

# CowCounter-App

## Skript 2016

Konfigurations- und Schulungsunterlagen

Schulung:	<a href="#">Didaktische Ansätze zur Android-Programmierung</a>
Referent:	<a href="#">Christine Janischek</a>

Stand: 7. Jun 2016



© Christine Janischek

## Inhaltsverzeichnis

1 Allgemeines.....	3
2 Das Projekt Cow Counter.....	5
2.1 Überblick.....	5
2.2 Grundlagen: Projekt erzeugen.....	6
2.3 View: Layouts, Komponenten & XML für die Benutzeroberfläche.....	10
2.4 Modell: Implementierung der Fachklassen für die Datenhaltung.....	28
2.5 Controller: Ereignisse steuern.....	33

## 1 Allgemeines



Das Skript schildert den Umgang mit Android Studio anhand von konkreten Beispielen die unter Umständen auch in den Unterricht im Fachbereich Wirtschaftsinformatik respektive im Fachbereich Informatik einbetten lassen.

Aktuelle Versionen des Skriptes selbst und die im Skript behandelten Quellcodes können Sie online herunterladen und testen:

Skript & Sources für die Projekte (für Fortgeschrittene):

→ [Alle Arbeitsmaterialien in Chrissis Edublog herunterladen](#)



Für alle Inhalte gilt natürlich das Urheberrecht. Ich selber achte auch darauf. Um Details zur Creative-Commons-Lizenz für die von mir selbst verfassten Texte und Quellcodes zu erhalten, klicken Sie links auf das CC-BY-NC-SA-Logo. Für Ergänzungs- und/oder Verbesserungsvorschläge schreiben Sie mir bitte eine E-Mail: [cjanischek@gmx.de](mailto:cjanischek@gmx.de)

Weitere Skripte und Sources online:

[Einführung in die Programmierung von Android Apps anhand klassischer Unterrichtsbeispiele](#)

[Fortgeschrittene Apps mit Android Studio erstellen](#)

[Android Apps erstellen](#)

[Java Programmieren im Unterricht](#)

[Java-E-Learning zum Unterricht](#)

[Objektorientierte Sytementwicklung in Java](#)

[Dynamische Webseiten mit PHP \(objektorientiert\) programmieren](#)

[Webprogrammierung im Unterricht](#)

[Entwickeln mit Javascript Framework \(jQuery, JQuery mobile\)](#)

[Einführung in PHP und die WordPress-Theme-Entwicklung](#)

[Relationale Datenbanken](#)

Alle Quellangaben wurden nach bestem Gewissen genannt und aufgeführt. Permanent begleitende Literatur waren:

[BUC01]

Buchalka, Tim, "Master Android 6.0 Marshmallow Apps Development Using Java", timbuchalka.com, 2016, Udemy Course

[KUE01]

Künneht, Thomas, "Android 5 – Apps entwickeln mit Android Studio", 978-3-8362-2665-3, 2015, Galileo Computing

[WAC00]

Wagner, Chris, "Das Android SQLite Datenbank Tutorial", <http://www.programmierenlernenhq.de/android-sqlite-datenbank-tutorial/>, 2016, programmierenlernenhq.de, zuletzt getestet am 09.04.2016

[FLE00]

Flowers, Eric, "WeatherIcons", <https://github.com/erikflowers/weather-icons/tree/master/font>, 2016, <http://www.helloerik.com>, zuletzt getestet am 26.04.2016

[HAA00]

Hathibelagal, Ashraff „Create a Weather App on Android“, <http://code.tutsplus.com/tutorials/create-a-weather-app-on-android--cms-21587>, zuletzt getestet am 26.04.2016

[AZF00]

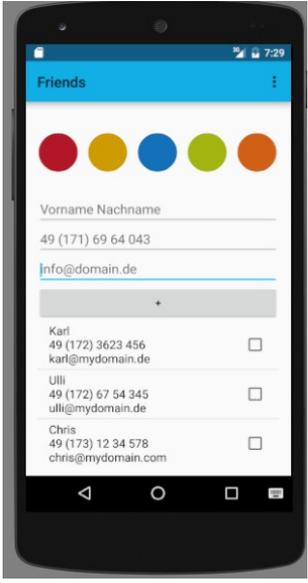
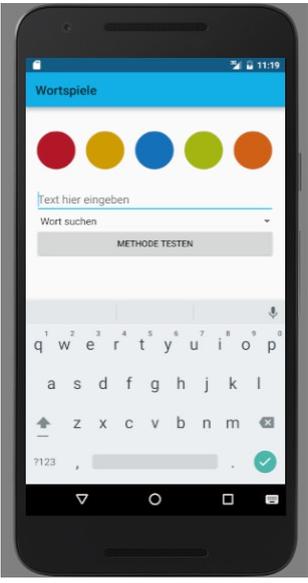
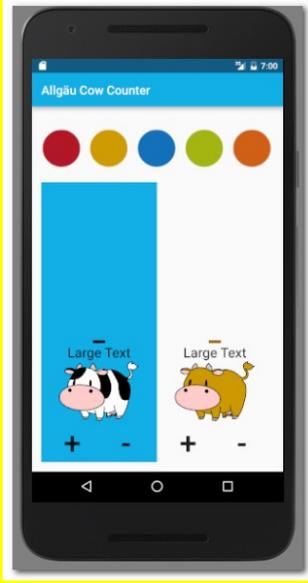
Azzola, Francesco „Android: Build real weather app: JSON, HTTP and Openweathermap“, <https://www.javacodegeeks.com/2013/06/android-build-real-weather-app-json-http-and-openweathermap.html>, 2013, zuletzt getestet am 30.04.2016

## 2 Das Projekt Cow Counter

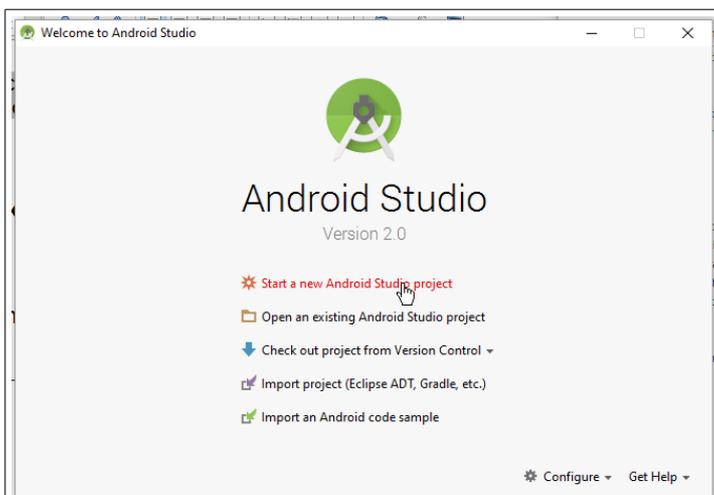
### 2.1 Überblick

#### Cow Counter App:

Das Projekt soll an einer Benutzeroberfläche zeigen, auf welche Weise Layouts für die Darstellung und Manipulation eines einfachen Säulendiagramms, genutzt werden können. Die Cow Counter App ermöglicht zudem die Steuerung zweier Zählervariablen (Inkrementieren, Dekrementieren), um die Anzahl brauner und schwarz-weißer Kühe über die Benutzeroberfläche der Anwendung zu zählen.

Weather App	Friends App	Wortspiele App	Cow-Counter App
			
			
<p>Tags: OpenWeatherMap, http, Netzwerk, JSONObject, Fragment, Schrift, Exceptions, Fehlerbehandlung, Thread, Dialog</p>	<p>Tags: Datenbankzugriff, SQLite, ListView, Menüs, Dialog</p>	<p>Tags: Stringverarbeitung, Kontrollstrukturen, Spinner, Dialoge, Fallunterscheidungen, Schleifen, Algorithmen</p>	<p>Tags: Zähler, Inkrementieren, Dekrementieren, Layouts, Säulendiagramm</p>

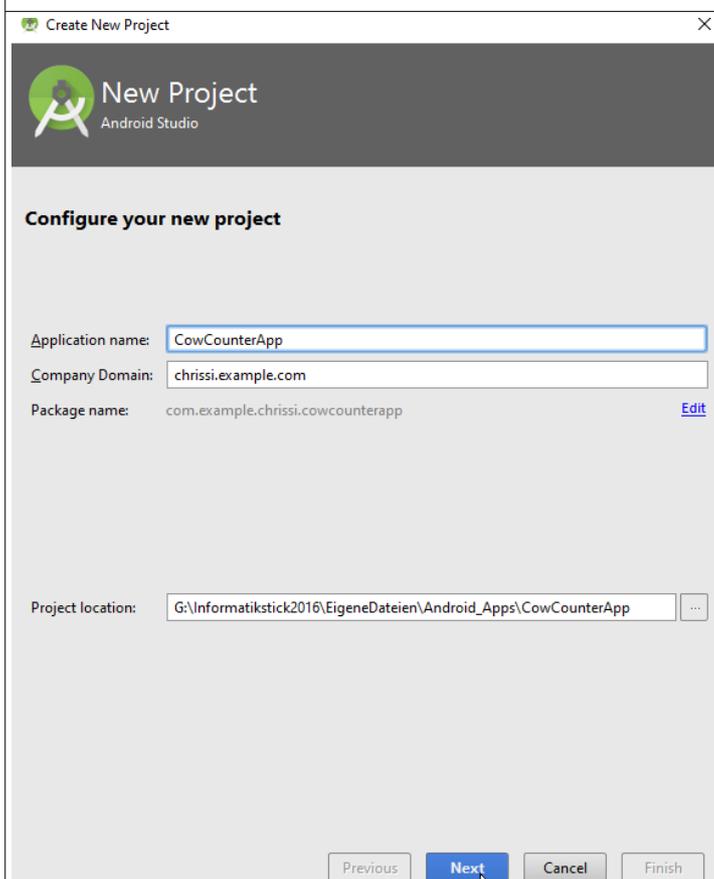
## 2.2 Grundlagen: Projekt erzeugen



### Ein Neues Projekt erzeugen.

Der angezeigte Dialog öffnet sich für den Fall, dass zuvor alle Projekte geschlossen wurden bzw. die Entwicklungsumgebung erstmals geöffnet wird.

Um ein neues Projekt zu erzeugen, wählen Sie im Quick Start-Menü die Option → Start a new Android Studio project.



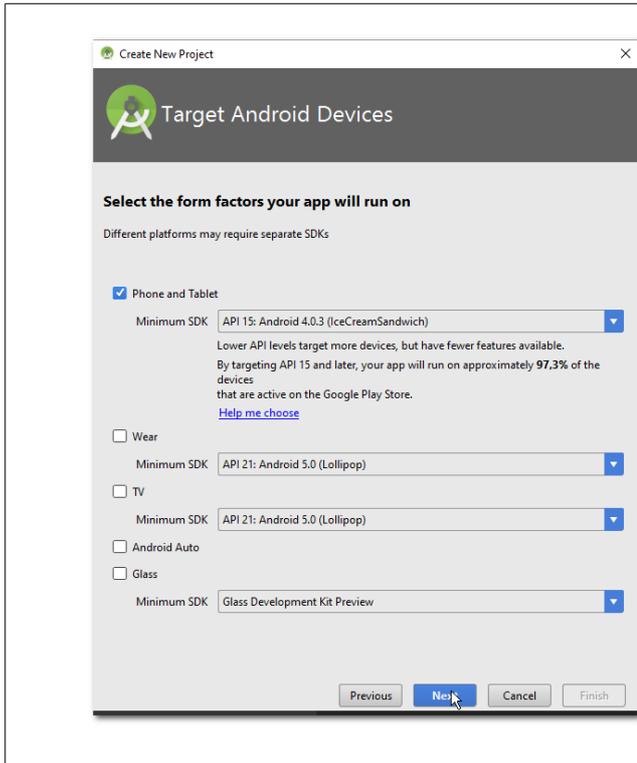
### Legen Sie nun schrittweise die Eigenschaften für Ihr neues Android-Projekt fest.

Geben Sie dazu die nebenstehend angezeigten Angaben für

1. Application name:  
Der Anwendungsname.
2. Company Domain:  
Ihre Internetadresse, die Ihrer Schule oder den Standardwert „name.example.com“.
3. Project location:  
Wir nutzen bestenfalls den bereits vorhandenen Arbeitsbereich in → EigeneDateien\Android\_Apps der Digitalen Tasche auf dem USB-Stick.

G:\Informatikstick2016\EigeneDateien\Android\_Apps\CowCounterApp

Je nach Konfiguration können diese Angaben variieren

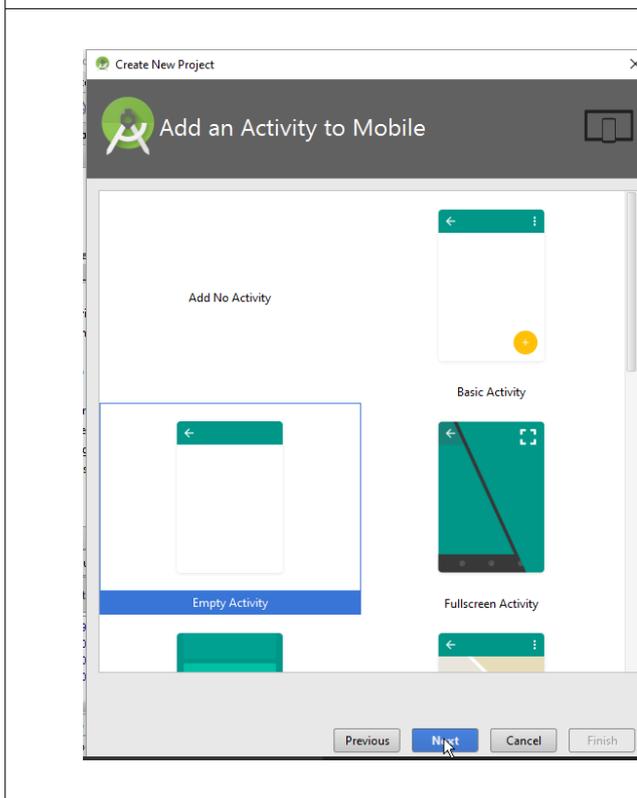


*Laufzeitumgebung unserer Anwendung.*

Wir wählen als Ziel unserer Anwendung das API Level, mit der höchsten Abdeckung für die Lauffähigkeit auf verfügbaren Android Geräten, aus.

Der Assistent macht uns dazu einen Vorschlag für Telefone und Tablets.

Wir nehmen den Vorschlag an und klicken auf die Schaltfläche → Next.



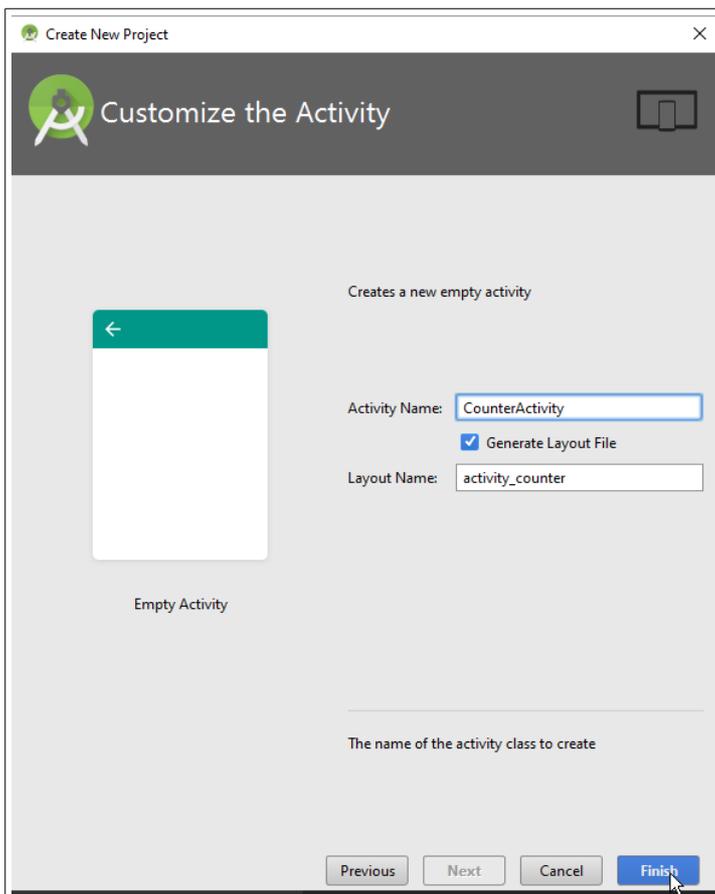
*Aktivität wählen.*

Im ersten Schritt nutzen wir die einfachste Form zur Steuerung von Ereignissen. Die → Empty Activity. Wählen wir diese Aktivität bekommen wir einige Standards mitgeliefert.

