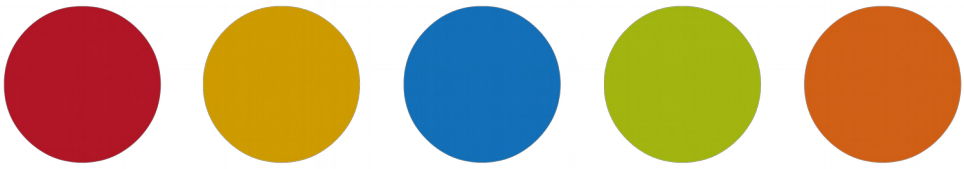
# Wortspiele-App

## Skript 2016

Konfigurations- und Schulungsunterlagen

| | |
|---|---|
| Schulung: | Didaktische Ansätze zur Android-Programmierung |
| Referent: | Christine Janischek |

Stand: 7. Jun 2016



# Inhaltsverzeichnis

# 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



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]
Künneth, 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

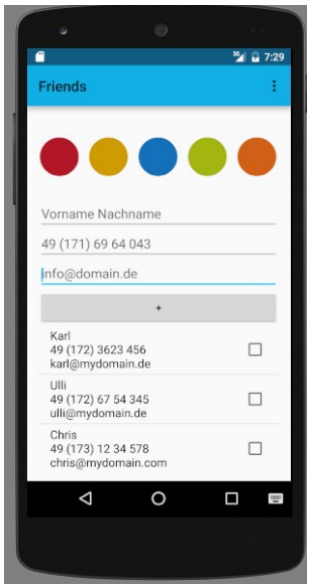
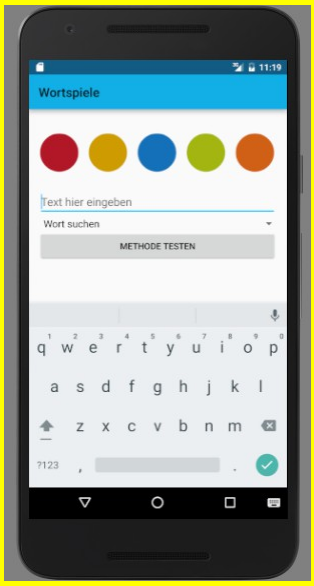
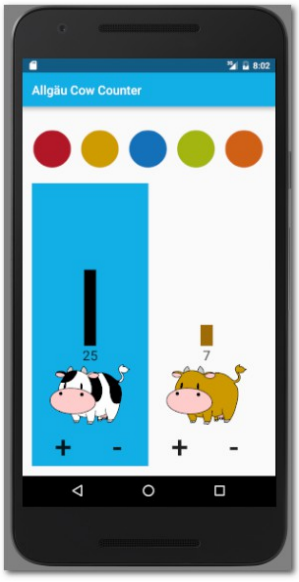
[AZF00]
Azzola, Francesco „Android: Build real weather app: JSON, HTTP and Openweathermap", https://www.javacodegeeks.com/2013/06/android-build-real-weather-app-json-http-and-openweathermap.html, 2013, zuletzt getestet am 30.04.2016

# 2 Das Projekt Wortspiele

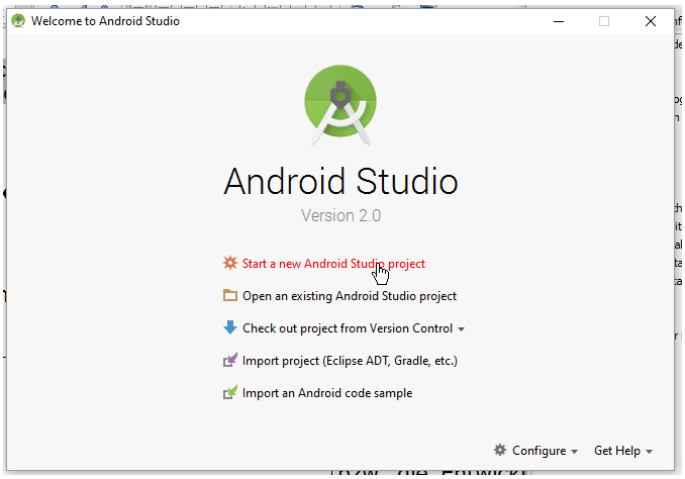
## 2.1 Überblick

Wortspiele App:

Das Projekt soll an einer einfachen Benutzeroberfläche zeigen auf welche Weise die Themen: Stringverarbeitung, API, Algorithmen (Suchen, Ersetzen, Sortieren, Tauschen, Entfernen) und Kontrollstrukturen im Unterricht geübt und angewendet werden können.

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

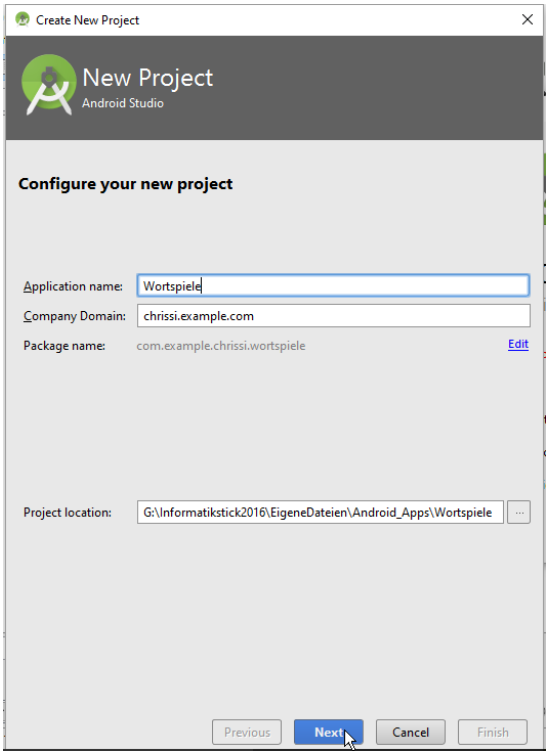
## 2.2 Grundlagen: Projekte erstellen




*Ein Neues Projekt erzeugen.*

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

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

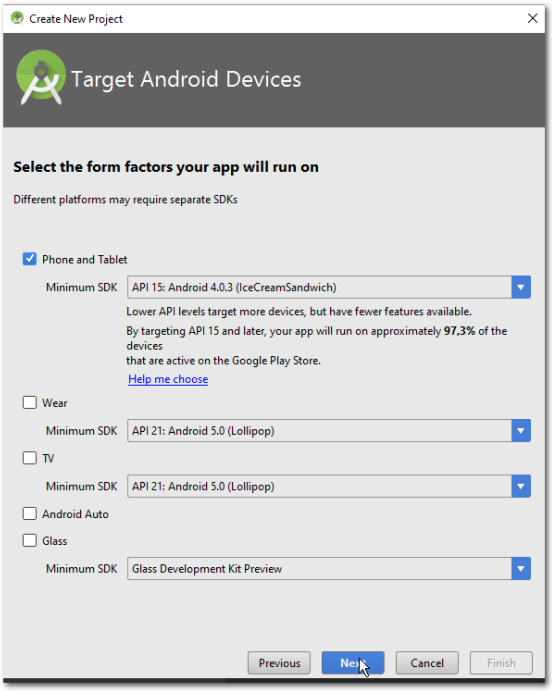



*Legen Sie nun schrittweise die Eigenschaften für Ihr neues Android-Projekt fest.*

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.


Je nach Konfiguration können diese Angaben variieren

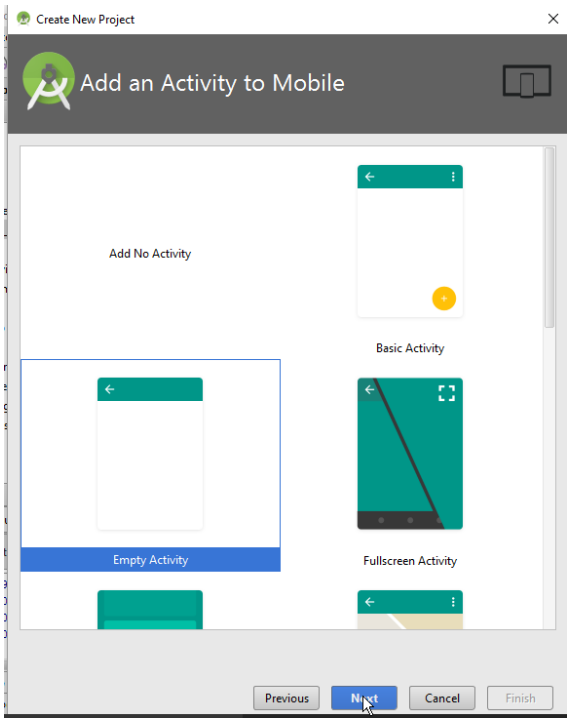



*Laufzeitumgebung unserer Anwendung.*

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.

