewiesen und müsste bei Änderungen der App (z. B. Umstellen der Sprache) manuell geändert werden. Um das zu umgehen, können wir die Datei `strings.xml` verwenden, die wir in Abschnitt 3.3 kennengelernt haben. Hier können wir unseren Text eintragen und dann im Attribut-Manager auf den Eintrag in der Datei verweisen. So müsste etwa bei einer Sprachänderung nur die XML-Datei ausgetauscht werden. Dazu öffnen wir die `strings.xml`-Datei und fügen zwischen den beiden `resources`-Tags eine neue Zeile ein:

```
<string name="tvHelloText">Hallo Welt!</string>
```

Über das Attribut `name` können wir hierbei unserem Eintrag einen Namen zuweisen, also hier beispielsweise `tvHelloText`. Auch ist es hilfreich, sich ein festes Schema für die Namen anzueignen. Anschließend können wir im Attribut-Manager auf unseren neu angelegten String verweisen, indem wir im `text`-feld folgendes eingeben: `@string/tvHelloText`

[5]Felder lassen sich auch über das Suchsymbol oben rechts im Attribut-Manager finden.

Wir können den String auch direkt auswählen, indem wir auf den Balken rechts im `text`-Feld klicken und den String aus der Liste auswählen.

4.3. App modifizieren: Main-Klasse

Der in Abschnitt 4.2 genannte Weg, den Text einer TextView anzupassen, eignet sich z. B. für Überschriften oder andere Textelemente, die sich nie während der Nutzung der App verändern. Wollen wir jedoch beispielsweise den Text einer TextView von *Hello World!* auf *Hallo Welt!* ändern, wenn der Benutzer den Button drückt, so müssen wir dies in der Main-Klasse in Form von **Java-Code** tun.

Dazu öffnen wir zunächst die Datei `MainActivity.java`, die im Abschnitt 3.5 beschrieben wurde. Dort wollen wir in der `onCreate`-Methode, die beim Start der App ausgeführt wird, unsere Änderungen vornehmen. Zunächst müssen wir aber die TextView, die ja bis jetzt nur in der GUI existiert, im Java-Code **referenzieren**, indem wir ein Java-Objekt anlegen und mithilfe der festgelegten ID auf die TextView verweisen.

Wir legen dazu zuerst ein entsprechendes Objekt in der Klasse `MainActivity` an, indem wir folgenden Code noch vor der `onCreate`-Methode einfügen:

```
1 private TextView tvHello;
```

Wir erstellen also ein `TextView`-Objekt namens `tvHello`, dem wir aber noch kein GUI-Element zuweisen. Das passiert im nächsten Schritt durch das Einfügen folgender Zeile in der `onCreate`-Methode (nach `setContentview(...);`) einfügen:

```
1 tvHello = findViewById(R.id.tvHello);
```

Damit weisen wir unserem `TextView`-Objekt die TextView mit der ID `tvHello` zu[6] und können im nächsten Schritt beispielsweise den aktuellen Text verändern:

```
1 tvHello.setText("Hallo andere Welt!");
```

Diese Veränderung wird direkt beim App-Start ausgeführt und sollte damit sofort sichtbar sein. Später nutzen wir Buttons und andere Eingabemethoden, um solche Veränderungen auszulösen.

[6] **Achtung:** Hier muss zwischen der ID tvHello und dem Namen des Java-Objektes unterschieden werden – diese sind unabhängig voneinander und können somit auch unterschiedlich sein!

5. Eventbasierte Programmierung - Buttons & Co.

5.1. Eventhandling – was ist das?

Bisher haben wir nur gelernt, wie wir Aktionen direkt beim Start der App ausführen können: Wir schreiben die entsprechenden Befehle einfach in die `onCreate`-Methode. Das allein reicht natürlich nicht aus, um eine sinnvolle App zu programmieren. Wichtig für eine gute App ist es, dass die App auf Benutzeraueingaben – beispielsweise auf die Eingabe von Daten oder das Drücken eines Buttons – reagiert.