Wir wählen die → Empty Activity und klicken Sie auf die Schaltfläche → Next.

**Hinweis:**

Alternativ können wir auch die Option → Add No Activity wählen und können dann nachträglich alle Maßnahmen für die Implementierung der Activity selber treffen.



*Aktivität anpassen.*

Activities enthalten die Ereignissteuerung für einen bzw. eine ganze Reihe von zusammengehörenden Vorgänge (Interaktionen, Verhaltensweisen) einer App.

Übernehmen Sie die nebenstehenden Werte und klicken Sie anschließend die Schaltfläche → Finish.

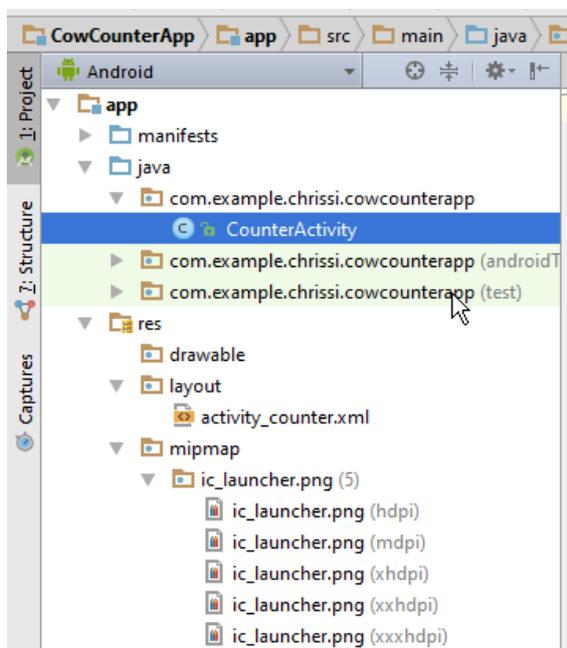
Mit dem Klick auf → Finish wird die Projektstruktur (Architektur) erzeugt.

**Hinweis:**

Je nach Rechnerausstattung kann die Erzeugung einen Moment dauern.

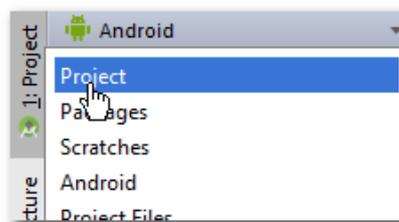
Android Studio nutzt u.a. das Gradle-PlugIn als Builtsystem. Gradle ist dabei ein Werkzeug das komplett in Android Studio integriert ist und zur Build-Automatisierung und - Management genutzt wird. Jede Anwendung muss nach jeder Änderungen im Quellcode neu erzeugt werden, dabei werden außer der Kompilierung viele weitere Bindungsprozesse (z.B. mit den Ressourcen) durchgeführt.

Android View



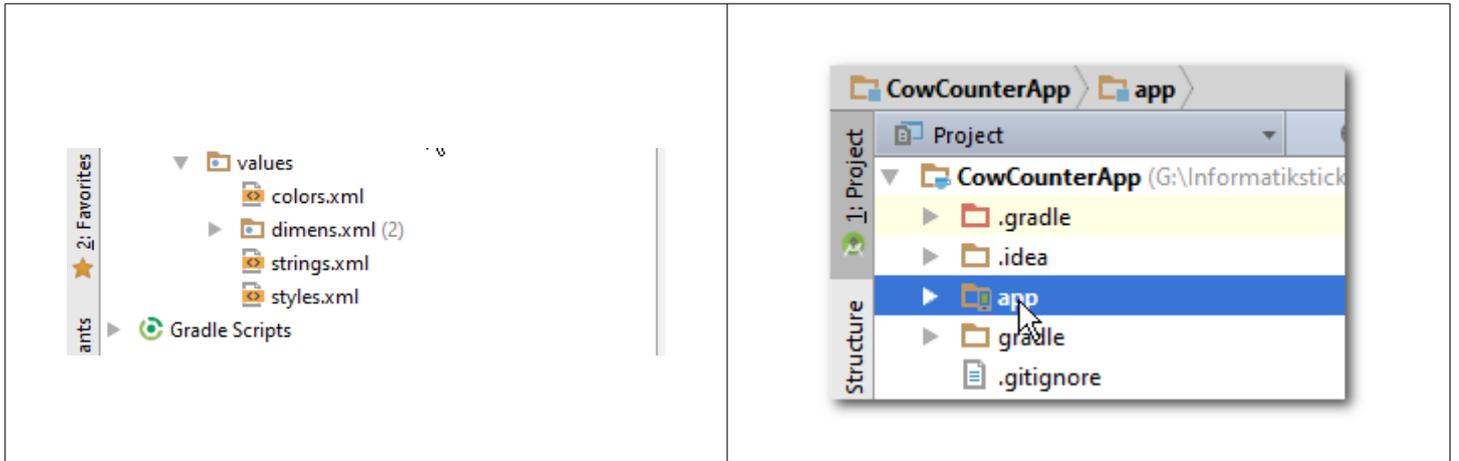
*Projektstruktur am Anfang.*

Im Anschluss an den abgeschlossenen Built-Prozess finden Sie im linken Frame die folgende Projektstruktur vor.



Klicken Sie oberhalb auf den Androiden um die Projektansicht → Project View zu wählen:

Folgen Sie den nächsten Schritten, um ein ersten Entwurf der Benutzeroberfläche zu erzeugen.



## 2.3 View: Layouts, Komponenten & XML für die Benutzeroberfläche

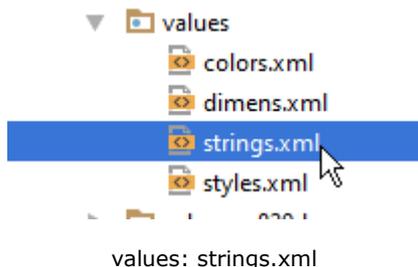


*Vorgehensweise erläutern.*

Es folgen nun die Erläuterungen zur Erstellung unserer Benutzeroberfläche. Dazu gehen wir folgende Schritte:

1. Bezeichner (Strings) deklarieren und initialisieren
2. Farben deklarieren und initialisieren
3. Activity-Layout anpassen

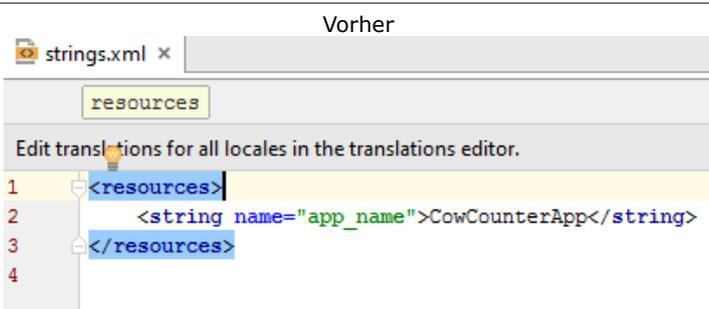
Folgen Sie der Schritt-Für-Schritt-Anleitung. Bedenken Sie, dass fehlende Schritte nachträglich zu Fehlern führen können.



*Bezeichner (Strings) deklarieren und initialisieren.*

Öffnen Sie dazu im Verzeichnis → app → res → values die Datei strings.xml mit einem Doppelklick auf den Dateinamen.

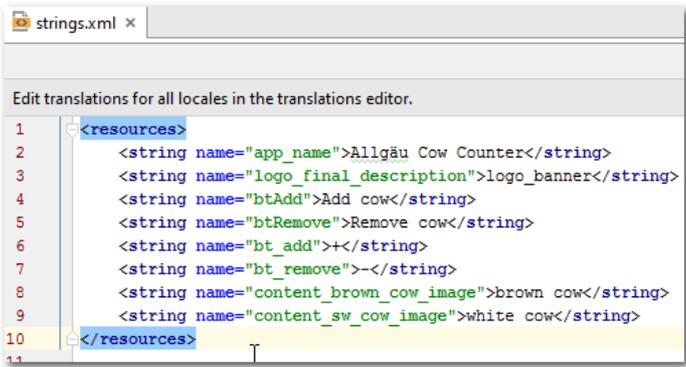
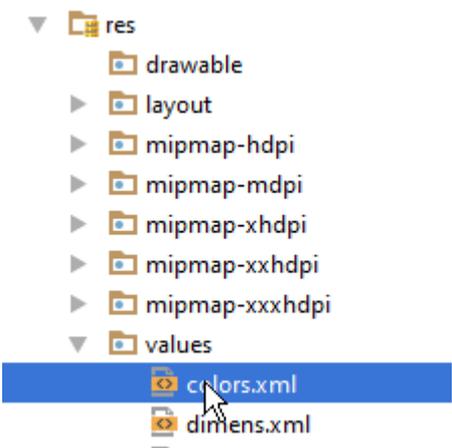
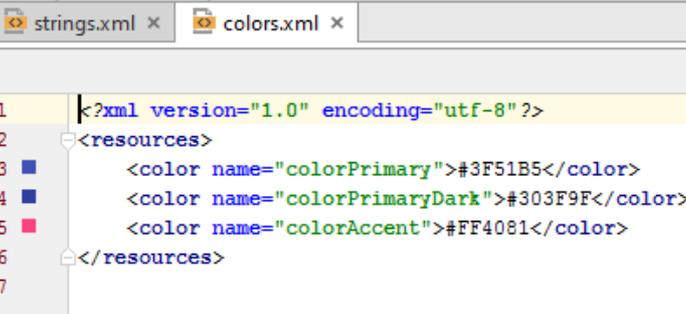
Ergänzen Sie den fehlenden Quellcode.



*Bezeichner definieren.*

Für die Allgemeinen Angaben definieren wir dazu, wie folgt:

```
<resources>
  <string name="app_name">
    Allgäu Cow Counter</string>
  <string name="logo_final_description">
    logo_banner</string>
  <string name="btAdd">Add cow</string>
  <string name="btRemove">Remove cow</string>
  <string name="bt_add">+</string>
  <string name="bt_remove">-</string>
  <string name="content_brown_cow_image">
    brown cow</string>
  <string name="content_sw_cow_image">
```

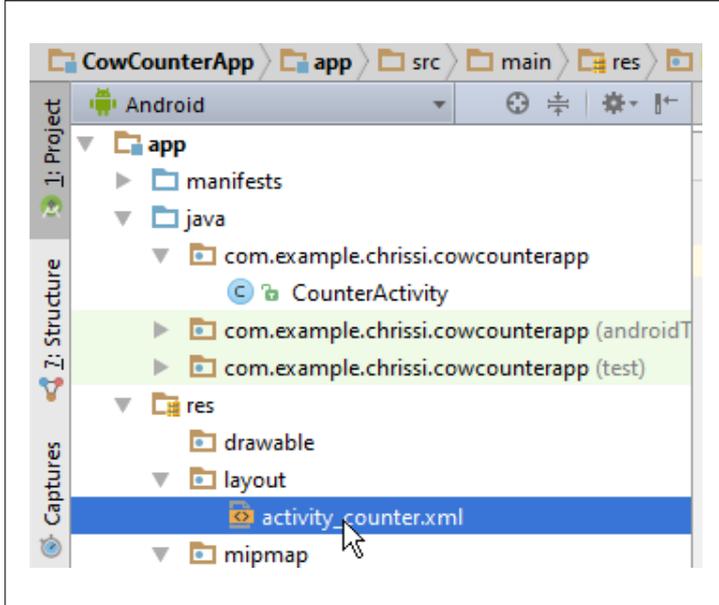
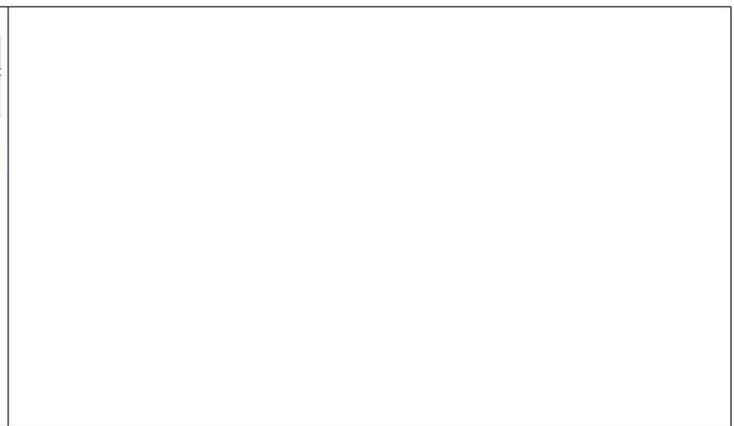
<p style="text-align: center;">Nachher</p> 	<pre style="background-color: yellow;">white cow&lt;/string&gt; &lt;/resources&gt;</pre> <p>Ergänzen Sie den Quellcode, wie nebenstehend angezeigt.</p>
 <p style="text-align: center;">values: colors.xml</p>	<p><i>Farben deklarieren und initialisieren.</i></p> <p style="background-color: yellow;">Öffnen Sie dazu im Verzeichnis → app → res → values die Datei colors.xml mit einem Doppelklick auf den Dateinamen.</p>
<p style="text-align: center;">Vorher</p> 	<p><i>Änderung der Farben durchführen</i></p> <p>Hexadezimalcodes für die verwendeten Farben:</p> <pre style="background-color: yellow;">&lt;color name="colorPrimary"&gt;#ff11afe5&lt;/color&gt; &lt;color name="colorPrimaryDark"&gt;#ff125c84&lt;/color&gt; &lt;color name="colorAccent"&gt;#ff11c4ff&lt;/color&gt; &lt;color name="layout_hintergrund"&gt;#ff11afe5&lt;/color&gt; &lt;color name="layout_links"&gt;#ff11afe5&lt;/color&gt; &lt;color name="layout_rechts"&gt;#fafafa&lt;/color&gt;</pre> <p>Ergänzen Sie den Quellcode, wie nebenstehend angezeigt.</p>