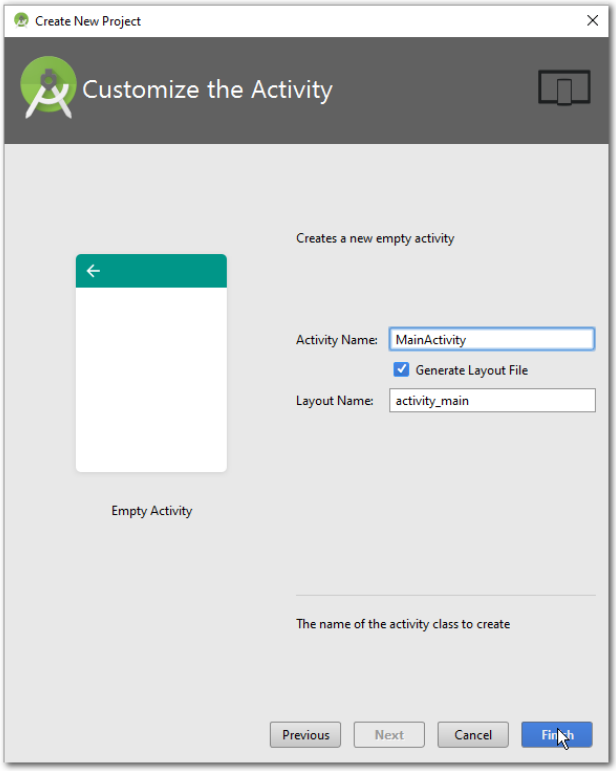



*Aktivität wählen.*

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.*

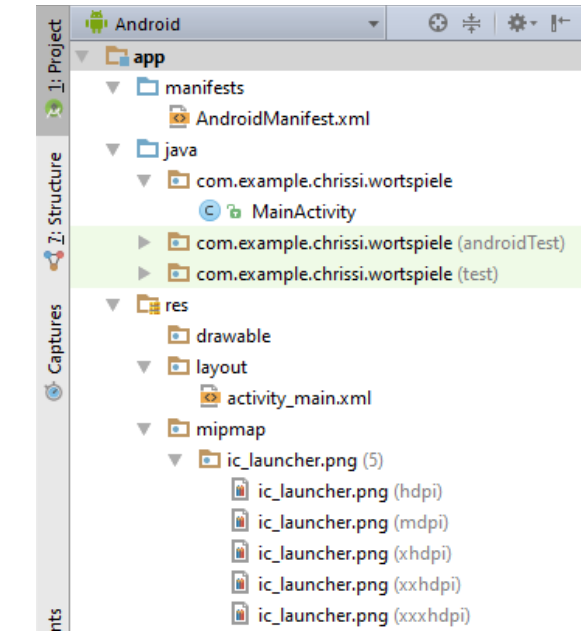
Activities enthalten die Ereignissteuerung für einen bzw. eine ganze Reihe von zusammengehörenden Vorgänge (Interaktionen, Verhaltensweisen) einer App.

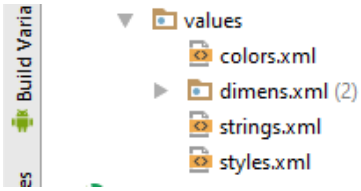
Übernehmen Sie die nebenstehenden Werte und klicken Sie anschließend die Schaltfläche → Finish.

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.

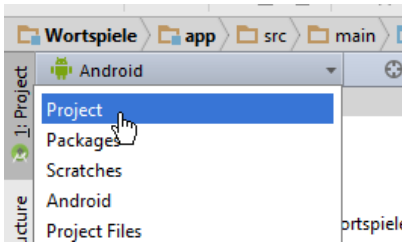


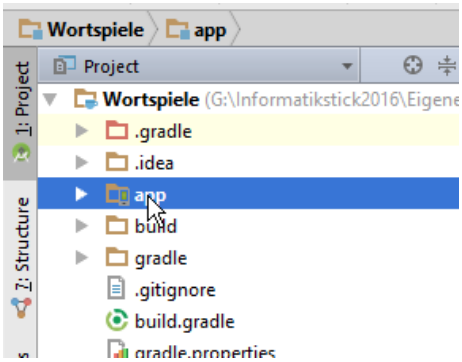

*Projektstruktur am Anfang.*

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 zu wählen:



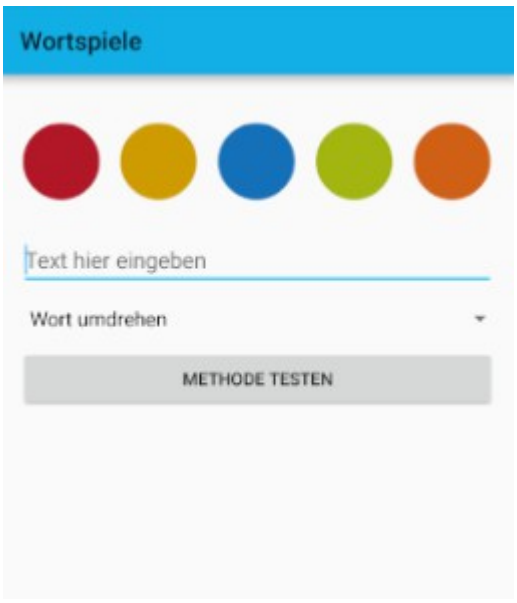

Folgen Sie den nächsten Schritten, um ein ersten Entwurf der Benutzeroberfläche zu erzeugen.



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

### 2.3.1 Benutzeroberfläche

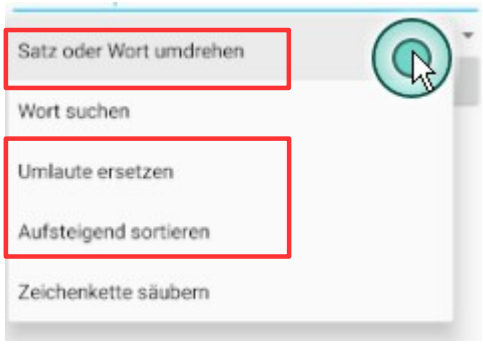
Für die Operationen → umdrehen, → ersetzen und → sortieren sind keine weiteren Eingaben erforderlich, damit ist das folgende Layout ausreichend.

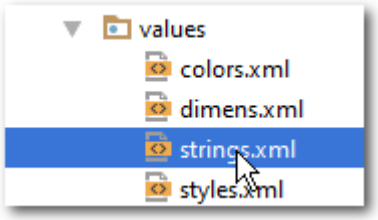



*Erstes Layout.*

Wir werden nun die Benutzeroberfläche für unsere Wortspiele-App erzeugen.

Benutzeroberflächen werden in Android-Apps in der Seitenbeschreibungssprache XML beschrieben.



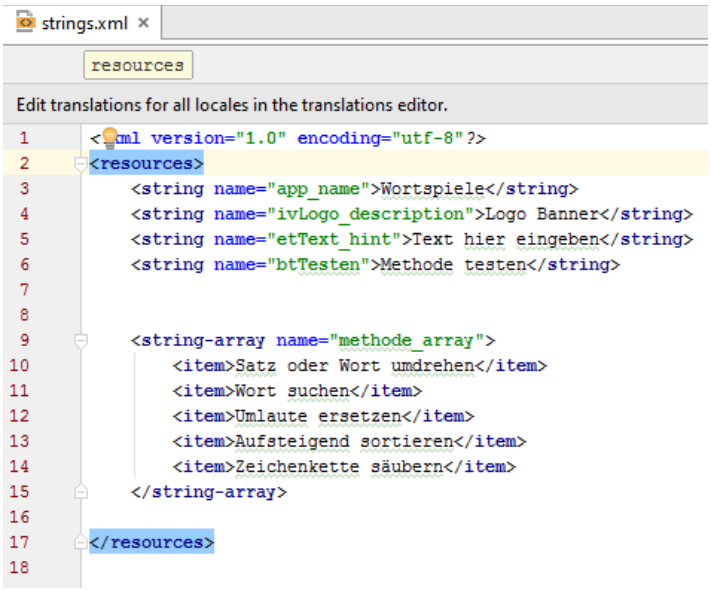

Hinweis:
Aus gutem Grund werden alle Bezeichner der Benutzeroberfläche in eine String-Ressource → (res/values/strings.xml) ausgelagert. Für den Fall, dass Apps in anderen Sprachen verfügbar gemacht werden sollen. Findet der Übersetzer alle benötigten Begriffe in genau einer Datei.