Um auf Benutzereingaben reagieren zu können, wird unter Android das sogenannte **Eventhandling** verwendet. Im Wesentlichen wird dabei im Hintergrund auf ein bestimmtes Ereignis gewartet und bei Bedarf entsprechend darauf reagiert, anstatt es ständig aktiv danach zu fragen, ob das Ereignis eingetreten ist. Nehmen wir z. B. einen Button: Die einfachste Art und Weise, diesen abzufragen, wäre es, in einer Schleife immer wieder und wieder zu prüfen, ob dieser gerade gedrückt ist. Das benötigt aber sehr viel Rechenzeit und wir können in diesem Moment keine anderen Eingaben abfragen oder Berechnungen durchführen. Im Gegenzug würden wir diese Eingabe vielleicht auch verpassen, wenn wir gerade eine andere Berechnung durchführen, es muss also eine bessere Lösung geben.

Wir verwenden deshalb einen **Eventlistener**: Dieser beobachtet ständig unser GUI-Objekt und führt eine vorgegebene Aktion aus, wenn das gewünschte Ereignis – also beispielsweise ein Klick auf den Button – eintritt.

5.2. Ein Beispiel: Buttons

5.2.1. Einen neuen Button erstellen

Zunächst einmal erstellen wir in unserer GUI einen Button. Wir ziehen dazu im GUI-Editor einen Button aus der `Palette` unter `Common` auf unsere GUI unter die bereits vorhandene TextView:

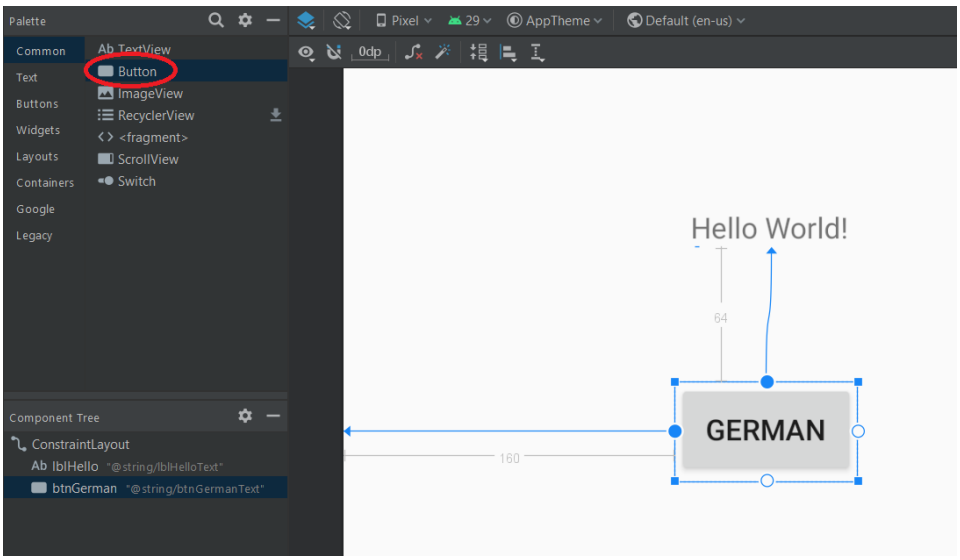


Abbildung 19.: Einfügen eines Buttons

Um die Position des Buttons festzulegen, müssen wir noch die **Constraints**[1] des GUI-Elements anpassen – es also auf einer Seite in x- und y-Richtung verankern. Wir klicken dafür mit der Maus auf einen der Punkte am Rand des Buttons und ziehen von dort einen Pfeil zu dem Objekt, an dem der Button ausgerichtet werden soll. In diesem Fall ziehen wir den oberen Pfeil auf den unteren Punkt der TextView und den linken Pfeil an den linken Rand des Bildschirms, sodass diese einrasten. Nun können wir den Button beliebig unterhalb der TextView auf unserer GUI platzieren.