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <resources>
3   <color name="colorPrimary">#ff11afe5</color>
4   <color name="colorPrimaryDark">#ff125c84</color>
5   <color name="colorAccent">#ff11c4ff</color>
6   <color name="layout_hintergrund">#ff11afe5</color>
7   <color name="layout_links">#ff11afe5</color>
8   <color name="layout_rechts">#fafafa</color>
9 </resources>
10

```

Nachher



Activity Layout

*Layout der Activity anpassen.*

Für die erste Benutzeroberfläche:

Öffnen Sie dazu die Datei aus dem Unterverzeichnis app → src → main → res → layout → activity\_counter.xml mit einem Doppelklick auf den Dateinamen.

Im folgenden wird schrittweise beschrieben welche Änderungen erfolgen sollten.

**Das Relative Layout:**

Die in einem relativen Layout enthaltenen Komponenten werden immer in Abhängigkeit seiner direkt benachbarten Komponenten betrachtet. Deshalb erfolgt die Beschreibung der Platzierung auch in Abhängigkeit der direkt benachbarten Komponenten.

**Das Lineare Layout (vertikal):**

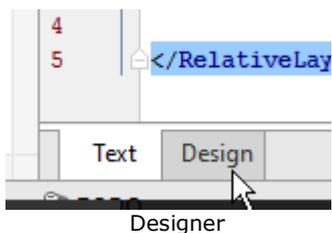
Die in einem vertikalen Linearen Layout platzierten Komponenten werden untereinander angeordnet.

**Das Lineare Layout (horizontal):**

Die in einem horizontalen Linearen Layout platzierten Komponenten werden nebeneinander angeordnet.

**Das Frame Layout:**

Die platzierten Komponenten können ausgehend vom linken oberen Rand ausgerichtet werden.



Designer

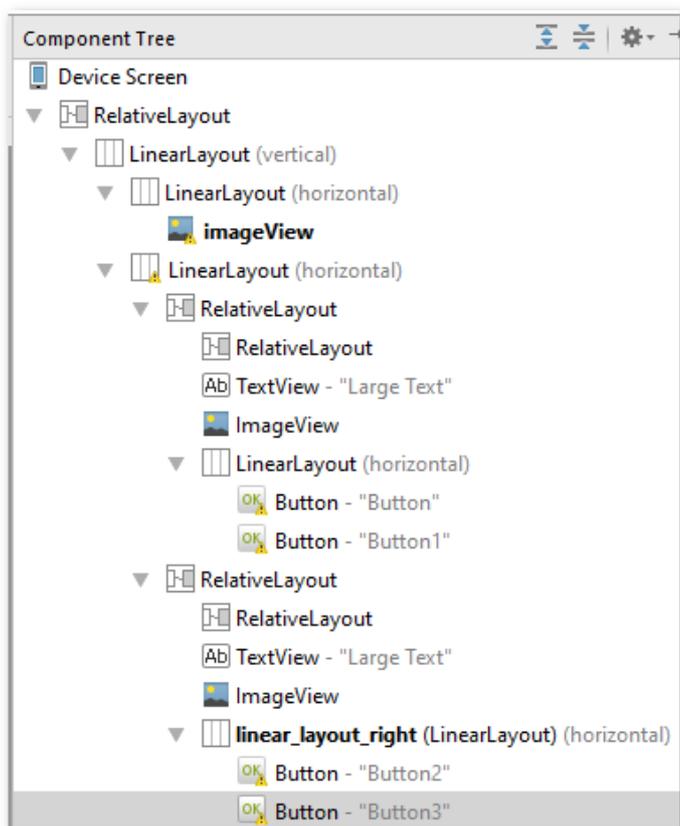
**Hinweis:**

Die Anwendung besitzt ähnlich, wie in Eclipse der im WindowsBuilder enthaltene Swing-Designer einen Quellcode-Generator. Im Gegensatz zu Eclipse erzeugt der Quellcode-Generator in Android Studio XML-Quellcode. Wir können jederzeit zwischen den Ansichten → Text und → Design wechseln.

*In den Design-Modus wechseln.*

Um das Design zu erstellen nutzen wir den Oberflächendesigner.

Klicken Sie dazu auf den Reiter → Design unterhalb des angezeigten XML-Quellcodes.



Gewünschtes Ergebnis

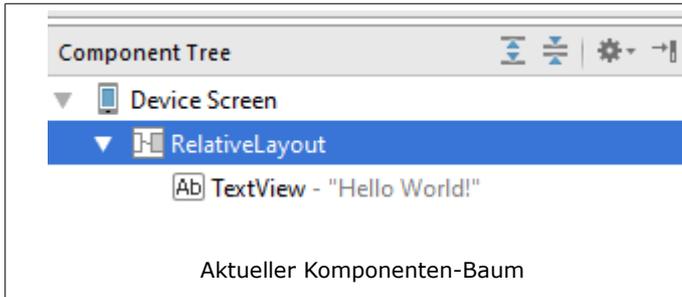
*Vorgehensweise: Component Tree.*

1. Layout (falls nötig) schachteln
2. Komponenten im Layout platzieren
3. Komponenteneigenschaften definieren

Nun folgen die Änderungen im aktuellen Komponenten-Baum, um das nebenstehende gewünschte Ergebnis zu erzeugen.

*Der Komponenten-Baum.*

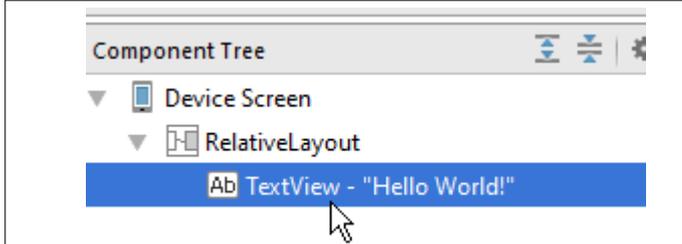
Im oberen, rechten Frame-Fenster wird der Komponenten-Baum (Component Tree) angezeigt.



Als Komponenten werden alle Elemente einer Benutzeroberfläche bezeichnet.

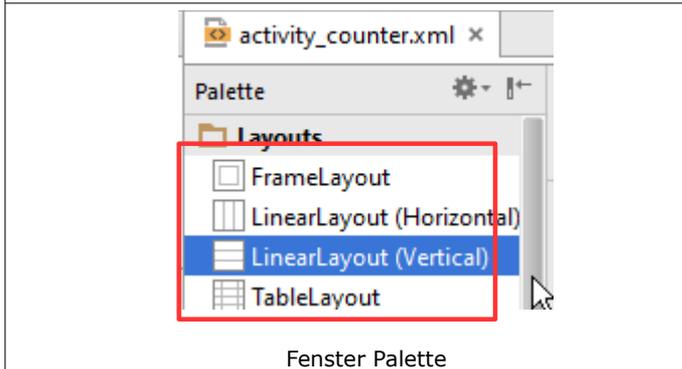
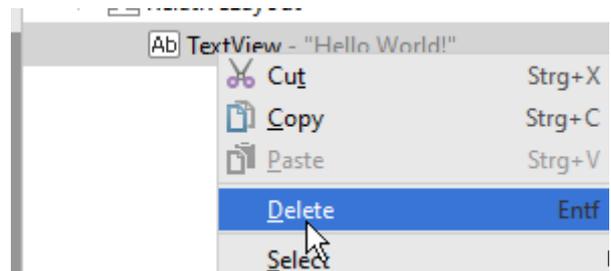
Die Grundlage jeder Benutzeroberfläche sind die Layouts.

Das Standard-Layout ist das → Relative Layout.



*Element entfernen.*

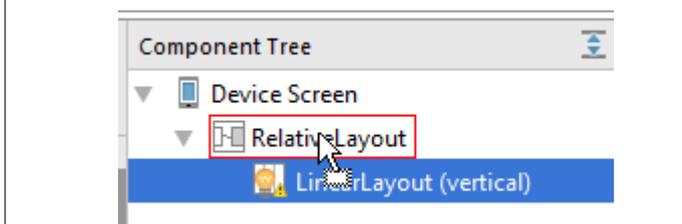
Klicken Sie mit der rechten Maustaste (Kontext-Menü) auf das TextView-Element „Hello World!“ und entfernen Sie das Element mit der Option → Delete

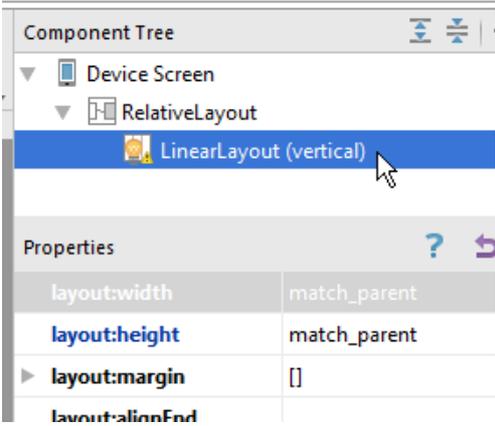
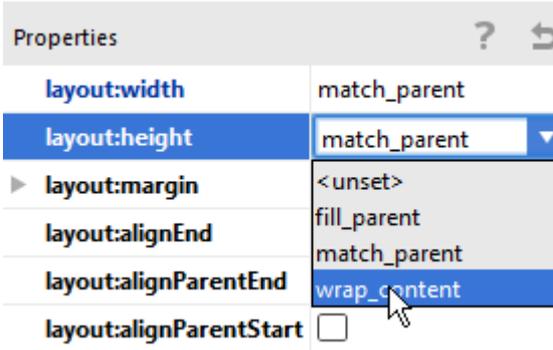
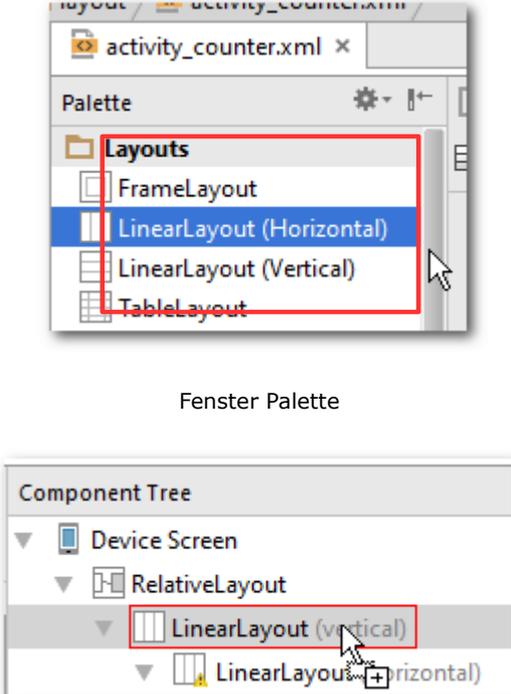


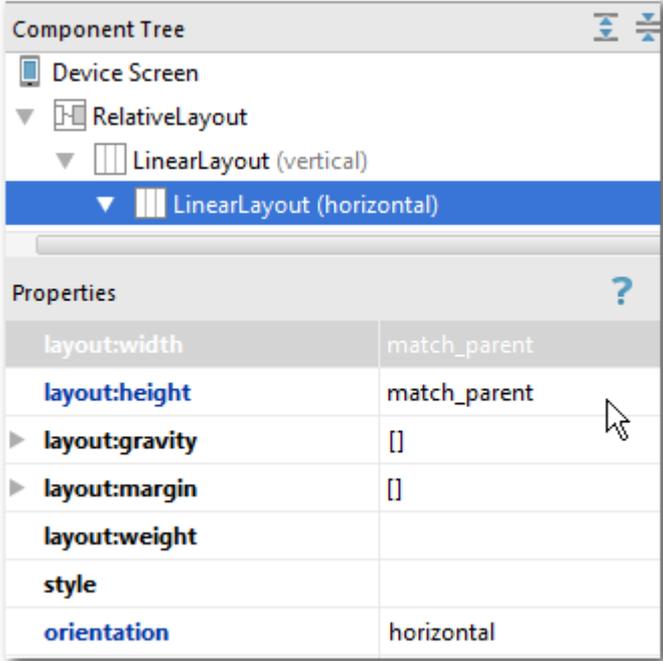
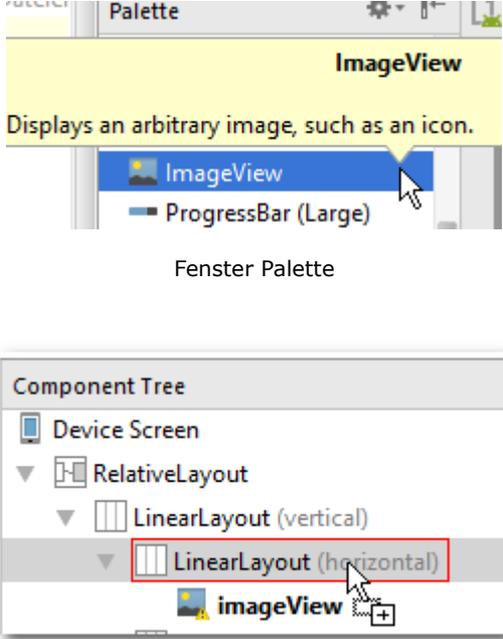
*LinearesLayout (Vertical) verwenden.*

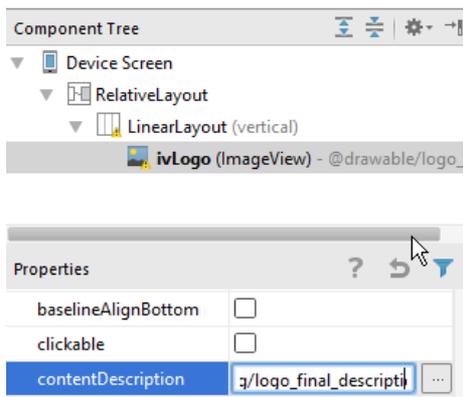
Wählen Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne die Option → "LinearLayout (Vertical)".

Ziehen Sie dann diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt. Lassen Sie dann die Maustaste los.



<p style="text-align: center;">Fenster Component Tree</p>  <p style="text-align: center;">Fenster Component Tree und Properties</p>	<p><i>Eigenschaften des vertikalen Layouts ändern.</i></p> <p>Klicken Sie dazu im Fenster → Component Tree auf das → LinearLayout (vertical).</p> <p>Prüfen Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.</p>  <p>Ändern Sie die Eigenschaften ggf. wie folgt ab.</p> <p>Eigenschaften:</p> <table style="background-color: yellow;"> <tr> <td>layout:width:</td> <td>match_parent</td> </tr> <tr> <td>layout:height:</td> <td>wrap_parent</td> </tr> <tr> <td>orientation:</td> <td>vertical</td> </tr> </table>	layout:width:	match_parent	layout:height:	wrap_parent	orientation:	vertical
layout:width:	match_parent						
layout:height:	wrap_parent						
orientation:	vertical						
 <p style="text-align: center;">Fenster Palette</p>	<p><i>LinearesLayout (Horizontales) verwenden.</i></p> <p>Wählen Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf die Option → "LinearLayout (horizontal)".</p> <p>Ziehen Sie dann diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt. Lassen Sie dann die Maustaste los.</p>						

<p style="text-align: center;">Fenster Component Tree</p>  <p style="text-align: center;">Fenster Component Tree und Properties</p>	<p><i>Eigenschaften des horizontalen Layouts ändern.</i></p> <p>Klicken Sie dazu im Fenster → Component Tree auf das → LinearLayout (horizontal).</p> <p>Prüfen Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.</p> <p>Ändern Sie die Eigenschaften ggf. wie folgt ab.</p> <p>Eigenschaften:</p> <table border="1" style="background-color: yellow;"> <tr> <td>layout:width:</td> <td>match_parent</td> </tr> <tr> <td>layout:height:</td> <td>match_parent</td> </tr> <tr> <td>orientation:</td> <td>horizontal</td> </tr> </table>	layout:width:	match_parent	layout:height:	match_parent	orientation:	horizontal
layout:width:	match_parent						
layout:height:	match_parent						
orientation:	horizontal						
 <p style="text-align: center;">Fenster Palette</p> <p style="text-align: center;">Fenster Component Tree</p>	<p><i>ImageView-Komponente verwenden.</i></p> <p>Wählen Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf die Option → ImageView.</p> <p>Ziehen Sie dann diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt. Lassen Sie dann die Maustaste los.</p>						



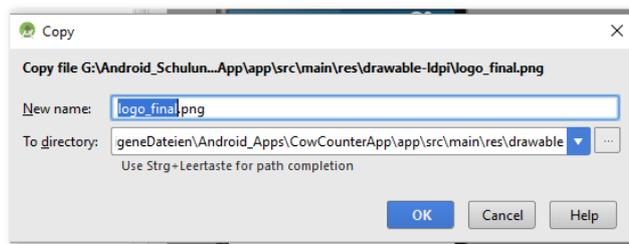
Fenster Component Tree und Properties



View im Designer

### Eigenschaften der ImageView ändern.

Kopieren Sie dafür die Grafik mit der Tastenkombination STRG + C und fügen Sie anschließend die Grafik mit der Tastenkombination STRG +V in das Unterverzeichnis → drawable ein:



Klicken Sie dann im Fenster → Component Tree auf die → ImageView Komponente.

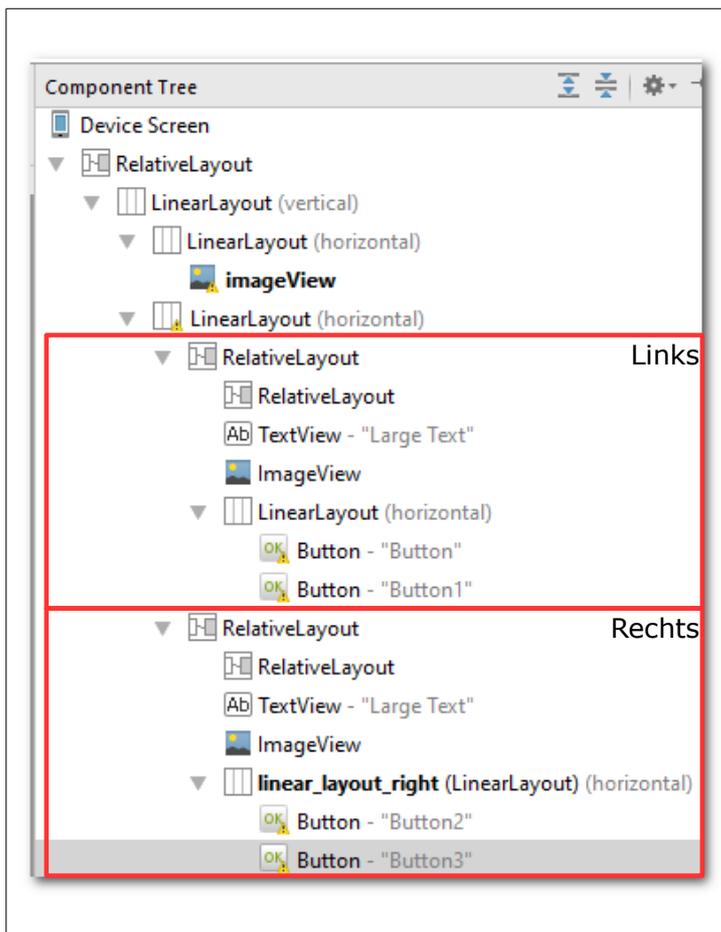
Prüfen Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.

Ändern Sie die Eigenschaften ggf. wie folgt ab.

#### Eigenschaften:

```

layout:width:      wrap_content
layout:height:    wrap_content
layout:gravity:    [center], both
contentDescription: @string/logo_final_description
id:                ivLogo
src:               @drawable/logo_final
    
```



Fenster Component Tree

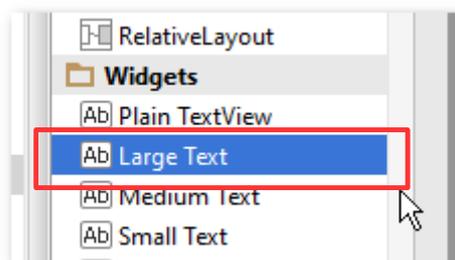
Hinweis:  
Wundern Sie sich nicht, wenn die Anzeige im Designer noch etwas ungeordnet aussieht. Wir werden im folgenden Schritt mit der Festlegung der Eigenschaften für alle Komponenten und Layouts Ordnung schaffen.

### Komponenten und Layouts platzieren.

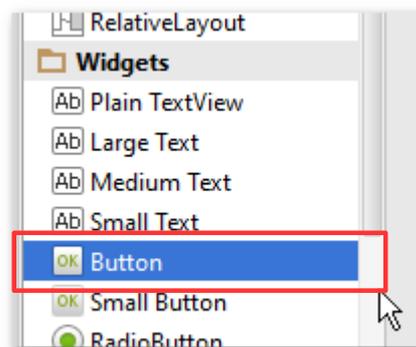
Das Prinzip sollte nun klar sein. Alle restlichen Komponenten und Layouts werden wir im nun folgenden Schritt im Component Tree integrieren. Anschließend werden wir für jede Komponente die Eigenschaften individuell festlegen.

Gehen Sie auf gleiche Weise vor, wie zuvor. Suchen Sie im Fenster Palette die Komponente/Layout aus ziehen und platzieren Sie dann die Komponente bzw. das Layout mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster, im → Component Tree.

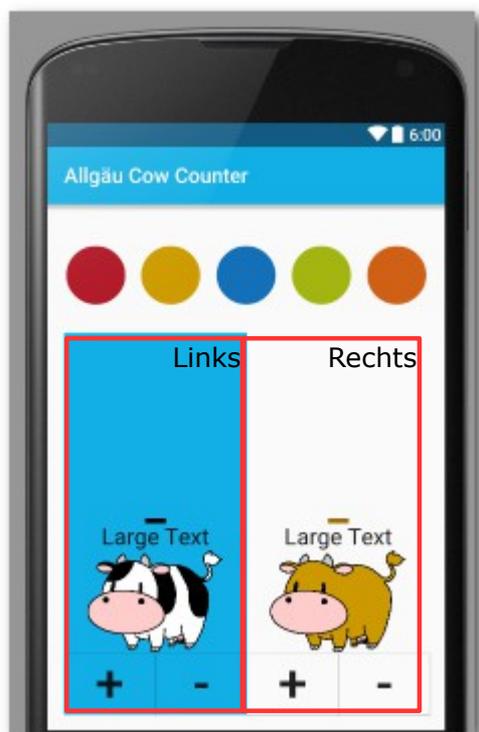
Layouts/Komponenten im Fenster Palette:



Large TextView



Button



Gewünschte View im Designer

### Density-independent pixel (dp):

Eine virtuelle Pixel-Maßeinheit (optisch unabhängige Dichte). Wird genutzt, um die Größenangaben für Layouts zu definieren. Im übrigen ist es aufgrund der Anpassungsfähigkeit in vielen Fällen besser auf „statische Größenangaben“ gänzlich zu verzichten.

### Density-independent Pixels:

Eine abstrakte Einheit welche auf der physikalischen Dichte des Displays basiert. Diese Einheit wird relative zu einem Display von 160dpi berechnet. 1dp entspricht also 160dpi auf einem Display.

### Eigenschaften festlegen.

Wir schaffen Ordnung und erzeugen die gewünschte View im Designer. Klicken Sie dazu die Komponente/Layout im Fenster → Component Tree nacheinander an und nutzen Sie jeweils die vertikale Bildlaufleiste im Fenster → Properties, um die Eigenschaft individuell, ändern zu können.

Die folgenden Änderungen sind notwendig.

Beginnen Sie mit dem horizontalen Layout unterhalb des bereits eingefügten Logos.

### LinearLayout (horizontal):

Box für die Inhalte aus Links und Rechts

```
layout:width:      match_parent
layout:height:     match_parent
orientation:       horizontal
```

### Für Links

### RelativeLayout:

Zum Anzeigen des linken blauen Layouts

```
layout:width:      wrap_content
layout:height:     fill_parent
layout:weight:     1
background:        @color/layout_links
```

### RelativeLayout:

Zum Anzeigen des schwarzen Balkens

```
layout:width:      20dp
layout:height:     5dp
layout:alignComponent: [bottom:top],textView
layout:centerInParent: horizontal
background:        #000000
id:                relativ_layout_bar_black
```

### Large TextView:

Für die Anzeige des Zählers für sw-Kühe

```
layout:width:      wrap_content
layout:height:     wrap_content
layout:alignComponent: [bottom:top],ImageView
layout:centerInParent: horizontal
id:                tv_white
text:              Large Text
textAppearance:    ?android:attr/textAppearanceLarge
```

### ImageView:

**Scale-independent pixel (sp):**

Ist die bevorzugte Größeneinheit für Schriften. Die Größenangabe verhält sich auf ähnliche Weise wie die dynamische Angabe in dp. Sie berücksichtigt jedoch zusätzlich die vom Benutzer präferierte Größenangabe des Benutzer.



View im Designer:  
mit gesetzten Eigenschaften

Für die Anzeige der Grafik der sw-Kuh

```
layout:width:      wrap_content
layout:height:     100dp
layout:alignComponent:[bottom:top],LinearLayout
layout:centerInParent:  horizontal
contentDescription:@string/content_sw_cow_image
id:                iv_sw_cow
src:               @drawable/kuh_sw
```

Hinweis: die Grafik fügen wir später ein.

**LinearLayout (horizontal):**

Box für die Anordnung der +/- Schaltflächen

```
layout:width:      wrap_content
layout:height:     wrap_content
layout:alignParent: [bottom]
layout:centerInParent:  horizontal
orientation:        horizontal
gravity:            [bottom]
id:                linear_layout_left
```

**Button:**

Schaltfläche „+“ für schwarz-weiße Kühe

```
layout:width:      wrap_content
layout:height:     wrap_content
background:        @android:color/transparent
id:               bt_add_sw_cow
text:              @string/bt_add
textSize:         50sp
```

**Button: Schaltfläche „-“ für schwarz-weiße Kühe**

```
layout:width:      wrap_content
layout:height:     wrap_content
background:        @android:color/transparent
id:               bt_remove_sw_cow
text:              @string/bt_remove
textSize:         50sp
```

**Für Rechts****RelativeLayout:**

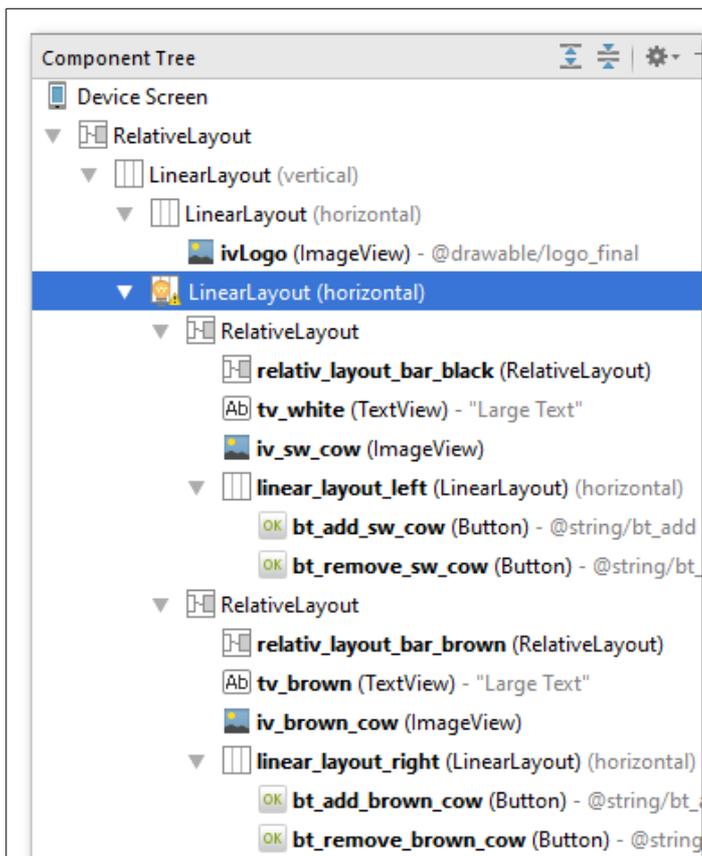
Zum Anzeigen des linken weißen Layouts

```
layout:width:      wrap_content
layout:height:     fill_parent
layout:weight:     1
background:        @color/layout_rechts
```

**RelativeLayout:**

Zum Anzeigen des brauner Balkens