*XML-Deklarationen.*

Dazu definieren wir in einem ersten Schritt alle verwendeten Bezeichnungen für die Komponenten die wir auf unserer Benutzeroberfläche verwenden möchten. Sie sollten in der Datei strings.xml definiert werden.

Öffnen Sie dazu die Datei strings.xml.

Sie finden diese Datei im Unterverzeichnis → app → res → values → strings.xml.




*Bezeichner definieren.*

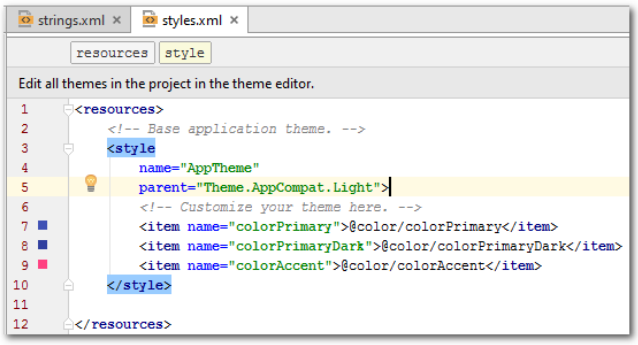
Öffnen Sie die Datei → strings.xml mit einem Doppelklick auf den Dateinamen und ändern Sie die darin enthaltenen Angaben wie nebenstehend angezeigt.

Hinweis:
Vergleichen Sie die definierten Strings mit der Benutzeroberfläche und identifizieren Sie die Bezeichner.

Eingabehilfe:

```
<resources>
    <string name="app_name">Wortspiele</string>
    <string name="ivLogo_description">Logo
Banner</string>
    <string name="etText_hint">Text hier
eingeben</string>
    <string name="btTesten">Methode testen</string>
    <string-array name="methode_array">
        <item>Satz oder Wort umdrehen</item>
        <item>Wort suchen</item>
        <item>Umlaute ersetzen</item>
        <item>Aufsteigend sortieren</item>
        <item>Zeichenkette säubern</item>
    </string-array>
</resources>
```

styles.xml




Sie finden diese Datei im Unterverzeichnis → app → res → values → styles.xml.

*Farben der App ändern.*

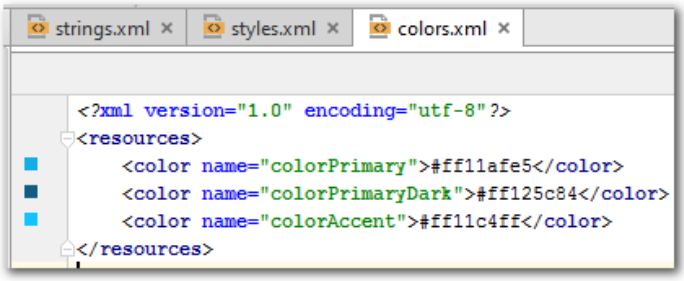
Öffnen Sie dazu die Datei → styles.xml mit einem Doppelklick auf den Dateinamen.

Ändern Sie die Angaben ggf. wie nebenstehend angezeigt

```
<style
        name="AppTheme"
        parent="Theme.AppCompat.Light">
```

Öffnen Sie die Datei → colors.xml mit einem Doppelklick auf den Dateinamen und ändern Sie die Farbcodes, wie nebenstehend angezeigt.

```
<color name="colorPrimary">#ff11afe5</color>
<color name="colorPrimaryDark">#ff125c84</color>
<color name="colorAccent">#ff11c4ff</color>
```



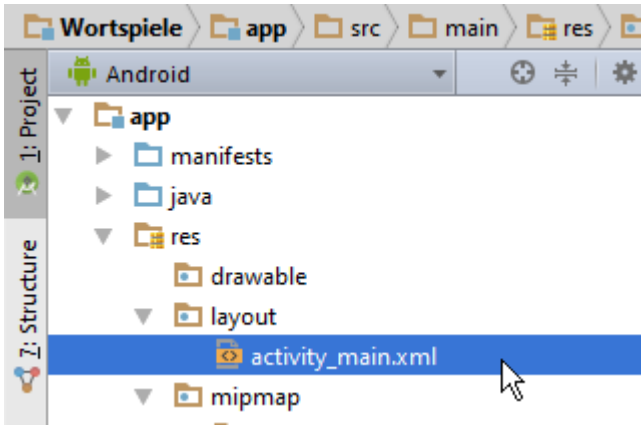

colors.xml

Sie finden diese Datei im Unterverzeichnis
→ app → res → values → colors.xml.

Die styles-Datei holt sich dabei an entsprechender Stelle die Farbangaben aus der colors-Datei.

Hinweis:
Themes enthalten Layoutvorgaben. Diese können wir nach belieben anpassen. Auch wenn unsere Anwendung nur eine Beispielanwendung ist soll sie trotzdem schick sein. Alle generellen Angaben zum Format und Aussehen unserer App werden in der Datei → colors.xml und → styles.xml definiert:

1. Aussehen: res/values/styles.xml
2. Farben: res/values/colors.xml

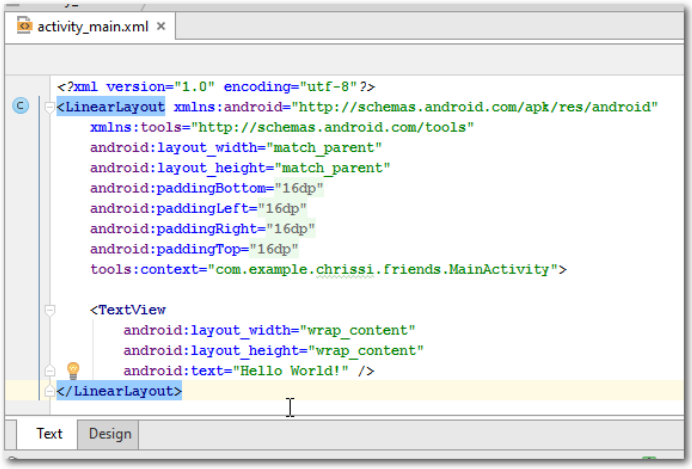



*Layout erstellen.*

Wir werden nun das Layout unserer eigentlichen Benutzeroberfläche erstellen.

Öffnen Sie dazu das Verzeichnis
app → res → layout

Öffnen Sie die Datei → activity_main.xml mit einem Doppelklick auf den Dateinamen.



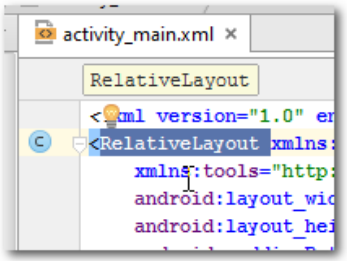

Das Lineare Layout (vertikal):
Die in einem vertikalen Linearen Layout platzierten Komponenten werden untereinander angeordnet.

Das Lineare Layout (horizontal):
Die in einem horizontalen Linearen Layout platzier-

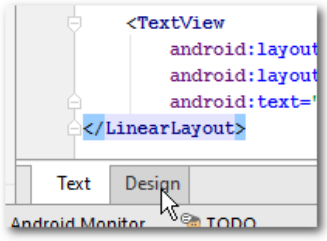
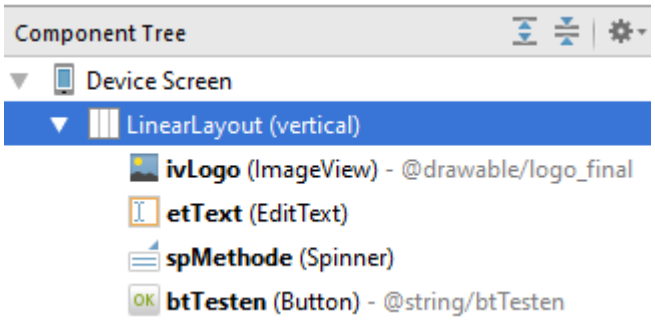
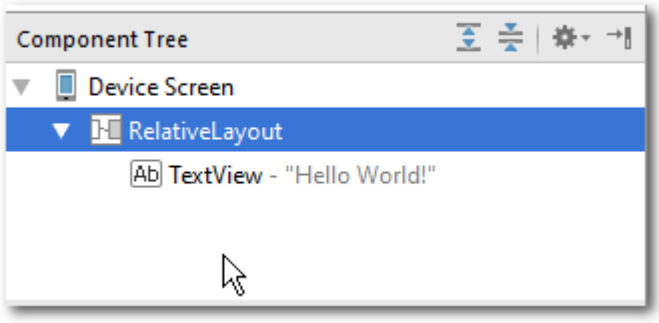
*Layout definieren.*

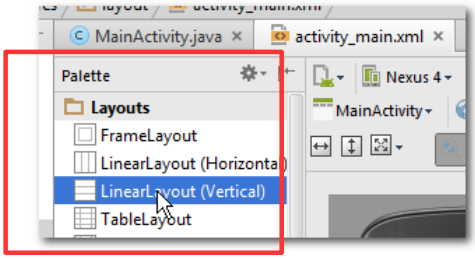
Wir ersetzen das → RelativeLayout durch ein → LinearLayout.