Anschließend vergeben wir noch wie in Abschnitt 4.2 gezeigt eine ID (hier `btnGerman`) sowie einen Text (hier *German*) für den Button. Wir wollen nämlich auf Knopfdruck den Text der TextView von *Hello World!* in *Hallo Welt!* ändern können.

5.2.2. Eventlistener erstellen in Java

Nun sind wir, was die GUI betrifft, fertig und können zu programmieren beginnen. Wir wechseln also wieder in die Klasse `MainActivity` und legen dort ein `Button`-Objekt an, wie bereits am Beispiel der TextView gezeigt:

```
1 private Button btnGerman;
```

Dieses Objekt belegen wir in der `onCreate`-Methode wieder mit einer Referenz, diesmal auf den Button:

```
1 btnGerman = findViewById(R.id.btnGerman);
```

[1] Constraints bezeichnen den Abstand zur Ober- bzw. Unterkante des Bildschirms und zur rechten bzw. linken Bildschirmseite.

Nun können wir den Eventlistener erstellen, der auf das Antippen des Buttons wartet. Das passiert in der `onCreate`-Methode wie folgt:[2]

```
btnGerman.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

Wir haben nun eine Methode `onClick` erzeugt, in der wir definieren können was beim Drücken des Buttons passieren soll. Der `OnClickListener` ist dabei eine spezielle Art von Eventlistener, die auf das Drücken von Buttons reagieren kann. Wir Wollen, dass sich der Text der TextView wie oben beschrieben ändert – außerdem wollen wir noch den Button deaktivieren, sodass er nur einmal gedrückt werden kann. Also schreiben wir folgendes:[3]

```
btnGerman.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        tvHello.setText("Hallo Welt!");
        btnGerman.setEnabled(false);
    }
});
```

Wenn wir die App nun ausführen und auf den Button drücken, sollte sich der Text der TextView ändern und der Button deaktiviert werden.

Der Befehl `tvHello.setText("Hallo Welt!");` ändert dabei den Text der TextView, der Befehl `btnGerman.setEnabled(false);` deaktiviert den Button für weitere Eingaben.

5.3. Weiteres Beispiel: Eingabefelder

Diese Art von Eventhandling gilt natürlich nicht nur für Buttons: Es existieren für jede Art von GUI-Objekt verschiedene Arten von Eventlisteners, um beispielsweise auf die Auswahl eines Eintrags aus einem Dropdown-Menü reagieren zu können. Nun wollen wir uns noch das GUI-Element **PlainText** anschauen, dabei handelt es sich um ein Eingabefeld, über das wir Text eingeben können.

5.3.1. PlainText in der GUI

Wir wollen unserer App nun noch eine Möglichkeit hinzufügen, einen eigenen Text einzugeben. Dieser soll dann auf Knopfdruck auf der TextView angezeigt werden. Wir gehen

[2] Den Großteil dieser Eingabe kann man über die Autovervollständigung erreichen, es muss nicht jedes Mal alles abgetippt werden.

[3] Wir schreiben diesen Code unter der Annahme, dass das HelloWorld-Programm aus Kapitel 4 bereits existiert, die Zeile mit dem setText-Aufruf kann gelöscht werden.

dazu zunächst wieder in den GUI-Editor und ziehen einen neuen `PlainText` auf unsere GUI. Diesen finden wir ebenfalls unter der Kategorie `Common`.

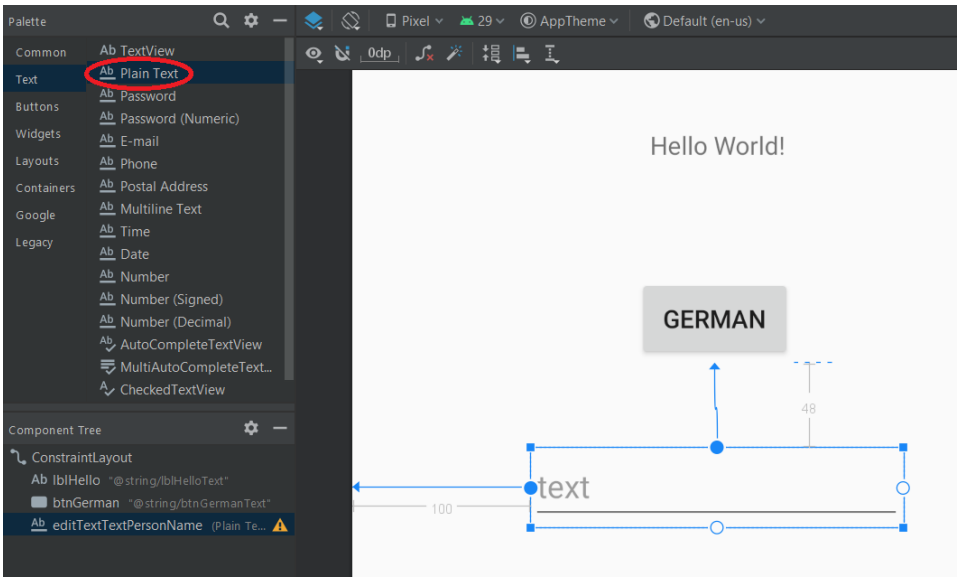


Abbildung 20.: Einfügen eines Plaintext-Eingabefelds

Auch für dieses GUI-Element können wir eine ID (hier `edInput`) sowie ein Text vergeben. Anstatt eines Textes empfiehlt es sich hier aber, einen `hint` (hier *text*) anzugeben. Dabei handelt es sich um einen grau dargestellten Hinweis, der verschwindet, sobald der Nutzer etwas in das Textfeld eingibt.

5.3.2. PlainText auslesen

Nun können wir wieder in die Klasse `MainActivity` wechseln und dort den nötigen Code hinzufügen um das Eingabefeld auf Knopfdruck auszulesen. Wir erstellen wieder zuerst ein Java Objekt:[4]