```
layout:width:      20dp
layout:height:     5dp
layout:alignComponent: [bottom:top],textView
layout:centerInParent: horizontal
background:        #9f6c07
```



Fenster Component Tree:  
mit gesetzten Eigenschaften

**id:** relativ\_layout\_bar\_brown

**Large TextView:**

Für die Anzeige des Zählers für braunen Kühe

**layout:width:** wrap\_content  
**layout:height:** wrap\_content  
**layout:alignComponent:** [bottom:top],ImageView  
**layout:centerInParent:** horizontal  
**id:** tv\_brown  
**text:** Large Text  
**textAppearance:** ?android:attr/textAppearanceLarge

**ImageView:**

Für die Anzeige der Grafik der braunen Kuh

**layout:width:** wrap\_content  
**layout:height:** 100dp  
**layout:alignComponent:** [bottom:top],LinearLayout  
**layout:centerInParent:** horizontal  
**contentDescription:** @string/content\_brown\_cow\_image  
**id:** iv\_brown\_cow  
**src:** @drawable/kuh\_braun

Hinweis: die Grafik fügen wir später ein.

**LinearLayout (horizontal):**

Box für die Anordnung der +/- Schaltflächen

**layout:width:** wrap\_content  
**layout:height:** wrap\_content  
**layout:alignParent:** [bottom]  
**layout:centerInParent:** horizontal  
**orientation:** horizontal  
**gravity:** [bottom]  
**id:** linear\_layout\_right

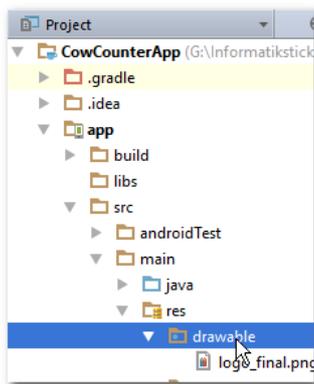
**Button:**

Schaltfläche „+“ für braune Kühe

**layout:width:** wrap\_content  
**layout:height:** wrap\_content  
**background:** @android:color/transparent  
**id:** bt\_add\_brown\_cow  
**text:** @string/bt\_add  
**textSize:** 50sp

**Button: Schaltfläche „-“ für braune Kühe**

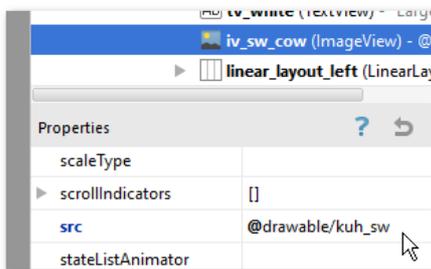
**layout:width:** wrap\_content  
**layout:height:** wrap\_content  
**background:** @android:color/transparent  
**id:** bt\_remove\_brown\_cow  
**text:** @string/bt\_remove  
**textSize:** 50sp



Verzeichnis: drawable



View im Designer:  
mit Grafiken

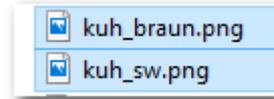


Fenster Properties: Eigenschaft → src

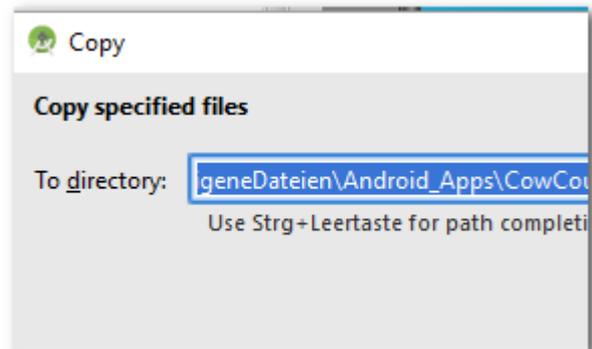
### Grafiken einfügen.

Wir fügen nun die Grafik für die schwarz-weiße und braune Kuh ein.

Kopieren Sie die beiden Grafiken mit der Tastenkombination STRG + C.



Wählen Sie dann in Ihrem Projekt das Unterverzeichnis app → src → main → res → drawable und fügen Sie die Grafiken mit der Tastenkombination STRG+ V ein.

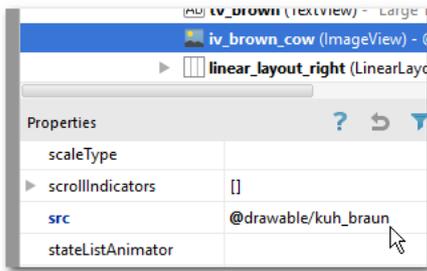


Klicken Sie abschließend auf die Schaltfläche → OK.

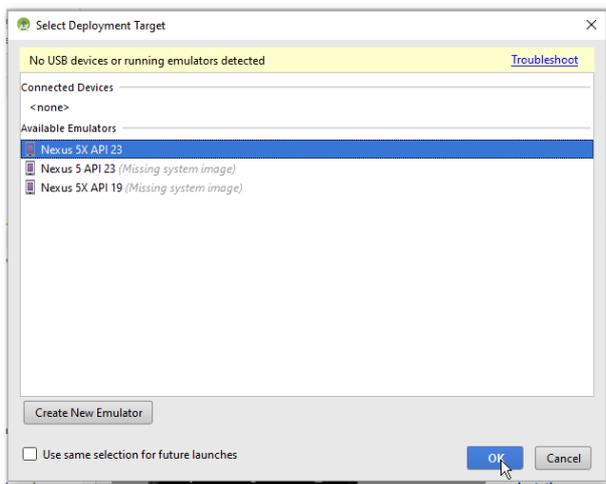
Kontrollieren Sie anschließend nochmals sicherheitshalber die Eigenschaften → src für die ImageView-Komponenten (iv\_sw\_cow, iv\_brown\_cow) im Fenster Properties.

ImageView: iv\_sw\_cow  
@drawable/kuh\_sw

ImageView: iv\_brown\_cow  
@drawable/kuh\_braun



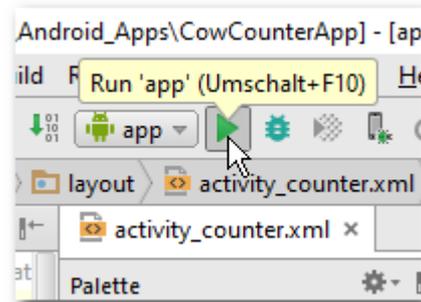
Fenster Properties: Eigenschaft → src



Alternativ → Create New Emulator:  
 Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit API 15 (SanwichIceCream) zu erzeugen:

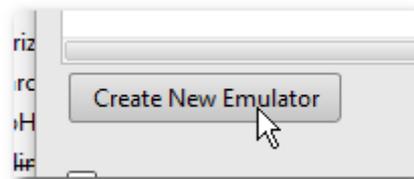
*Testen der View.*

Wir starten nun den Emulator.



**Emulator:**

Der Emulator simuliert im vorliegenden Fall ein virtuelles Mobiltelefon vom Typ → Nexus 5 API 23.



*Der Emulator öffnet sich.*

Beim ersten öffnen kann das einen Moment dauern.

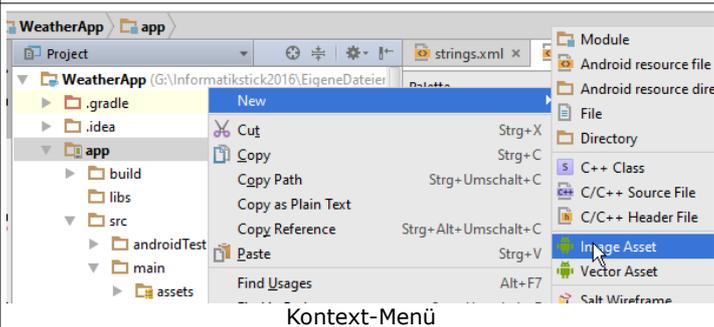
Ziehen Sie dann das auf dem Display erscheinende Schlösschen mit gedrückter linken Maustaste senkrecht nach oben.

Wenn Sie nicht ungeduldig werden, startet der Emulator die App nach Abschluss des Built-Pro-



zesses von selbst.

Im Ergebnis sollte die Benutzeroberfläche erscheinen.



*App Icon ändern.*

Klicken Sie dazu mit der rechten Maustaste in Ihrem Projekt auf das Verzeichnis app → src → res wählen Sie im Kontext-Menü die Option → New → Image Asset.

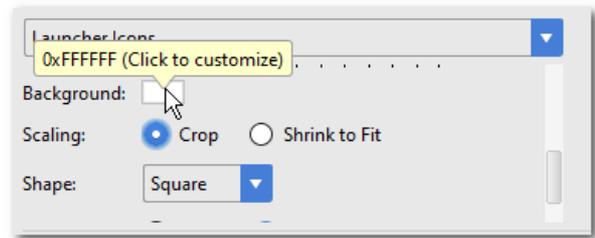
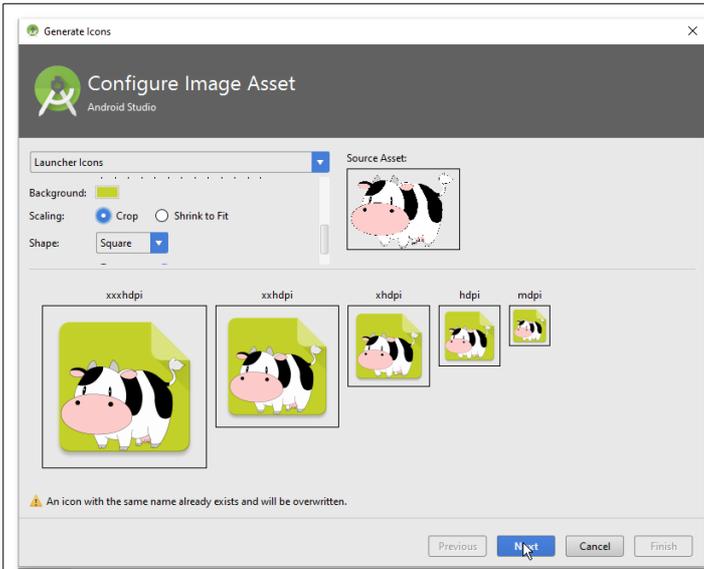
*Image Icon definieren.*

Wählen Sie für den Asset Type die Option Image. Wählen Sie dann für den Image-File-Pfad die Bild-Datei aus. Klicken Sie dazu auf die Schaltfläche ... und wählen Sie die Bildquelle aus.

Bildquelle:

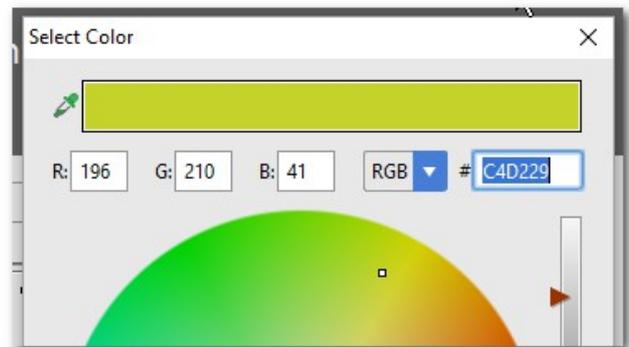
Material\AppIcons\_CowCounter\kuh\_sw\_icon.png

Klicken Sie dann auf das weiße Feld → Background



Geben Sie im Fenster → Select Color den Hexadezimalcode ein:

**C4D229**



Bestätigen Sie die Eingabe mit einem Klick auf die Schaltfläche → OK.

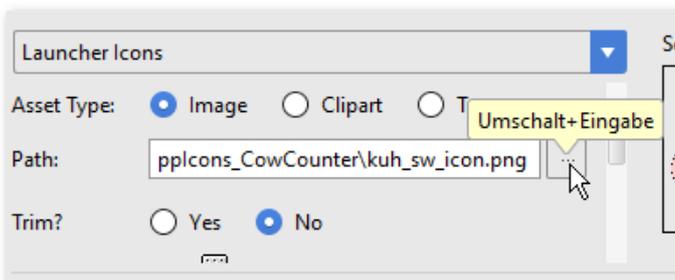
Aktivieren Sie weiter für die Eigenschaft → Scaling die Option → Crop und für die Eigenschaft → Shape die Option → Square.

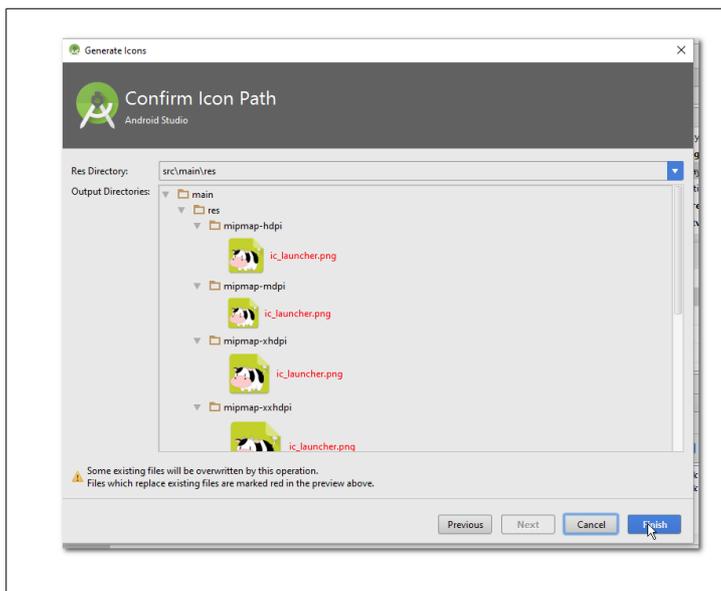
Klicken Sie auf die Schaltfläche → Next.



G:\Android\_Schulung\Material\Applcons\_CowCounter

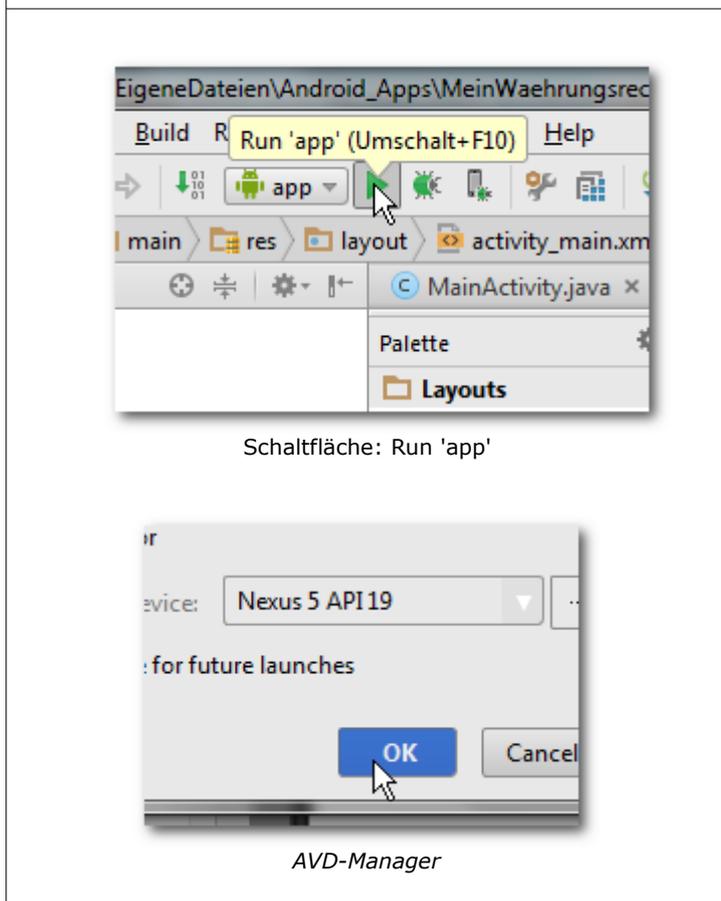
Bildquelle





### Icon Konfiguration abschließen.

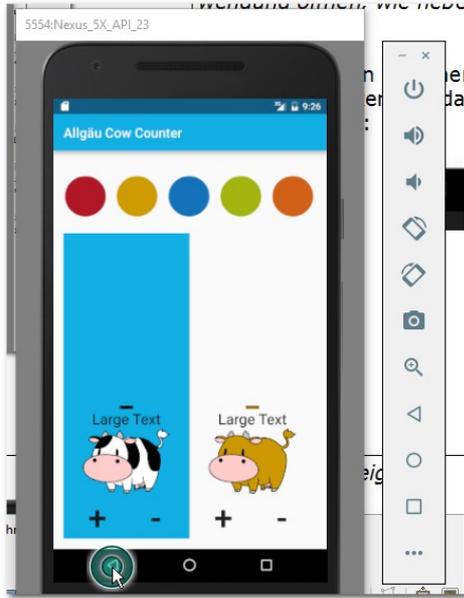
Klicken Sie auf Finish. Dabei wird das vorhandene Icon überschrieben.



### Icon und Logo Testen.

Testen Sie wie gewohnt die Anwendung. Klicken Sie dazu in der Symbol-Leiste auf die Schaltfläche „Run“.

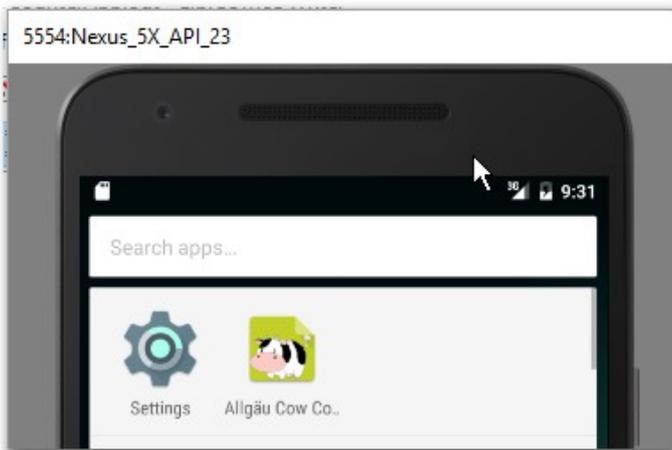
Starten Sie die AVD mit einem Klick auf die Schaltfläche „OK“.



*Bild (Logo) anzeigen.*

*Mit dem Öffnen der AVD sollte sich auf die Anwendung öffnen, wie nebenstehend angezeigt.*

Um das App Icon zu sehen wechseln Sie in das App-Menü. Klicken Sie dazu diese Schaltfläche auf dem Display:



*Icon anzeigen.*

Auf dem Display ist das App Icon nun aufgeführt.



**Gratulation das Layout ist erstellt**

## 2.4 Modell: Implementierung der Fachklassen für die Datenhaltung



Designer

*Vorgehensweise erläutern.*

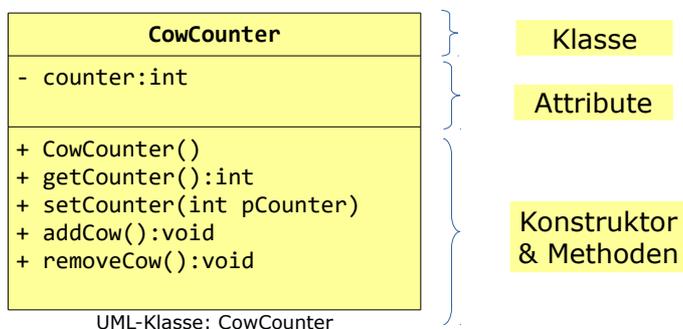
Es folgen nun die Erläuterungen zur Erstellung unseres Modells. Das Modell enthält die Deklaration und Implementierung aller systemrelevanten Objekteigenschaften und -verhaltensweisen die der zeitweisen Datenaquise und -haltung dienen.

Im Falle der CowCounter App benötigen wir nur eine ganz schlichte Fachklasse → CowCounter.

*Neue Modellklasse → CowCounter erstellen.*

Die Objekte dieser Klasse soll eine Zählervariable → counter inkrementieren (hochzählen) und dekrementieren (runterzählen) können.

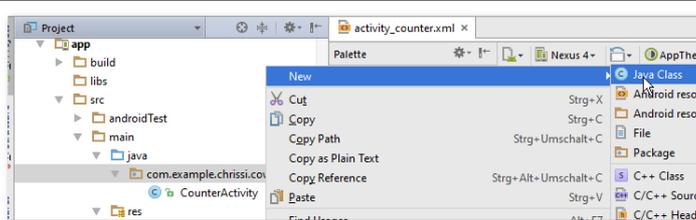
Entsprechend den Vorgaben (Anforderungen) der nebenstehend angezeigten UML-Klasse, werden wir in den kommenden Schritten diese Fachklasse implementieren.

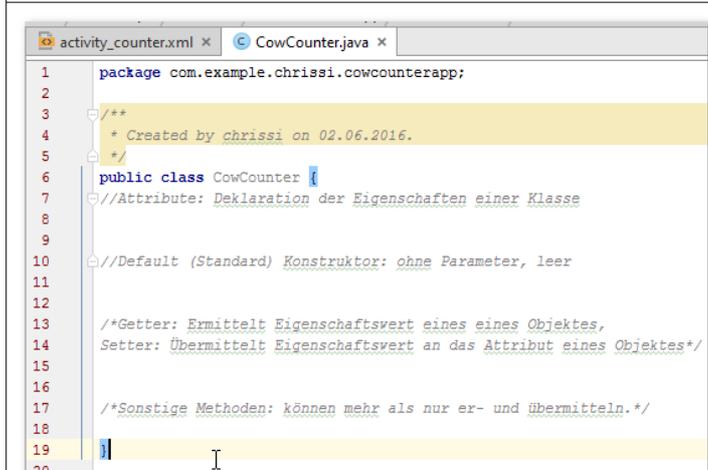
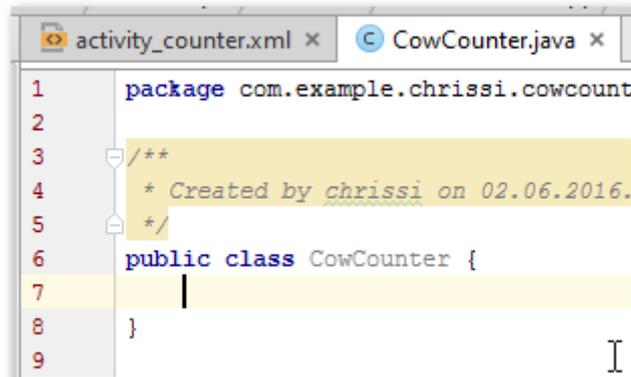
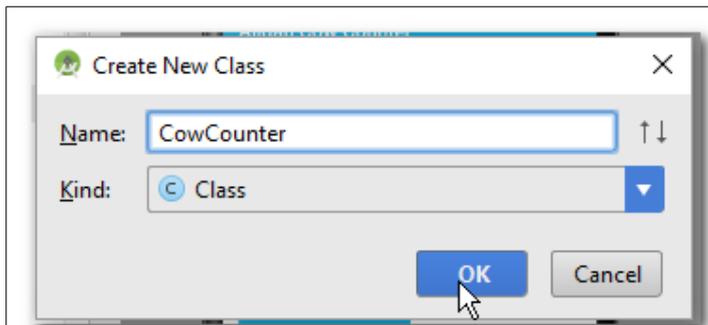


*Klassenname festlegen.*

Klicken Sie im → app-Verzeichnis mit der rechten Maustaste auf das → Package und wählen Sie die Option New → Java Class.

Geben Sie als Klassenname → CowCounter ein und klicken Sie auf die Schaltfläche → OK.





Grundgerüst einer Klasse festlegen.

Übernehmen Sie die nebenstehend angezeigten Kommentare.

Im Allgemeinen Fall ist das Grundgerüst einer Modell- oder Fachklasse, wie folgt aufgebaut:

- ✓ Deklaration der Attribute
- ✓ Deklaration des Konstruktors
- ✓ Get-Methoden (Getter) deklarieren und implementieren.
- ✓ Set-Methode (Setter) deklarieren und implementieren.
- ✓ Sonstige Methoden deklarieren und implementieren

Was ist → deklarieren?

In der objektorientierten Programmierung ist mit der Deklaration die

- ✓ Festlegung einer Dimension, eines Bezeichners,
- ✓ eines Datentyp und
- ✓ weiterer Aspekte einer Klasse, eines Konstruktors, einer Eigenschaft (Attribut) oder einer Verhaltensweise (Methode und Signatur),

gemeint.

Was ist → implementieren?

In der objektorientierten Programmierung ist mit der Implementation die Einbettung bzw. Umsetzung konkreter Programmstrukturen gemeint. Die sogenannte Umsetzung vom „Business Logic“ (automatisierte Prozesse) in Programmcode (Quellcode, Source) einer bestimmten Programmiersprache. Zumeist handelt es sich um das Anfüllen der Methoden mit dem benötigten Quellcode, also Inhalt einer Methode. Dabei dient

Eingabehilfe:

```

/**Attribute: Deklaration der Eigenschaften einer Klasse

/**Default (Standard) Konstruktor: ohne Parameter, Leer*/

/**Getter: Ermittelt Eigenschaftswert eines eines Objektes, Setter: Übermittelt Eigenschaftswert an das Attribut eines Objektes*/

/**Sonstige Methoden: können mehr als nur er- und übermitteln. */
    
```

der Quellcode dazu, die gewünschten Verhaltensweisen eines Systems (Programms) zu realisieren.

```

1 package com.example.chrissi.cowcounterapp;
2
3
4 /**
5  * Created by chrissi on 02.06.2016.
6  */
7 public class CowCounter {
8     //Attribute: Deklaration der Eigenschaften einer Klasse
9     private int counter = 0;

```

**Erläuterung Zugriffsmodifikatoren:**

- private (-)  
stellt sicher, dass nur die Objekte der Klasse selbst auf die Eigenschaftswerte direkt zugreifen können.
- package (~)  
stellt sicher, dass die Objekte des Pakets auf die Eigenschaftswerte direkt zugreifen können.
- public (+)  
stellt sicher, dass alle Objekte auf die Eigenschaftswerte direkt zugreifen können.
- protected, kein Modifikator (#)  
stellt sicher, dass nur die Objekte der Klasse und Objekte erbender Klassen auf die Eigenschaftswerte direkt zugreifen können.

**Übersicht Zugriffsmodifikatoren:**

	Class	Package	Subclass	World
public	j	j	j	j
protected	j	j	j	n
no modifier	j	j	n	n
private	j	n	n	n

j: erreichbar/zugreifbar  
n: nicht erreichbar/zugreifbar

**Deklaration und Initialisierung der Attribute.**

Fügen Sie nun das Attribut → counter ein, wie nebenstehend angezeigt. Wir initialisieren den Zähler mit dem Anfangswert 0.

Eingabehilfe:

```
private int counter = 0;
```

**Modifikatoren:**

**static:**

Ist ein Schlüsselwort (keyword) für Attribute und Methoden. Wenn in Java eine Eigenschaft als static deklariert wird bedeutet das, dass alle Objekte dieser Klasse den selben Eigenschaftswert nutzen. Die Attributnamen statischer Eigenschaften werden kursiv geschrieben.

**final:**

Ist ein Schlüsselwort (keyword) für Attribute in Java. Wenn in Java eine Eigenschaft als final deklariert wird ist eine Änderung des Eigenschaftswertes unerwünscht. Auch deshalb haben finale Eigenschaften keine implementierten Getter und Setter. Die Attributnamen finaler Eigenschaften werden in Großbuchstaben geschrieben.

**Zugriffsmodifikatoren:**

regeln den Zugriff auf Eigenschaftswerte einer Klasse (Rechtesystem in Objektorientierten Sprachen).

Folgen Sie den Erläuterungen, um die restliche Implementierung der Klasse schrittweise zu vollziehen.

<pre> 10 //Default (Standard) Konstruktor: 11 // ohne Parameter, leer 12 public CowCounter() { 13 14 }</pre>	<p><i>Deklaration des Standard-Konstruktors.</i></p> <p>Entsprechend dem Grundgerüst einer Klasse implementieren wir den Standardkonstruktor für diese Klasse.</p> <p>Eingabehilfe:</p> <pre> public CowCounter(){     //hier fehlt Quellcode }</pre>
<p>Beispiel Attribut „counter“:</p> <p>Get-Methode</p> <pre> public int getCounter(){     return this.counter; }</pre> <p>Set-Methode</p> <pre> private void setCounter(int pCounter){     this.counter = pCounter; }</pre> <p>Kapselung: Ist ein Prinzip der Informatik bei dem der Zugriff auf Daten bewusst durch den Programmierer reguliert wird. In der OOP werden dazu sog. Zugriffsmodifikatoren (z.B. → private, → public, → protected) genutzt. Damit verhindert der Programmierer der Klasse, dass ein anderer Programmierer durch den Zugriff aus seiner Klasse unfreiwillige Manipulierungen der Daten durchführen kann. Mit den Get- und Set-Methoden kann der Programmierer den bewussten Zugriff auf Daten ermöglichen/erlauben.</p>	<p><i>Deklaration und Implementierung der Get- und Set-Methoden.</i></p> <p>Berücksichtigen Sie, dass wir auf die Eigenschaftswerte der CowCounter-Objekte von außerhalb der Klasse (z.B. von der Benutzeroberfläche aus) zugreifen müssen. Jedes Attribut benötigt deshalb eine Get- und Set-Methode.</p> <p>Implementieren Sie die Get- und Set-Methoden für das Attribut → counter.</p> <p>Eingabehilfe: Getter</p> <pre> public int getCounter(){     return this.counter; }</pre> <p>Eingabehilfe: Setter</p> <pre> private void setCounter(int pCounter){     this.counter = pCounter; }</pre>

```

26      /*Sonstige Methoden: können mehr als nur er- und übermitteln.*/
27
28      //Erhöht (inkrementiert) den counter um 1
29      public void addCow(){
30          counter++;
31      }
32
33      //Minimiert (dekrementiert) den counter um 1
34      public void removeCow(){
35          //minimieren
36          counter--;
37
38          //Prüfung um einen negative Anzahl zu vermeiden
39          if(counter < 0){
40              //setzt den counter gleich 0
41              counter = 0;
42          }
43      }
    
```

### Deklaration und Implementierung sonstiger Methoden.

Die folgende Methode erhöht (inkrementiert) den Zähler (→ counter) um 1.

Implementieren Sie diese Methode.

Eingabehilfe: Variante 1

```

public void addCow(){
    counter++;
}
    
```

Weitere mögliche Varianten für → addCow() erzielen das gleiche Ergebnis:

```

public void addCow(){
    counter = counter + 1;
}
    
```

Variante 2

```

public void addCow(){
    counter += 1;
}
    
```

Variante 3

Die folgende Methode minimiert (dekrementiert) den Zähler (→ counter) um 1.

Implementieren Sie diese Methode.

Eingabehilfe: Variante 1

```

public void removeCow(){
    //minimieren
    counter--;
    //Prüfung um einen negative Anzahl zu vermeiden
    if(counter < 0){
        //setzt den counter gleich 0
        counter = 0;
    }
}
    
```

Weitere Varianten für → removeCow():

```

public void removeCow(){
    //minimieren
    counter = counter - 1;

    //Prüfung um einen negative Anzahl zu vermeiden
    if(counter < 0){
        //setzt den counter gleich 0
        counter = 0;
    }
}
    
```

Variante 2

```

public void removeCow(){
    //minimieren
    counter -= 1;

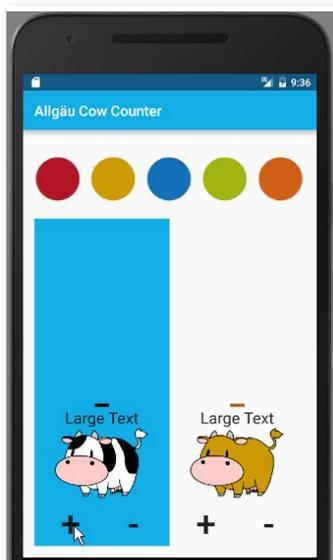
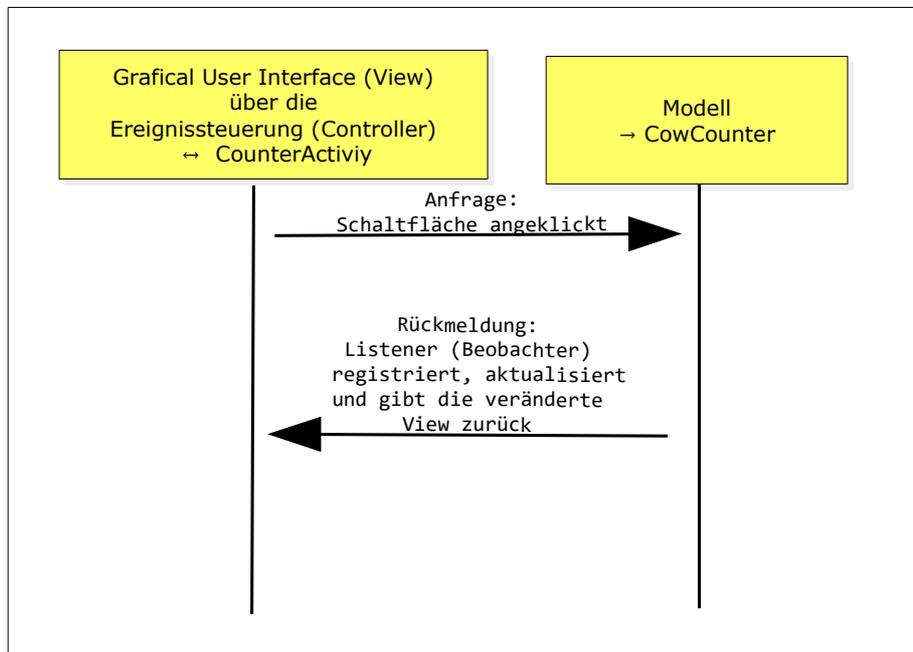
    //Prüfung um einen negative Anzahl zu vermeiden
    if(counter < 0){
        //setzt den counter gleich 0
        counter = 0;
    }
}
    
```

Variante 3

Damit sind die Voraussetzungen für unsere CowCounter App geschaffen. Im nächsten Schritt werden wir die Ereignissteuerung implementieren. Wir brauchen einen Beobachter der die Aktivitäten auf der Benutzeroberfläche registriert und die Zähler-Objekte für schwarz-weiße und braunen Kühe steuern kann. Diese Funktion wird die Klasse → CounterActivity.java übernehmen. Los geht's!

## 2.5 Controller: Ereignisse steuern

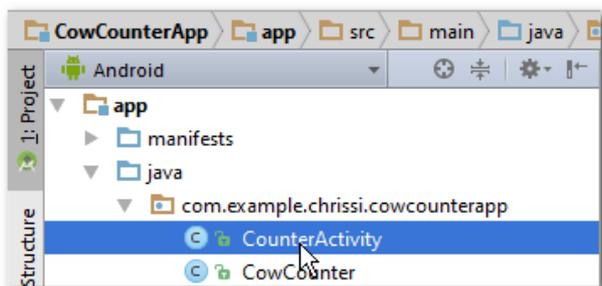
### Ereignissteuerung der CowCounter App



#### *Vorgehensweise erläutern.*

Es folgen nun die Erläuterungen zur Erstellung unserer Ereignissteuerung (Controller). Dazu implementieren wir gehen wir folgende Schritte:

1. CowCounter-Objekte erzeugen
2. Komponenten deklarieren
3. Benutzeroberfläche initialisieren
4. Benutzeroberfläche beobachten
5. Benutzeroberfläche aktualisieren



Öffnen Sie die Klasse CounterActivity.java.

**Activity:**

Bei Anwendungen auf Android Betriebssystemen erfolgt die Zerlegung aufgabenorientiert. Konkret bedeutet das, dass der Quellcode für die Steuerung einer Funktionalität in eine Activity-Klasse ausgelagert wird. Vielfach erkennt man die Aktivitäten (Activities) schon auf der Benutzeroberfläche, denn u.a. repräsentieren Schaltflächen solche Funktionalitäten.