Übernehmen Sie die Angaben, wie nebenstehend angezeigt.

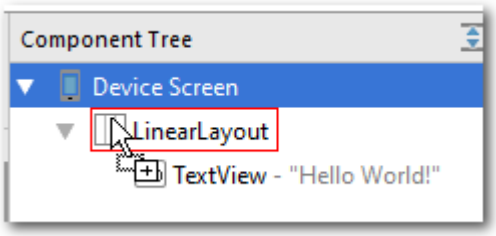



Das Relative Layout:
Die in einem relativen Layout enthaltenen Komponenten werden immer in Abhängigkeit seiner direkt benachbarten Komponenten betrachtet. Deshalb er-

| | |
|---|---|
| ten Komponenten werden nebeneinander angeordnet. | folgt die Beschreibung der Platzierung auch in Abhängigkeit der direkt benachbarten Komponenten. |
|  <br>Designer | *In den Design-Modus wechseln.*<br><br>Um das Design zu erstellen nutzen wir den Oberflächendesigner.<br><br>Klicken Sie dazu auf den Reiter → Design unterhalb des angezeigten XML-Quellcodes.<br><br>Hinweis:<br>Die Anwendung besitzt ähnlich, wie in Eclipse der 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. |
|  <br>Gewünschtes Ergebnis | *Vorgehensweise: Component Tree.*<br><br>1. Layouts (wenn nötig) schachteln<br>2. Komponenten im Layout platzieren<br>3. Komponenteneigenschaften definieren<br><br>Nun folgen dazu die Änderungen im aktuellen Komponenten-Baum um das nebenstehende **gewünschte Ergebnis** zu erzeugen. |
|  <br>Aktueller Komponenten-Baum | *Der Komponenten-Baum.*<br><br>Im oberen, rechten Frame-Fenster wird der Komponenten-Baum (Component Tree) angezeigt.<br><br>Als Komponenten werden alle Elemente einer Benutzeroberfläche bezeichnet.<br><br>Die Grundlage jeder Benutzeroberfläche sind die Layouts.<br><br>Das Standard-Layout ist das → Relative Layout. |




Fenster Palette

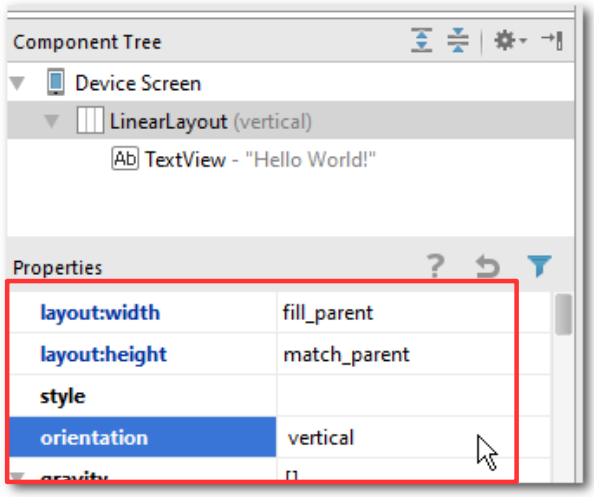



Fenster Component Tree

*LinearesLayout (Vertical) verwenden.*

Klicken Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf 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.



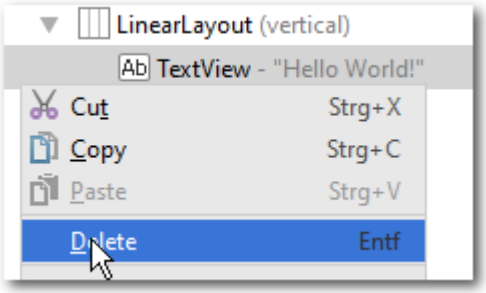

Fenster Component Tree und Properties

*Eigenschaften des Layouts ändern.*

Klicken Sie dazu im Fenster → Component Tree auf das → LineareLayout (vertical).

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

| | |
|---|---|
| layout:width: | `fill_parent` |
| layout:height: | `match_parent` |
| orientation: | `vertical` |

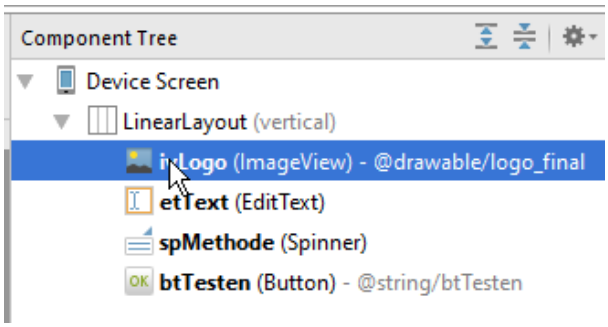



Fenster Component Tree

*Komponente löschen.*

Löschen Sie die nicht benötigte TextView-Komponente → Hello World!.

Klicken Sie dazu die Komponente im Fenster → Component Tree mit der rechten Maustaste an und wählen Sie im Kontext-Menü die Option → Delete.

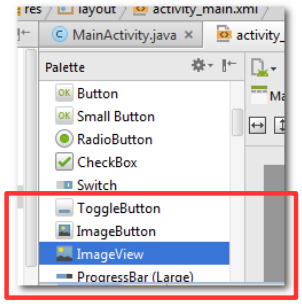



Fenster Component Tree

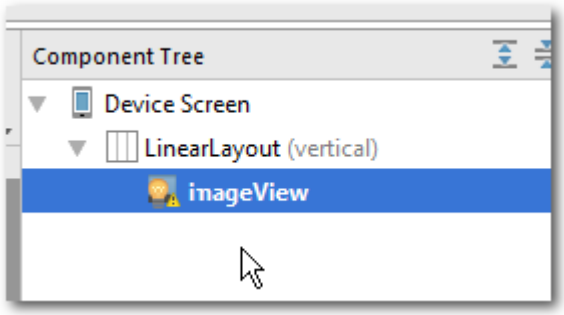
*Komponenten platzieren.*

Alle Komponenten werden wir untereinander in das LineareLayout integrieren. Anschließend werden wir für jede Komponente die Eigenschaften festlegen.

Gehen Sie auf gleiche Weise vor. Suchen Sie in der Palette die Komponente und ziehen Sie dazu diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree.




Fenster Palette




Fenster Component Tree

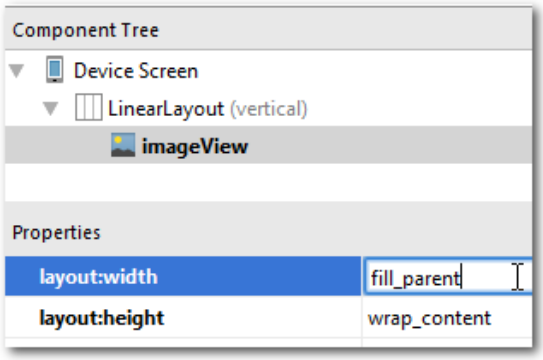
*Platz für das Logo schaffen.*

Um zu einem späteren Zeitpunkt ein Logo angezeigt zu bekommen, fügen wir die *ImageView Komponente einfügen.*

Hier ist eine Komponente vom Typ ImageView nötig. Eine → ImageView-Komponente (Platzhalter für ein Bild oder eine Grafik).

Wählen Sie dazu die ImageView-Komponente im linken Frame-Fenster → Palette aus.