```
1 private EditText edInput;
```

Dieses können wir nun wieder in der `onCreate`-Methode belegen:

```
1 edInput = findViewById(R.id.edInput);
```

Unser Ziel ist es jetzt, den Text des Eingabefelds auszulesen und in die TextView zu schreiben, sobald der Button gedrückt wird. Wir schreiben also in die `onClick`-Methode des `OnClickListeners`, den wir vorher angelegt haben:

```
1 String input = edInput.getText().toString();
2 tvHello.setText(input);
```

[4] **Achtung:** In Java gibt es kein PlainText-Objekt, wir verwenden deshalb das EditText-Objekt!

Wir lesen also den Inhalt des Eingabefelds und speichern diesen in einer String-Variable `input`. Hierbei müssen wir den Inhalt mithilfe von `toString()` zuerst noch in einen String konvertieren, da die `getText`-Methode nicht direkt einen String zurückliefert. Diesen String verwenden wir anschließend, um mithilfe der `setText`-Methode den Inhalt der TextView entsprechend anzupassen. Das Deaktivieren des Buttons aus Abschnitt 5.2.2 können wir hier auch weglassen, um unsere App besser testen zu können. Auch den Text des Buttons können wir nun natürlich auf etwas Passenderes ändern, z. B. *OK* oder *Text anzeigen*.

Wenn wir nun die App starten, etwas in das Textfeld eingeben und auf den Button drücken, so sollte der eingegebene Text auf der TextView erscheinen.

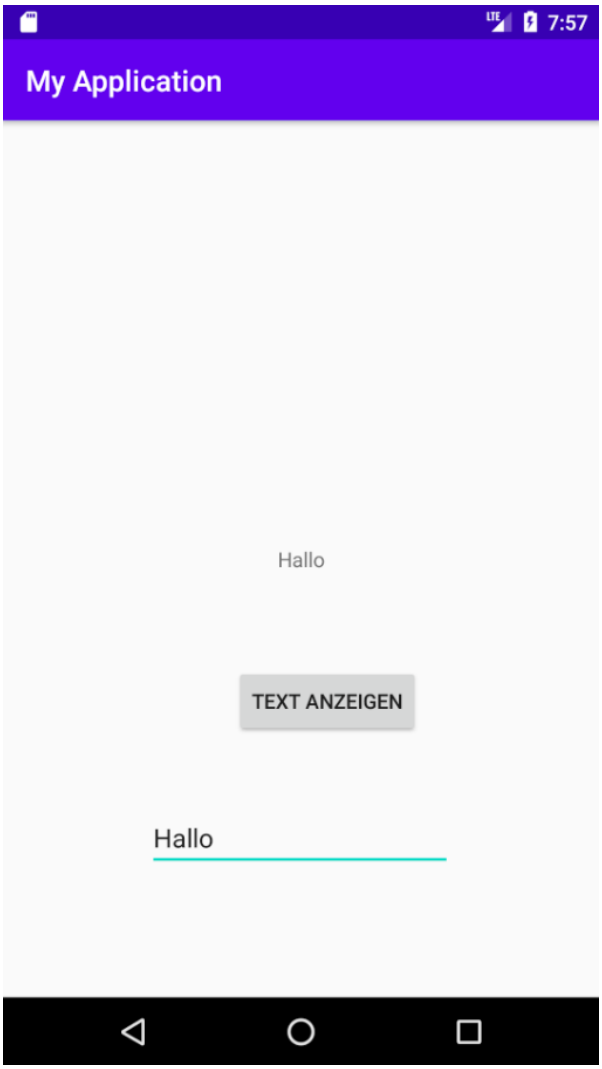


Abbildung 21.: Die fertige App

6. Arbeiten mit Variablen - Counter