```

1 package com.example.chrissi.cowcounterapp;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class CounterActivity extends AppCompatActivity {
7

```

**Pakete und Importe**

API Auszug: Klasse AppCompatActivity

**Hinweis zu älteren Projekten:**  
Bei den meisten älteren Projekten erbt die Activity noch von der ActionBarActivity

Die Verwendung der Klasse ActionBarActivity ist allerdings hinfällig (→ deprecated).

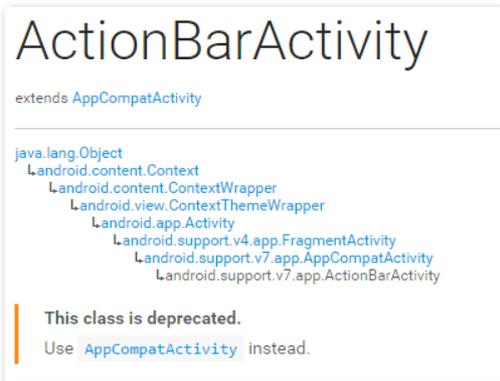
**Pakete und Importe.**

Zeile 1 beinhaltet die Angabe des Package. Die Angabe setzt sich zusammen aus den eingangs definierten Projekteigenschaften (→ Domain und → App name).

Im Gegensatz zu anderen Java-Anwendungen benötigen Android Apps die Activity, um eine Instanz der Anwendung zu erzeugen, außerdem stellt sie den Lebenszyklus der Instanz sicher und ergreift ggf. alle lebenserhaltenden Maßnahmen. Im Prinzip übernimmt das Objekt der Activity-Klasse u.a. die Funktionalität der Main-Methode einer konventionellen Java-Anwendung.

Die vererbten standardmäßig vorhandenen Verhaltensweisen (Methoden) einer → Activity erfordern die im oberen Teil der Klasse angegebenen Import-Anweisungen der Klassen → AppCompatActivity und → Bundle.

Die Activity-Klasse erbt zwischenzeitlich standardmäßig von der Klasse AppCompatActivity. In unserem Falle erbt die Klasse CounterActivity von der Super-Klasse AppCompatActivity:  
**CounterActivity extends AppCompatActivity**



API Auszug: Klasse ActionBarActivity

```

CounterActivity
- swCounter: CowCounter
- brownCounter: CowCounter
- bt_addCow_sw:Button
- bt_removeCow_sw:Button
- tv_cowCounter_sw:TextView
- bt_addCow_brown:Button
- bt_removeCow_brown:Button
- tv_cowCounter_brown:TextView

+ CounterActivity()
# onCreate(Bundle savedInstanceState)
- beobachte(): View.OnClickListener
- updateCowGui(Textview tv_cowCount,
                CowCounter theCow,
                int tv_xml_counter,
                int bar, String farbe)
- showToast(CharSequence pMessage)
    
```

UML-Klasse: CounterActivity.java

*Controller-Klasse → CounterActivity*

Öffnen Sie die Datei CounterActivity.java.

Die *CounterActivity* stellt mit der Methode:  
 → `onCreate(Bundle savedInstanceState)`

beim Starten der Anwendung sicher, dass die Benutzeroberfläche initialisiert und angezeigt wird.

Entsprechend den Vorgaben (Anforderungen) der nebenstehend angezeigten UML-Klasse, werden wir das in den kommenden Schritten tun.

```

1 package com.example.chrissi.cowcounterapp;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6
7 public class CounterActivity extends AppCompatActivity {
8
    
```

Vorher

*Deklaration der CowCounter-Objekte.*

Wir fügen die Assoziationen ein und erzeugen dazu zwei Zähler Objekte vom Typ CowCounter.

Ein Zähler-Objekt, um die schwarz-weißen Kühe zu zählen zu können und ein weiteres Zähler-Objekt, um die braunen Kühe zu zählen.

Fügen Sie dazu die folgenden Assoziationen unterhalb der Klassendeklaration ein.

```
private CowCounter swCounter = new CowCounter();
```

```

1 package com.example.chrissi.cowcounterapp;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5
6 public class CounterActivity extends AppCompatActivity {
7     //Zähler für die Kuehe
8     private CowCounter swCounter = new CowCounter();
9     private CowCounter brownCounter = new CowCounter();
10

```

Nachher

```
private CowCounter brownCounter = new CowCounter();
```

```

11 //Komponenten fuer die Schwarz-Weißen Kuehe
12 private Button bt_addCow_sw;
13 private Button bt_removeCow_sw;
14 private TextView tv_cowCount_sw;
15
16 //Komponenten fuer die Braunen Kuehe
17 private Button bt_addCow_brown;
18 private Button bt_removeCow_brown;
19 private TextView tv_cowCount_brown;
20

```

Komponenten deklarieren

```

private CowCounter brownCounter = new
? android.widget.Button? Alt+Eingabe
//Komponenten fuer die Schwarz-Weißen
private Button bt_addCow_sw;
private Button bt_removeCow_sw;
private TextView tv_cowCount_sw;

```

Import-Anweisungen einfügen

```

14 //Komponenten fuer die Schwarz-Weißen Kuehe
15 private Button bt_addCow_sw;
16 private Button bt_removeCow_sw;
17 private TextView tv_cowCount_sw;
18
19 //Komponenten fuer die Braunen Kuehe
20 private Button bt_addCow_brown;
21 private Button bt_removeCow_brown;
22 private TextView tv_cowCount_brown;

```

Gewünschtes Ergebnis

```

24 @Override
25 protected void onCreate(Bundle savedInstanceState) {
26     //Initialisiert die View (XML-Datei)
27     super.onCreate(savedInstanceState);
28     setContentView(R.layout.activity_counter);
29 }

```

onCreate-Methode

*Komponenten deklarieren.*

Wir deklarieren die +/- Schaltflächen und die TextView-Komponente für den Zähler der schwarz-weißen und braunen Kühe.

Übernehmen Sie die folgenden Deklarationen für die schwarz-weißen Kühe und ergänzen Sie die Deklarationen für die braunen Kühe, wie nebenstehend angezeigt. Da die Import-Anweisungen für die Klasse Button und TextView noch fehlen werden die Klassennamen noch rot angezeigt.

Eingabehilfe:

```
private Button bt_addCow_sw;
private Button bt_removeCow_sw;
private TextView tv_cowCount_sw;
```

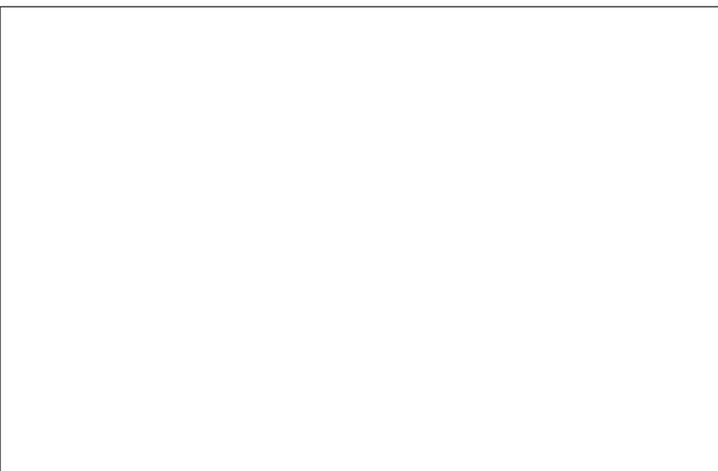
Fügen Sie die fehlenden import-Anweisung für die Button-und TextView-Klasse ein. Klicken Sie dazu jeweils auf den roten Klassennamen an und wählen Sie die Tastenkombination ALT + ENTER auf Ihrer Tastatur.

Kontrollieren Sie anschließend die Import-Anweisungen oberhalb der Klassendeklaration:

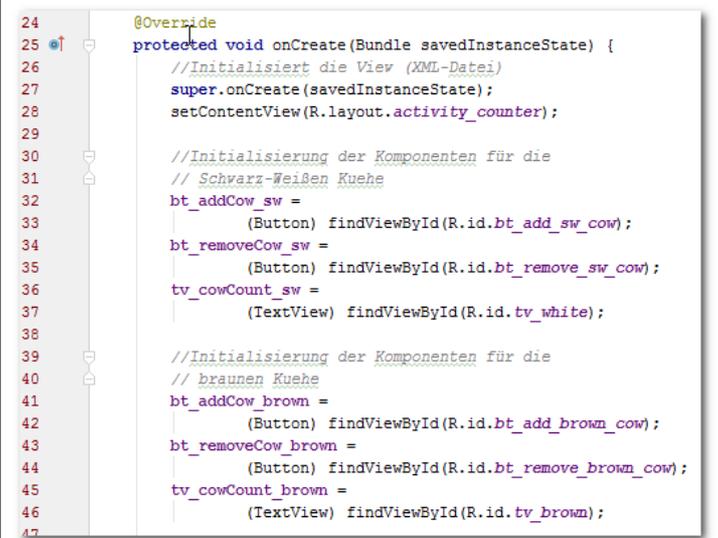
```
import android.widget.Button;
import android.widget.TextView;
```

*Die onCreate-Methode.*

Um die Benutzeroberfläche anzuzeigen wird in der onCreate-Methode die beim Aufruf des Activity-Objektes benötigte id der Benutzeroberfläche (XML-Datei) übermittelt und in einem Objektbaum entfaltet.



Vorher



Nachher

Eingabehilfe: für schwarz-weiße Kühe

```

bt_addCow_sw =
    (Button) findViewById(R.id.bt_add_sw_cow);
bt_removeCow_sw =
    (Button) findViewById(R.id.bt_remove_sw_cow);
tv_cowCount_sw =
    (TextView) findViewById(R.id.tv_white);
    
```

Eingabehilfe: für braune Kühe

```

bt_addCow_brown =
    (Button) findViewById(R.id.bt_add_brown_cow);
bt_removeCow_brown =
    (Button) findViewById(R.id.bt_remove_brown_cow);
tv_cowCount_brown =
    (TextView) findViewById(R.id.tv_brown);
    
```

Genau das geschieht mit dem Methodenaufruf

```
setContentView(...)
```

R ist eine Klasse deren Aufgabe es ist, alle Elemente der Layouts und anderer XML-Dateien zu verwalten, u.a. um diese in Java verfügbar zu machen.

Ergänzen Sie dazu die fehlenden Kommentare, wie nebenstehend angezeigt.

Ausstattung der onCreate-Methode.

Wir müssen sicherstellen, dass Komponenten, deren Inhalte gelesen bzw. in die geschrieben werden soll, zuvor initialisiert werden. Wir ergänzen dazu den Quellcode, wie nebenstehend angezeigt.

Erklärung:

```
bt_addCow_sw =
    (Button) findViewById(R.id.bt_add_sw_cow);
```

- **bt\_addCow\_sw:**  
Ist u.a. ein Klassenattribut der Activity-Klasse vom Typ Button (siehe Deklaration).
- **(Button):**  
Der Cast stellt sicher, dass die zugewiesene Komponente dem Typ entspricht.
- **findViewById(int)**  
Sucht den Parameterwert anhand der id. Als Parameter wird ein int-Wert erwartet.
- **R.id.bt\_add\_sw\_cow**  
R liefert zum String bt\_add\_sw\_cow den entsprechenden int-Wert zurück. Den entsprechenden Schlüsselwert.

```
(TextView) findViewById(R.id.tv_brown);
```

```

48 //Listener für die Button-Komponente addCow
49 bt_addCow_sw.setOnClickListener(beobachte());
50 bt_addCow_brown.setOnClickListener(beobachte());
51
52 //Listener für die Button-Komponente removeCow
53 bt_removeCow_sw.setOnClickListener(beobachte());
54 bt_removeCow_brown.setOnClickListener(beobachte());
    
```

Listener

Eingabehilfe:

Listener für die Button-Komponenten addCow

```
bt_addCow_sw.setOnClickListener(beobachte());
```

```
bt_addCow_brown.setOnClickListener(beobachte());
```

Eingabehilfe:

Listener für die Button-Komponenten removeCow

```
bt_removeCow_sw.setOnClickListener(beobachte());
```

```
bt_removeCow_brown.setOnClickListener(beobachte());
```

```

59 ##### Sonstige Methoden #####
60 #####
61 //Beobachter für alle Listeneraufrufe
62 @ private View.OnClickListener beobachte(){
63     return new View.OnClickListener(){
64
65
66     @Override
67     public void onClick(View v) {
68         //Deklaration und Initialisierung
69         // eines neuen noch leeren Strings
70         String farbe = new String();
71         //Prüfung welche Schaltfläche geklickt wurde
72         // EINGABE: Klick auf die Schaltfläche
73         // FALL 1: bt_addCow_sw
74         if(v == bt_addCow_sw){
75             //VERARBEITUNG
76             //Erhöht den Zähler um 1
77             swCounter.addCow();
78
79             //Setzt die Eigenschaft Farbe
80             farbe = "sw";
81
82             //AUSGABE
83             //Führt die Aktualisierung der GUI durch
84             updateCowGui(tv_cowCount_sw,
85                 swCounter,
86                 R.id.tv_white,
87                 R.id.relativ_layout_bar_black,
88                 farbe);
89         }
90     }
91 }
    
```

Fall 1

Listener in der onCreate-Methode.

Ein Listener ist wie ein Fühler der Veränderungen auf der Benutzeroberfläche registriert und in Form eines Impulses an das System weiterreicht.

Wir fügen dem editierbaren Objekt → bt\_addCow\_sw mit dem Methodenaufruf

```
bt_addCow_sw.setOnClickListener(beobachte());
```

den Listener hinzu. Als Parameter übergeben wir den Methodenaufruf → beobachte().

Der Methodenaufruf wird rot angezeigt da wir Sie noch nicht implementiert haben. Wir wenden uns also im nächsten der Implementierung genau dieser Methode.

Implementieren Sie den Methodenaufruf wie nebenstehend angezeigt.

Beobachter (Fühler) für alle Schaltflächen.

Da alle Schaltflächen ein OnClickListener-Objekt benötigen kapseln wir die Funktionalität in der Hilfsmethode → beobachte(). Auf diese Weise können wir das OnClickListener-Objekt dann verwenden.

Wir deklarieren dazu im Ersten Schritt die Methode beobachte() als Methode mit Rückgabewert.