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

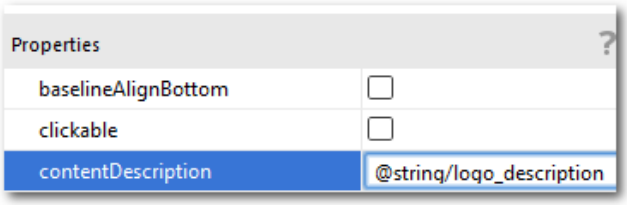



Fenster Component Tree und Properties

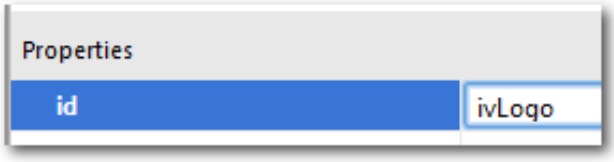
*Eigenschaften für die ImageView-Komponente festlegen.*

Im rechten, unteren Frame-Fenster unterhalb des → Component Tree werden die Eigenschaften (Properties) der Aktuell angeklickten Komponente angezeigt. Um die Eigenschaften für die gerade eingefügte ImageView-Komponente zu verändern müssen Sie diese im → Component Tree anklicken.

Nutzen Sie dann die vertikale Bildlaufleiste im




Fenster Properties

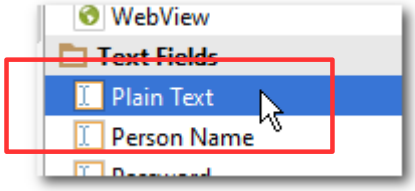



Fenster Properties

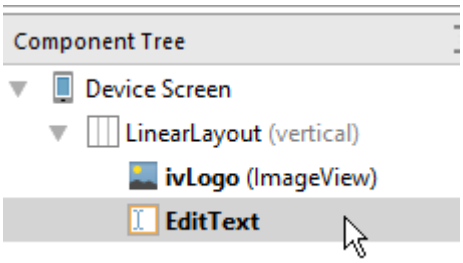
Fenster → Properties, um die Eigenschaft für die layout:width, → contentDescription und → id, wie nebenstehend angezeigt, ändern zu können.

| | |
|---|---|
| layout:width: | fill_parent |
| contentDescription: | @string/logo_description |
| id: | ivLogo |

Die Anpassung der restlichen Eigenschaften werden wir am Ende des Kapitels durchführen.


Fenster Palette




Fenster Component Tree

*Eingabe-Komponente einfügen.*

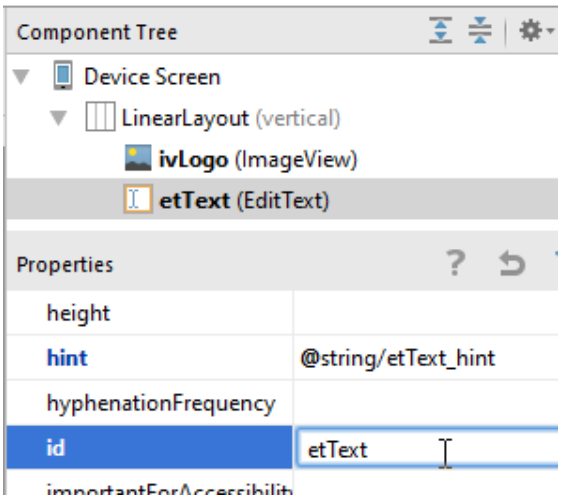
Hier ist eine Komponente vom Typ EditText nötig. Eine → EditText-Komponente (Plain Text, Texteingabefeld):

Wählen Sie dazu die EditText-Komponente im linken Frame-Fenster → Palette aus.

Ziehen Sie dazu diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree.

*Eigenschaften für die Eingabe-Komponente festlegen.*

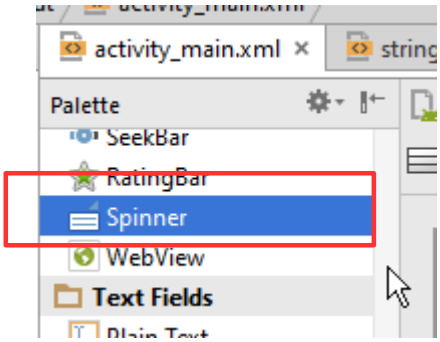
Klicken Sie die Komponente im Fenster → Component Tree und nutzen Sie dann die vertikale Bildlaufleiste im Fenster → Properties, um die Eigenschaft für die layout:width, → hint, → id, und → inputType, wie nebenstehend angezeigt, ändern zu können.



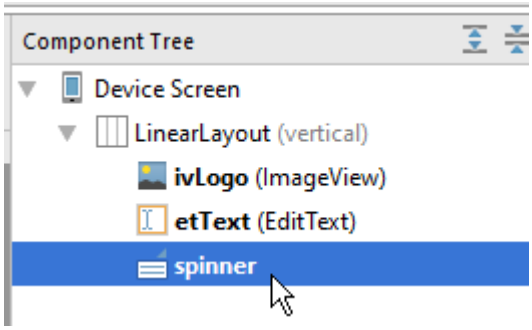

Fenster Component Tree und Properties

EditText: für die Eingabe des Textes

| | |
|---|---|
| layout:width: | fill_parent |
| hint: | @string/etText_hint |
| id: | etText |
| inputType: | [text] |


Fenster Palette




Fenster Component Tree

*Drop-Down-Menü einfügen.*

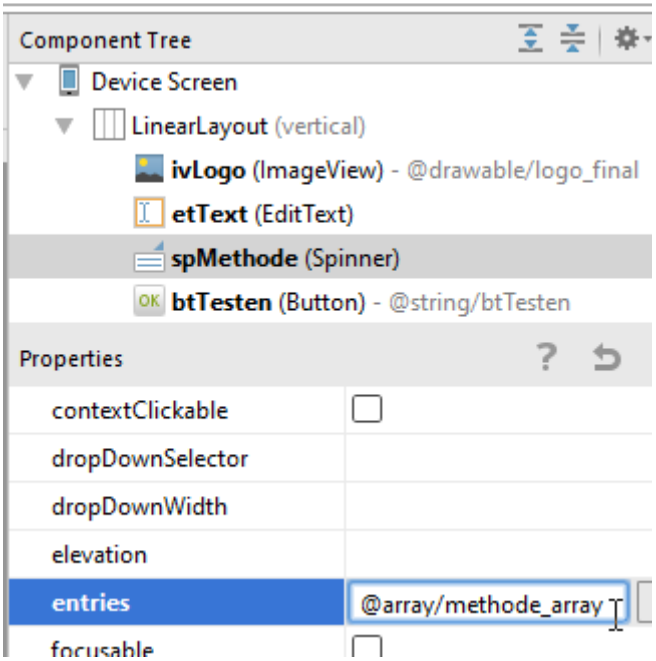
Hier ist eine Komponente vom Typ Spinner nötig. Eine → Spinner-Komponente (Drop-Down-Menü).

Wählen Sie dazu die Spinner-Komponente im linken Frame-Fenster → Palette aus.

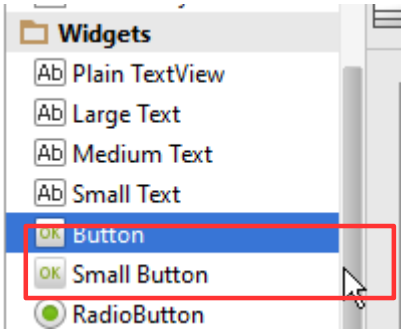
Ziehen Sie dazu diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree, wie nebenstehend angezeigt.

Spinner: für die Wahl der Methode

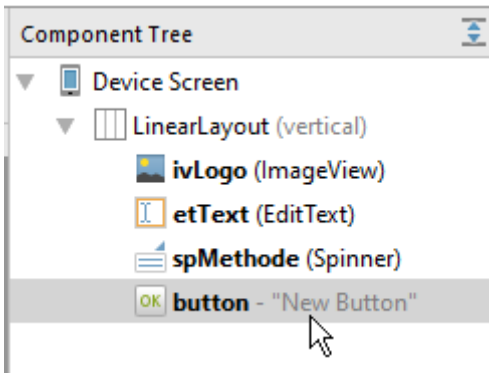
| | |
|---|---|
| layout:width: | fill_parent |
| id: | spMethode |
| entries: | @array/methode_array |




property entries: String array




Fenster Palette




Fenster Component Tree

*Schaltfläche einfügen.*

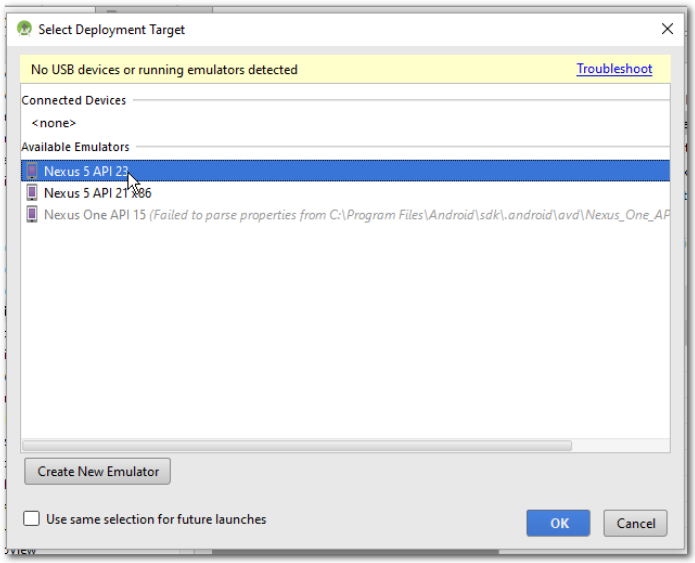
Hier ist eine Komponente vom Typ Button nötig. Eine → Button-Komponente (Button, Schaltfläche).