Im letzten Kapitel haben wir bereits eine String-Variable erstellt, um darin den eingegebenen Text zwischenzuspeichern. Nun wollen wir uns etwas genauer ansehen, was man mit Variablen noch alles machen kann. Dazu programmieren wir eine kleine Counter-App, also einen Zähler, den wir über drei Tasten hoch- bzw. herunterzählen sowie zurücksetzen können.

6.1. GUI erstellen

Beginnen wir also zunächst wieder mit dem Design der GUI. Wir erstellen dazu ein neues Projekt und öffnen die Datei `activity_main.xml` im GUI-Editor. Die bereits vorhandene TextView können wir direkt für unsere Zähleranzeige benutzen, sollte diese noch nicht existieren, fügen wir aus der Palette einfach eine TextView hinzu und erstellen die nötigen Constraints. Dieser TextView geben wir hier die ID `tvCounter`, um sie später in Java referenzieren zu können und setzen ihren Text auf *0*.

Nun müssen wir noch die drei nötigen Buttons für **Plus**, **Minus** und **Reset** auf die GUI ziehen und die passenden Constraints erstellen. Als IDs für die Buttons verwenden wir hier `btnPlus`, `btnMinus` und `btnReset`. Auch die Texte der Buttons passen wir entsprechend an.

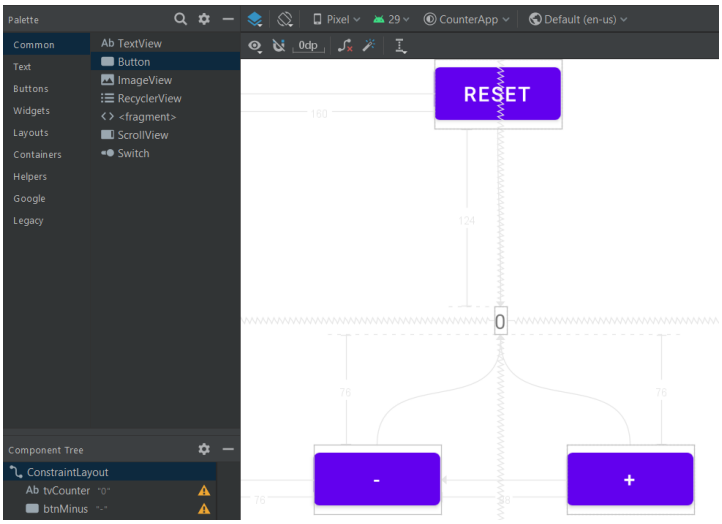


Abbildung 22.: Hinzufügen der Counter-Buttons

6.2. Der Programmcode

Nun, da wir mit der GUI fertig sind, können wir uns wieder ans Programmieren machen. In der `MainActivity`-Klasse legen wir zunächst wieder die Java-Objekte für unsere GUI-Elemente an. Außerdem erstellen wir hier eine Integer-Variable, in der später der aktuelle Wert des Zählers gespeichert werden soll:

```
1 private TextView tvCounter;
2 private Button btnMinus;
3 private Button btnPlus;
4 private Button btnReset;
5
6 private int counter; // Counter-Variable
```

In der `onCreate`-Methode belegen wir die GUI-Objekte wieder entsprechend mit ihren Referenzen uns setzen die Counter-Variable auf 0:

```
1 tvCounter = findViewById(R.id.tvCounter);
2 btnMinus = findViewById(R.id.btnMinus);
3 btnPlus = findViewById(R.id.btnPlus);
4 btnReset = findViewById(R.id.btnReset);
5
6 counter = 0;
```

Jetzt benötigen wir für jeden der Buttons einen OnClickListener, um auf Eingaben reagieren zu können. Beginnen wir mit dem Button `btnMinus`:

```
1 btnMinus.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         if(counter > 0){
5             counter--;
6             tvCounter.setText(""+counter);
7         }
8     }
9 });
```

Wenn dieser Button gedrückt wird, müssen wir zunächst einmal überprüfen, ob wir nicht schon bei 0 sind, da wir keine negativen Zahlen für den Counter zulassen wollen. Wir möchten also den Counter nur herunterzählen, wenn er größer als 0 ist. Dies überprüfen wir mit der if-Bedingung.[1] Sollte der Counter größer als 0 sein, zählen wir ihn zunächst mit `counter--` herunter. Das ist eine abkürzende Schreibweise für `counter = counter - 1`, die in Java gerne verwendet wird. Anschließend können wir den Text der TextView anpassen, dabei müssen wir die Counter-Variable allerdings in einen String konvertieren, da es sich ja um einen Integer handelt. Dies geht am einfachsten über das Anhängen der Zahl an einen leeren String mit `""+counter`.

Ensprechend können wir auch den OnClickListener für den Button `btnPlus` erstellen:

```
1 btnPlus.setOnClickListener(new View.OnClickListener() {
2     @Override
3     public void onClick(View v) {
4         if(counter < Integer.MAX_VALUE){
5             counter++;
6             tvCounter.setText(""+counter);
```

[1] vgl. Kapitel 1.3.1 auf Seite 7

```
        }
    }
});
```

Hier überprüfen wir, ob der Counter den Wertebereich des Integers nicht übersteigt – wiederum in einer if-Bedingung. `Integer.MAX_VALUE` steht dabei für den größten Wert, den eine Integer-Variable annehmen kann. Dieser ist zwar sehr hoch und kann somit praktisch unmöglich durch Klicken auf den Button erreicht werden,[2] der Vollständigkeit und Sicherheit halber fügen wir diese Bedingung jedoch trotzdem ein. Das Hochzählen funktioniert dann analog über `counter++` und auch beim Anpassen der TextView ändert sich nichts.

Zum Schluss noch der Code für den Button `btnReset`:

```
btnReset.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        counter = 0;
        tvCounter.setText(""+counter);
    }
});
```

Das ist der einfachste Button. Er setzt einfach nur den Wert der Counter-Variable auf 0 und passt die TextView entsprechend an.

Damit sind wir auch schon fertig und können unsere App testen!

[2] Genau genommen müssten wir hier bis $2^{31} - 1$ zählen – also sehr weit!

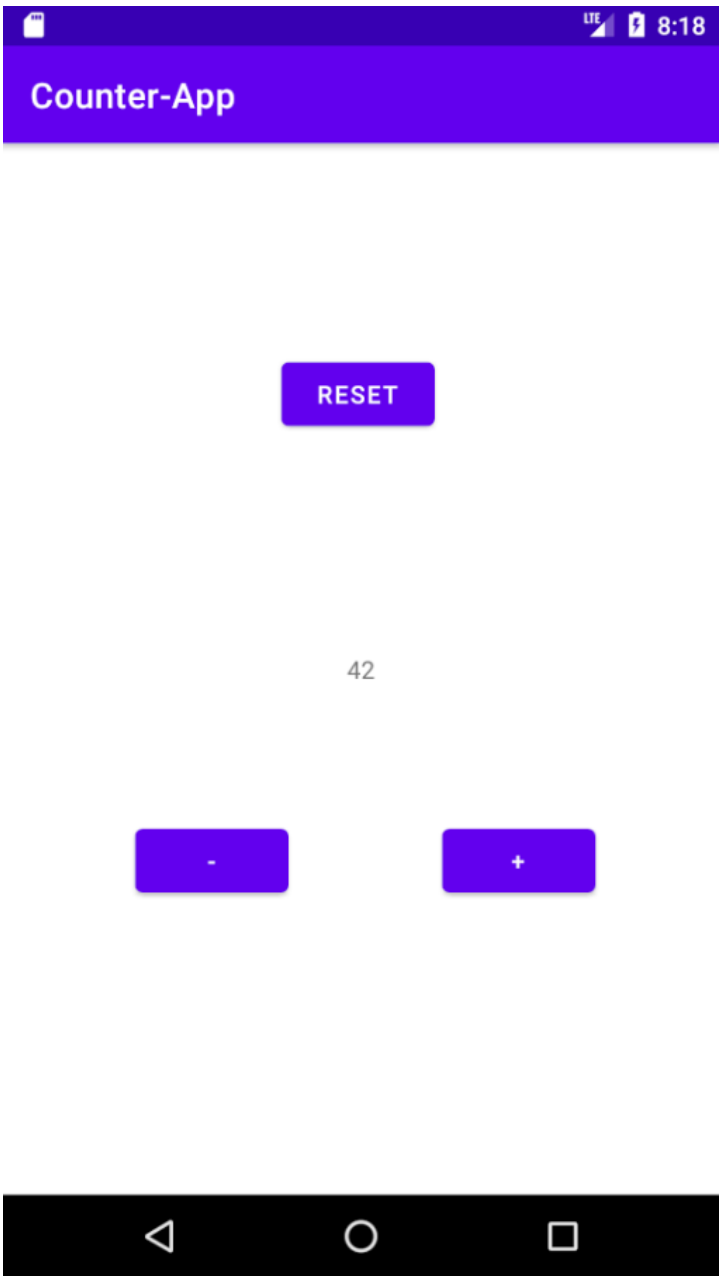


Abbildung 23.: Die fertige Counter-App

A. Anhang

A.1. Java und Objektorientierung

A.1.1. Tabelle weiterer Datentypen

Datentyp	Abk.	Beispiel	Wertebereich
Byte	byte	`byte b = 8;`	-128 bis 127
Short	short	`short s = 129;`	2^{15} bis $2^{15} - 1$
Integer	int	`int i = 0;`	2^{31} bis $2^{31} - 1$
Long	long	`long l = 12345678`	2^{63} bis $2^{63} - 1$
Char	char	`char c = 'l'`	meist UTF-8
Boolean	boolean	`boolean bool = true;`	true , false
String	String	`String str = "I bims 1 String"`	Verkettung von Chars

Tabelle 1.: Datentypen Übersicht

Literatur

[1] Jan-Dirk. *Programmiersprache für die Entwicklung von Android Apps*. 23. März 2018. URL: https://www.it-talents.de/blog/it-talents/welche-programmiersprache-fuer-die-entwicklung-von-android-apps.

[2] Matthias Tichy. *Programmierung von Systemen, Skript SoSe2020*. Universität Ulm. 4. Nov. 2020.

[3] wiki.ubuntuusers.de. *Android Studio*. 3. Aug. 2020. URL: https://wiki.ubuntuusers.de/Android_Studio/.