```
private View.OnClickListener beobachte(){
    //Hier fehlt Quellcode
}
```

Der Rückgabewert ist ein Objekt vom Typ View.OnClickListener. Wir implementieren innerhalb der Methode → beobachte() die Erzeugung dieses Objektes und geben es gleichzeitig zurück. Dazu implementieren wir im nächsten Schritt die folgende Anweisung:

```
return new View.OnClickListener(){
    //Hier fehlt Quellcode
}
```

```

90 //EINGABE: Klick auf die Schaltfläche
91 //FALL 2: bt_removeCow_sw
92 }else if(v == bt_removeCow_sw ){
93 //VERARBEITUNG
94     swCounter.removeCow();
95
96 //Setzt die Eigenschaft Farbe
97     farbe = "sw";
98
99 //Ausgabe
100 //Führt die Aktualisierung der GUI durch
101     updateCowGui(tv_cowCount_sw,
102                 swCounter,
103                 R.id.tv_white,
104                 R.id.relativ_layout_bar_black,
105                 farbe);
106

```

Fall 2

```

107 //EINGABE: Klick auf die Schaltfläche
108 //FALL 3: bt_removeCow_brown
109 }else if(v == bt_removeCow_brown ){
110 //VERARBEITUNG
111     brownCounter.removeCow();
112
113 //Setzt die Eigenschaft Farbe
114     farbe = "brown";
115
116 //AUSGABE
117 //Führt die Aktualisierung der GUI durch
118     updateCowGui(tv_cowCount_brown,
119                 brownCounter,
120                 R.id.tv_brown,
121                 R.id.relativ_layout_bar_brown,
122                 farbe);

```

Fall 3

```

124 //EINGABE: Klick auf die Schaltfläche
125 //FALL 4: bt_addCow_brown
126 }else if(v == bt_addCow_brown ){
127 //VERARBEITUNG
128     brownCounter.addCow();
129
130 //Setzt die Eigenschaft Farbe
131     farbe = "brown";
132
133 //AUSGABE
134 //Führt die Aktualisierung der GUI durch
135     updateCowGui(tv_cowCount_brown,
136                 brownCounter,
137                 R.id.tv_brown,
138                 R.id.relativ_layout_bar_brown,
139                 farbe);
140
141 //Ansonsten: keine Aktivität
142 }else {
143     finish();
144 } //end else if
145
146 //end onClick()
147 } //end View.OnClickListener
148 } //end beobachte()
149

```

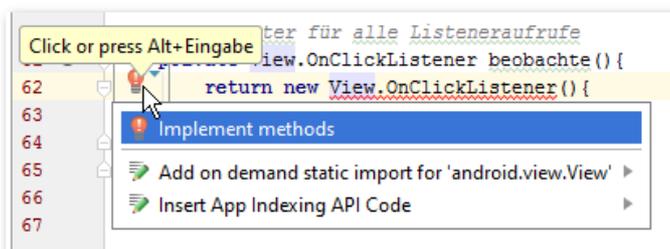
Fall 4

};

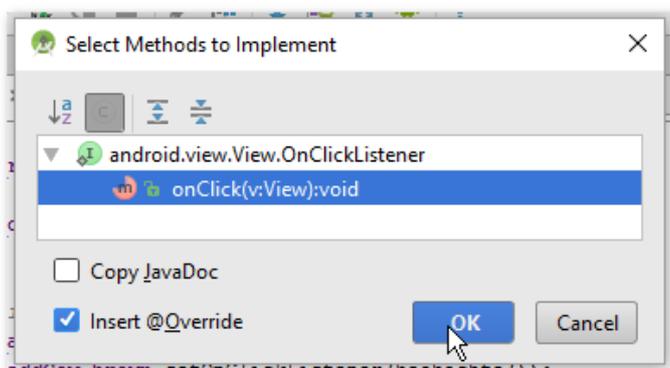
### View.OnClickListener

Ist eine Interface-Klasse. Ein Interface ist so etwas wie eine Vorlage. Verhaltensweisen die im Interface deklariert sind, sollten implementiert werden, da sie eine zwingende Verhaltensweise eines Objektes darstellen.

Klicken Sie dann auf das rot unterstrichene Objekt View:



Mit einem weiteren Klick auf die rote Glühbirne wählen Sie im Kontext-Menü die Option → Implement Methods oder nutzen Sie dazu alternativ die Tastenkombination Alt + Enter.



Klicken Sie weiter auf die Schaltfläche → OK um die Deklaration der onClick-Methode einzufügen.

### Ergebnis:

```

@Override
public void onClick(View v) {
    //Hier fehlt Quellcode
}

```

Wir deklarieren und initialisieren einen zu Beginn noch leeren String für die Farbe:

public static interface

# View.OnClickListener

android.view.View.OnClickListener

► Known Indirect Subclasses

[CharacterPickerDialog](#), [KeyboardView](#), [QuickContactBadge](#), [SearchOrbView](#), [SpeechOrbView](#)

## Class Overview

Interface definition for a callback to be invoked when a view is clicked.

Auszug der API: View.OnClickListener

Wir wenden bei der Umsetzung zwei weitere Prinzipien der Informatik an. Unser Fokus: Die Prinzipien „Zerlegung“ und „Wiederverwendung“.

### Zerlegung:

Ist eine der wichtigsten Hilfen in der Informatik bei der Lösung komplexer Probleme. Man unterteilt große, komplexe Probleme in kleine, strukturierte Teilprobleme (→ Hilfsmethoden) und setzt diese in Quellcode um.

Wenn alle Teilprobleme umgesetzt sind, ist damit auch das große, komplexe Problem gelöst. → divide and conquer (→ teile und herrsche)

### Wiederverwendung:

Aus der o.g. Zerlegung ergibt sich ein weiterer Vorteil. Die Auslagerung von Quellcode in Methoden und Hilfsmethoden ermöglicht die Wiederverwendung des Quellcodes an anderer Stelle.

Wir zerlegen also im ersten Schritt unser logisches Problemchen:

### EVA-Prinzip

#### //Eingabe

1. Klick auf eine der vier +/- Schaltflächen

#### //Verarbeitung

2. erhöhen bzw. minimieren des aktuellen Zählers um 1.
3. Setzen der Farbe

#### //Ausgabe

4. Aktualisierung der GUI

```
String farbe = new String();
```

Wir implementieren im Anschluss daran die Kontrollstruktur ELSE IF, um zu prüfen welche Schaltfläche angeklickt wurde:

```
if(v == bt_addCow_sw ){
    //Hier fehlt Quellcode für Fall1
}else if(v == bt_removeCow_sw ){
    //Hier fehlt Quellcode für Fall2
}else if(v == bt_removeCow_brown ){
    //Hier fehlt Quellcode für Fall3
}else if(v == bt_addCow_brown ){
    //Hier fehlt Quellcode für Fall4
}else {
    finish();
}
```

Implementieren Sie für Fall 1:

```
bt_addCow_sw
```

```
swCounter.addCow();

farbe = "sw";

updateCowGui(tv_cowCount_sw,
             swCounter,
             R.id.tv_white,
             R.id.relativ_layout_bar_black,
             farbe);
```

Ergänzen Sie zum besseren Verständnis die noch fehlenden Kommentare.

Implementieren Sie für Fall 2:

```
bt_removeCow_sw
```

```
swCounter.removeCow();

farbe = "sw";

updateCowGui(tv_cowCount_sw,
             swCounter,
             R.id.tv_white,
             R.id.relativ_layout_bar_black,
             farbe);
```

Implementieren Sie für Fall 3:

```
bt_removeCow_brown
```

```
brownCounter.removeCow();

farbe = "brown";

updateCowGui(tv_cowCount_brown,
             brownCounter,
             R.id.tv_brown,
             R.id.relativ_layout_bar_brown,
```

Ansonsten soll die Aktivität geschlossen werden.

```
farbe);
```

Implementieren Sie für Fall 4:

```
bt_addCow_brown
```

```
brownCounter.addCow();
```

```
farbe = "brown";
```

```
updateCowGui(tv_cowCount_brown,
             brownCounter,
             R.id.tv_brown,
             R.id.relativ_layout_bar_brown,
             farbe);
```

```
151 //Initialisiert, wandelt, aktualisiert und erzeugt
152 // die Counterausgabe für schwarz-weiße und braune
153 // Kühe auf der GUI
154 private void updateCowGui(TextView pTv_cowCount,
155                          CowCounter pTheCow,
156                          int pTv_xml_counter,
157                          int pBar,
158                          String pFarbe){
159
160 //Initialisiert die TextView mit dem
161 // übermittelten xml-Parameter
162 pTv_cowCount = (TextView) findViewById(pTv_xml_counter);
163
164 //Initialisiert einen String mit dem
165 // aktuellen Zählerstand
166 String mCowCountText = Integer
167     .toString(pTheCow.getCounter());
168
169 //Aktualisiert den angezeigten
170 // Zählerstand auf der GUI
171 pTv_cowCount.setText(mCowCountText);
```

*Aktualisierung der Benutzeroberfläche.*

Die Hilfsmethode initialisiert, wandelt und aktualisiert die Zählerausgabe und die Benutzeroberflächenelemente (Säulen) für schwarz-weißen und braunen Kühe auf der Benutzeroberfläche.

Die Methode übermittelt die aktuellen Eigenschaftswerte mittels der Parameterattribute in der Methodendeklaration.

```
private void updateCowGui(TextView pTv_cowCount,
                          CowCounter pTheCow,
                          int pTv_xml_counter,
                          int pBar,
                          String pFarbe){
    //Hier fehlt Quellcode
}
```

```
172
173 //Ermittlung (mit getResources().getDisplayMetrics())
174 // der Auflösung (density -->float)
175 // des aktuellen Gerätes
176 float density = getResources()
177     .getDisplayMetrics().density;
178
179 if(pFarbe.equals("sw")) {
180 //Wir ergänzen das Balkendiagramm
181 // (Relatives Layout) für die
182 // Schwarz-Weißen Kühe
183 RelativeLayout black_bar
184     = (RelativeLayout) findViewById(pBar);
185
186 //Wir berechnen die neue Höhe für
187 // den Schwarzen Balken (Konstante 5 je Kuh).
188 // Erweiterung: mit density multiplizieren
189 // und in eine ganze Zahl casten.
190 int blackBarHeight
191     =(int) (5 * pTheCow.getCounter() * density);
```

Im Folgenden werden wir die Implementierung schrittweise umsetzen. Ergänzen Sie zum besseren Verständnis auch die nebenstehend angezeigten Kommentare.

Wir initialisieren die übermittelte TextView anhand der ebenfalls übermittelten id für die TextView-Komponente des Zählers:

```
pTv_cowCount
= (TextView) findViewById(pTv_xml_counter);
```

Dann initialisieren wir ein lokales Attribut mit dem aktuellen Zählerstand:

```
String mCowCountText = Integer
    .toString(pTheCow.getCounter());
```

```

192
193 //Wir weisen dem schwarzen Balken
194 // die neue Höhe (height) zu (--> getLayoutParams())
195 black_bar.getLayoutParams().height = blackBarHeight;
196 }else{
197 //Wir ergänzen das Balkendiagramm
198 // (Relative Layout) für die Braune Kühe
199 RelativeLayout brown_bar
200     = (RelativeLayout) findViewById(pBar);
201
202 //Wir berechnen die neue Höhe für den
203 // Schwarzen Balken (Konstante 5 je Kuh).
204 // Erweiterung: mit density multiplizieren
205 // und in eine Ganze Zahl casten.
206 int brownBarHeight
207     =(int) (5 * pTheCow.getCounter() * density);
208
209 //Wir weisen dem schwarzen Balken die
210 // neue Höhe (height) zu (--> getLayoutParams())
211 brown_bar.getLayoutParams().height
212     = brownBarHeight;
213 }//end if else
214
215 }//end updateCowGui()
216 }//end Class

```

Und aktualisieren damit den angezeigten Zählerstand auf der Benutzeroberfläche (GUI):

```
pTv_cowCount.setText(mCowCountText);
```

Wir ermitteln unterstützend die Auflösung des aktuellen Gerätes:

```
float density
    = getResources()
      .getDisplayMetrics().density;
```

Wir nutzen die Kontrollstruktur IF ELSE um zu identifizieren welcher Teil (Links oder Rechts) der Benutzeroberfläche aktualisiert werden soll. Für den Fall, dass der Attributwert für Farbe dem Wert für schwarz-weiß (sw) entspricht:

```
if(pFarbe.equals("sw")) {
    //Hier fehlt Quellcode für den JA-FALL(Links)
}else{
    //Hier fehlt Quellcode für
    //den SONST-FALL(Rchts)
}
```

Für den JA-FALL (Links).

Wir initialisieren das RelativeLayout für unseren schwarzen Balken anhand der übermittelten id:

```
RelativeLayout black_bar
    = (RelativeLayout) findViewById(pBar);
```

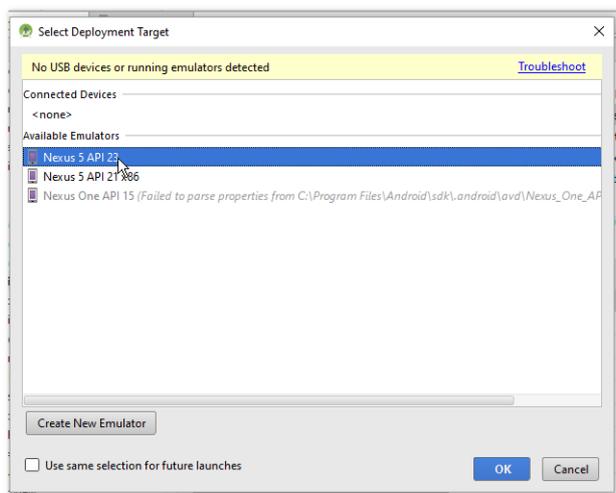
Wir berechnen die neue Höhe für den Schwarzen Balken und nutzen die Konstante 5 je neuer Kuh. Um die Veränderung der Höhe verhältnismäßige zum Display zu gestalten multiplizieren wir den ermittelten Wert für die Auflösung → density und wandeln den Wert im gleichen Schritt um, in eine Ganze Zahl (int, Cast):

```
int blackBarHeight
    =(int) (5 * pTheCow.getCounter() * density);
```

Dann weisen abschließend den Wert zu und aktualisieren damit den schwarzen Balken auf der Benutzeroberfläche:

```
black_bar.getLayoutParams().height
    = blackBarHeight;
```

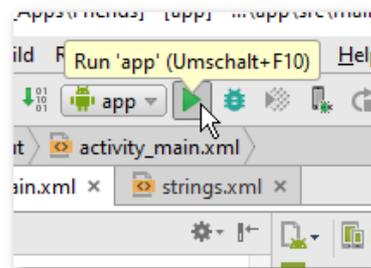
Ergänzen Sie den Quellcode für den SONST-FALL (Rechts) für braune auf die gleiche Art und Weise, passen Sie dazu die Attributnamen an.



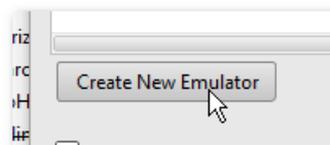
Alternativ → Create New Emulator:  
 Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit API 15 (SanwichIceCream) zu erzeugen:

*Testen der Anwendung.*

Wir starten nun den Emulator.



Emulator:  
 Der Emulator simuliert im vorliegenden Fall ein virtuelles Mobiltelefon vom Typ → Nexus 5 API 23.



Testen Sie alle Funktionen der App!  
**Gratulation!**

*Der Emulator öffnet sich.*

Beim ersten öffnen kann das einen Moment dauern.

Ziehen Sie dann das auf dem Display erscheinende Schösschen mit gedrückter linken Maustaste senkrecht nach oben.

Wenn Sie nicht ungeduldig werden, startet der Emulator die App nach Abschluss des Built-Prozesses von selbst.

Im Ergebnis sollte die Benutzeroberfläche erscheinen.

Hinweis:  
 Software ist nie optimal. Wir befinden uns in einem Kreislauf → Softwareentwicklungszyklus.

Eine „Never ending Story“ der Optimierung. Falls Sie also Verbesserungsmöglichkeiten wahrnehmen, sollten Sie in Erwägung ziehen die Optimierungen durchzuführen.