Wählen Sie dazu die Button-Komponente im linken Frame-Fenster → Palette aus.

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

Button: für die Wahl der Methode

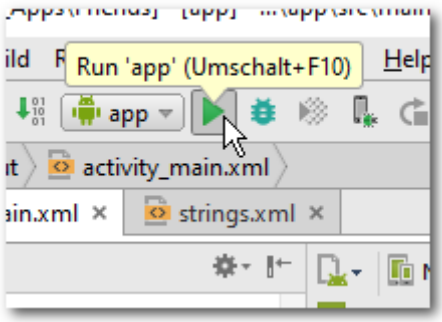
| | |
|---|---|
| layout:width: | `fill_parent` |
| id: | `btTesten` |
| `text:` | `@string/btTesten` |

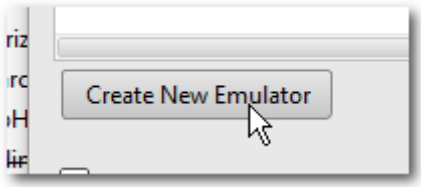

Alternativ → Create New Emulator:
Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit API 15 (SanwichIceCream) zu erzeugen:

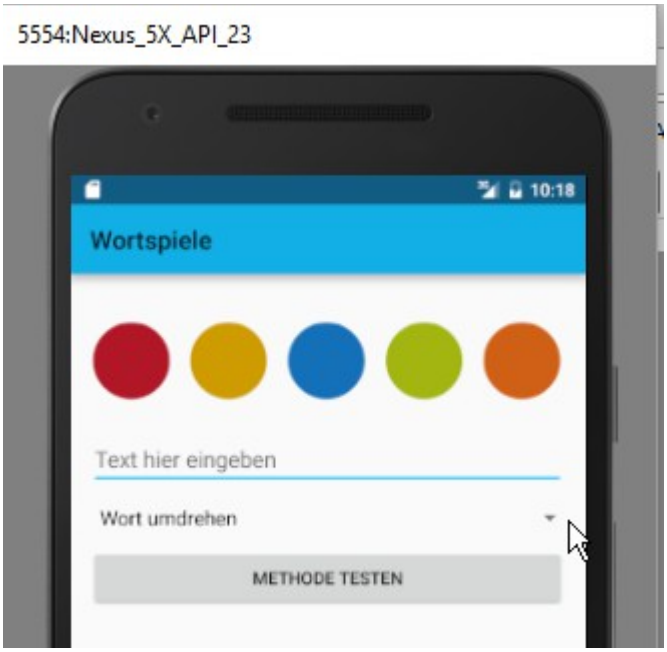
*Testen der View.*

Wir starten nun den Emulator.




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






*Der Emulator öffnet sich.*

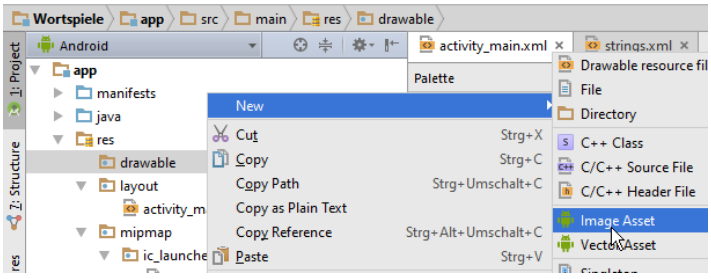
Beim ersten öffnen kann das einen Moment dauern.

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-Prozesses von selbst.

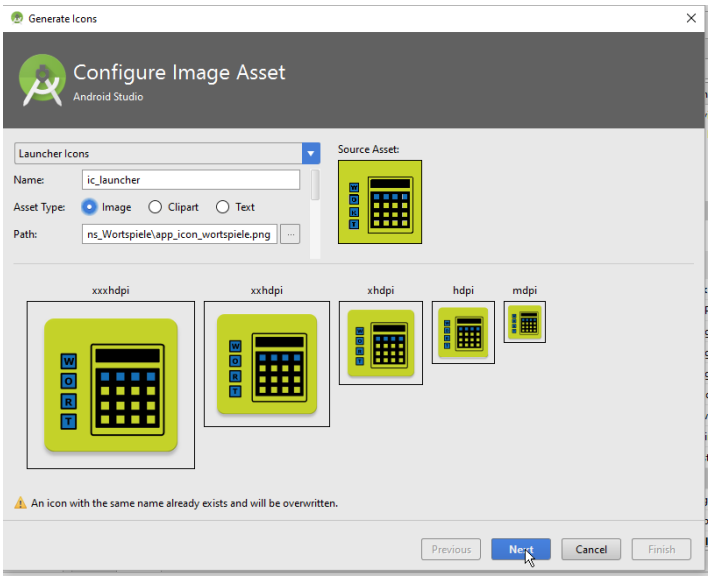
Im Ergebnis sollte die Benutzeroberfläche erscheinen.

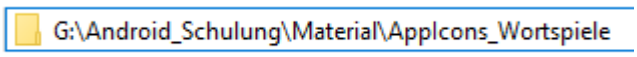

Um das AppIcon und das noch fehlende Logo werden wir uns gleich im nächsten Schritt.




*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.




Bildquelle

Wählen Sie dazu für den Image-File-Pfad die Bild-Datei aus:

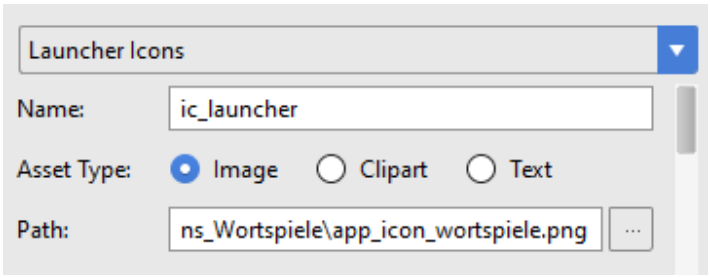
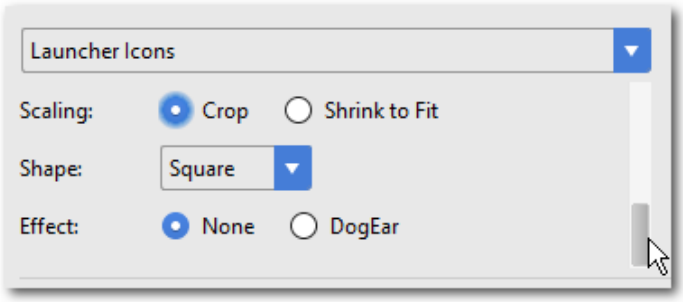
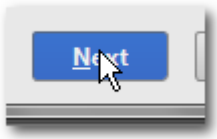


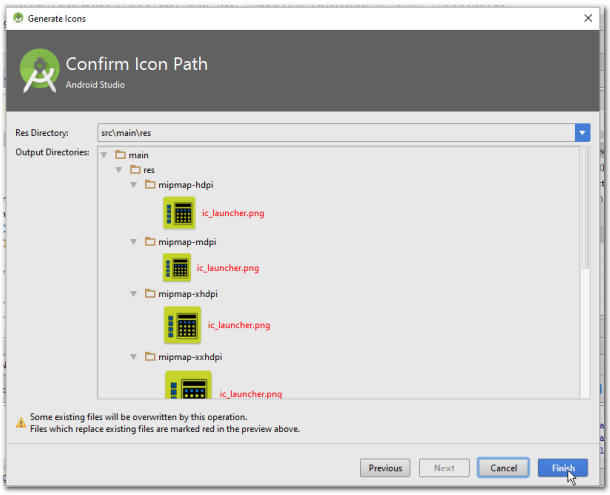

*Image Icon definieren.*

`AppIcons_Wortspiels\app_icon_wortspiele.png`
Aktivieren Sie für die Eigenschaft → Scaling die Option → Crop und für die Eigenschaft → Shape die Option → Square aus:




Klicken Sie auf die Schaltfläche → Next.




*Icon Konfiguration abschließen.*

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

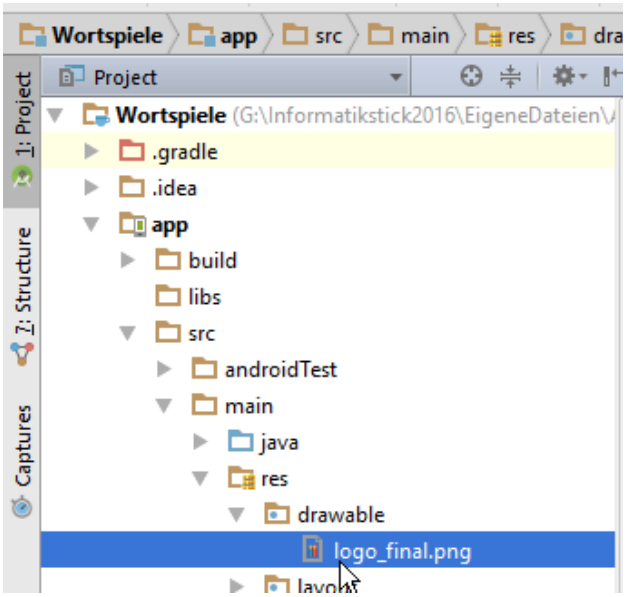
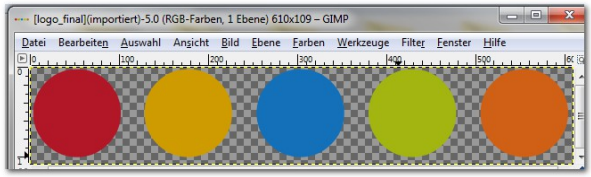
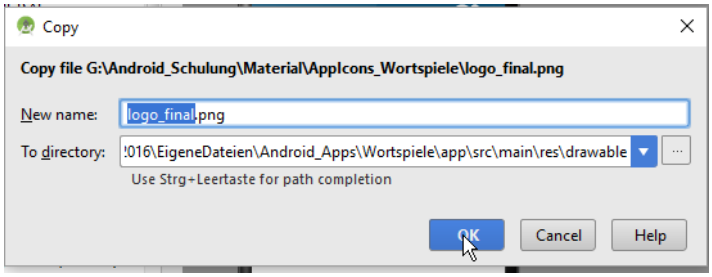


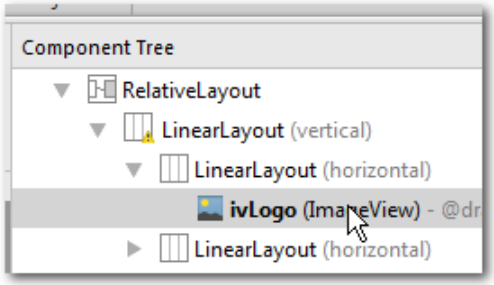

*Bild (Logo) einfügen.*




`AppIcons_Wortspiele\logo_final.png`

Bilddatei „logo_final.png" in das Verzeichnis → res → drawable kopieren:

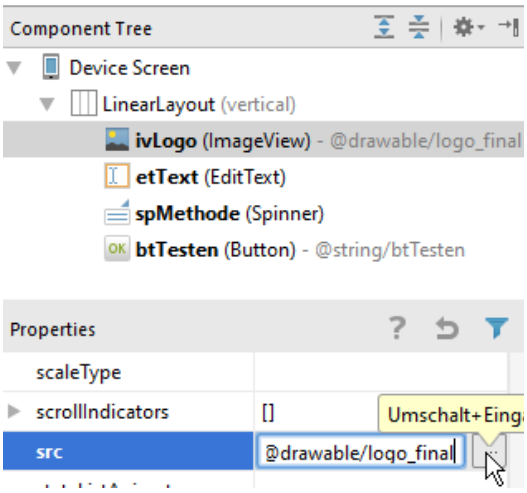



Fenster Component Tree

*Bildquelle definieren.*

Dazu im Fenster Component Tree auf die ImageView-Komponente klicken.

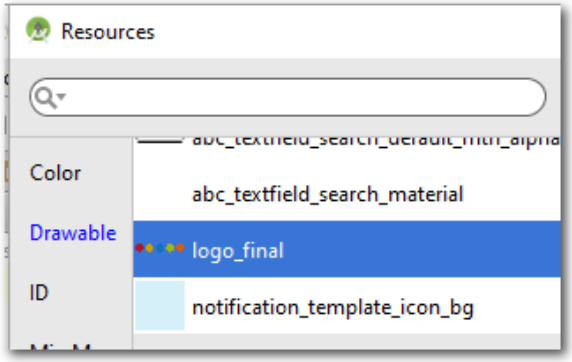



Fenster Component Tree und Properties

*Im Fenster Properties die Bildquelle eingeben.*

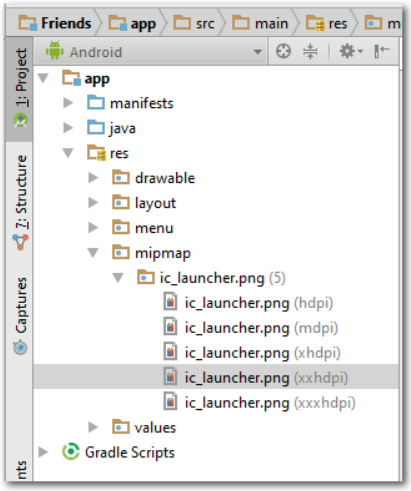
Dazu für die Eigenschaft → src die Quelle:
`@drawbable/logo_final`

definieren.




Logo resource

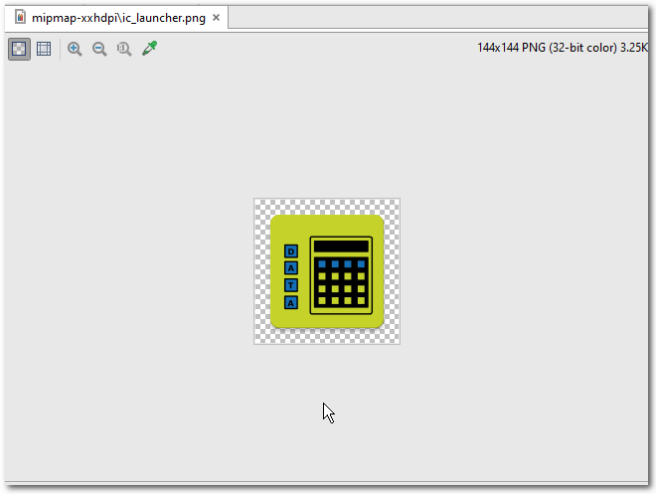
Android Project View



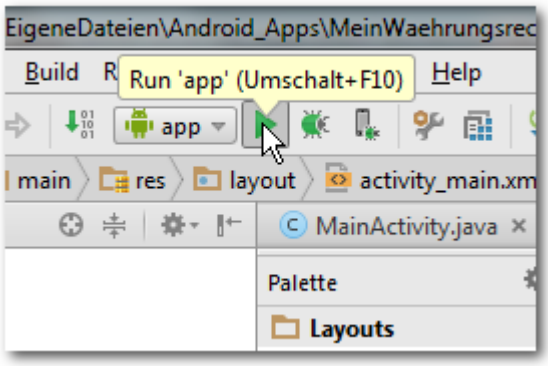

*Kontrolle.*

Dazu im app-Verzeichnis auf das Unterverzeichnis → mipmap klicken.

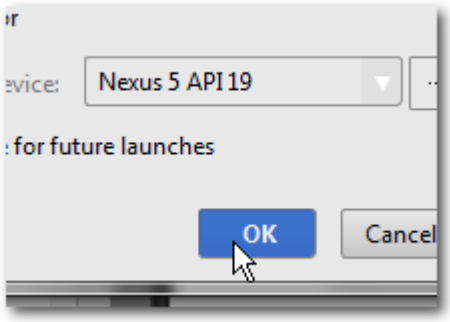
Im Verzeichnis → ic_launcher.png sind die erzeugten Icons in den unterschiedlichen Größen aufgeführt. Mit einem Doppelklick auf einer der Grafiken können Sie diese öffnen.




midmap icons




Schaltfläche: Run 'app'




AVD-Manager

*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".

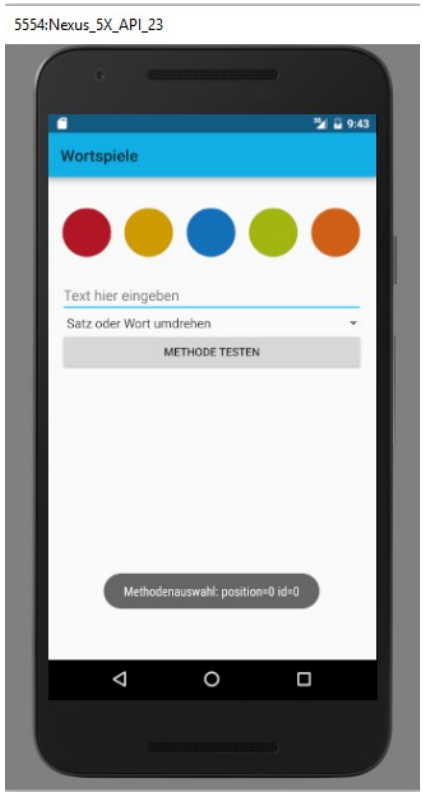
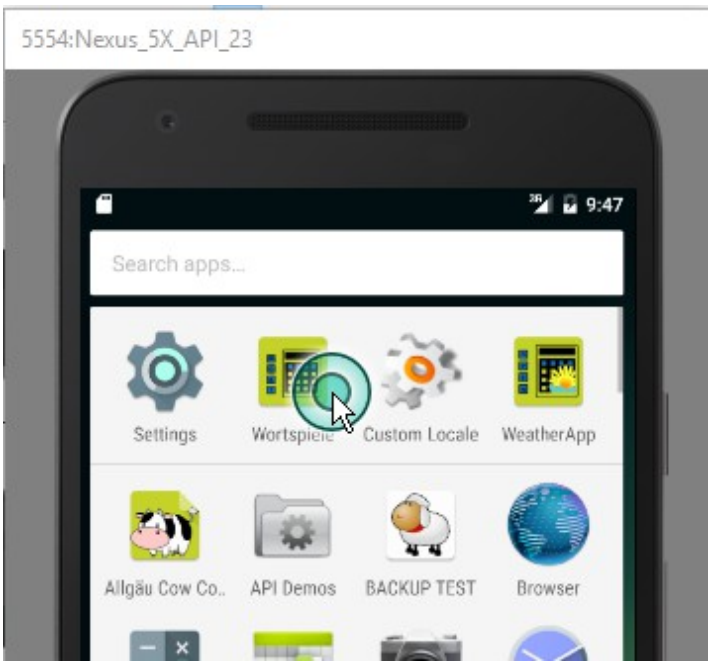



*Bild (Logo) anzeigen.*

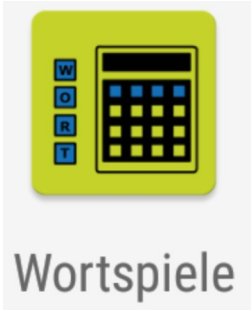
*Mit dem Öffnen der AVD sollte sich auf die Anwendung öffnen, wie nebenstehend angezeigt.*

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




*Icon anzeigen.*

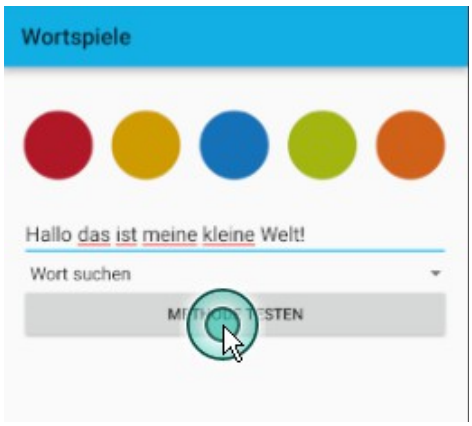
Auf dem Display ist das App Icon nun aufgeführt.

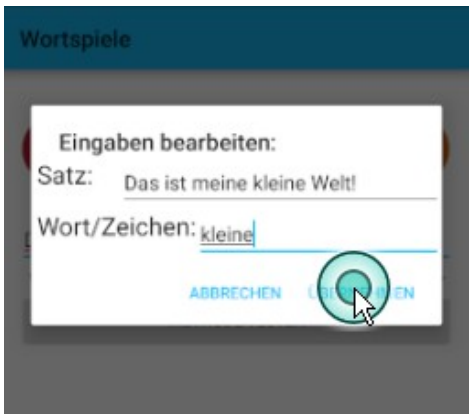



**Gratulation das Layout ist erstellt!**

### 2.3.2 Dialogfenster

Die Operationen → suchen und → entfernen machen zusätzliche Eingaben erforderlich. Wir nutzen ein zusätzliches Layout, um diese Eingaben möglich zu machen.

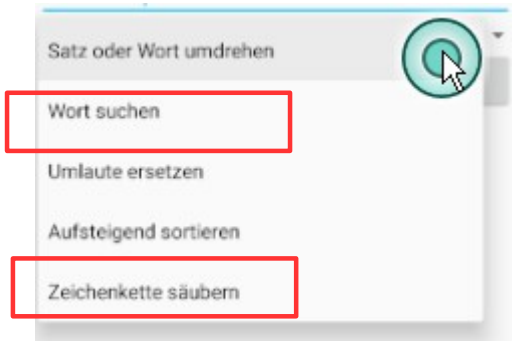

Ergebnis:

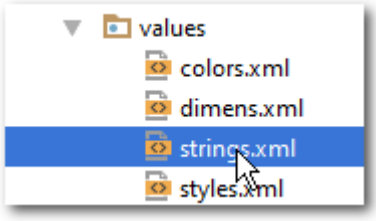


*Zweites Layout.*

Wir werden nun das Dialogfenster für unsere Wortspiele-App ergänzen.

Benutzeroberflächen werden in Android-Apps in der Seitenbeschreibungssprache XML beschrieben.





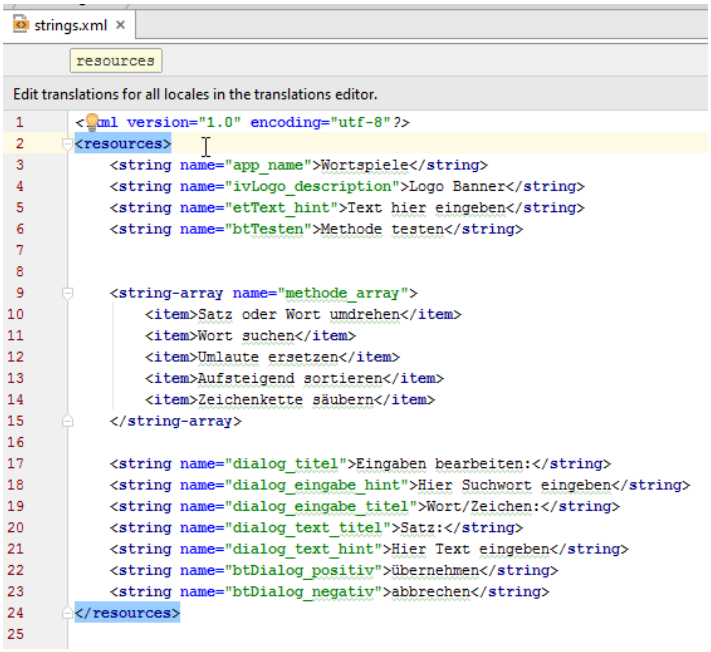

Hinweis:
Aus gutem Grund werden alle Bezeichner der Benutzeroberfläche in eine String-Ressource → (res/values/strings.xml) ausgelagert. Für den Fall, dass Apps in anderen Sprachen verfügbar gemacht werden sollen. Findet der Übersetzer alle benötigten Begriffe in genau einer Datei.

*XML-Deklarationen.*

Dazu definieren wir in einem ersten Schritt alle verwendeten Bezeichnungen für die Komponenten die wir auf unserer Benutzeroberfläche verwenden möchten. Sie sollten in der Datei strings.xml definiert werden.

Öffnen Sie dazu die Datei strings.xml.

Sie finden diese Datei im Unterverzeichnis → app → res → values → strings.xml.




*Bezeichner definieren.*

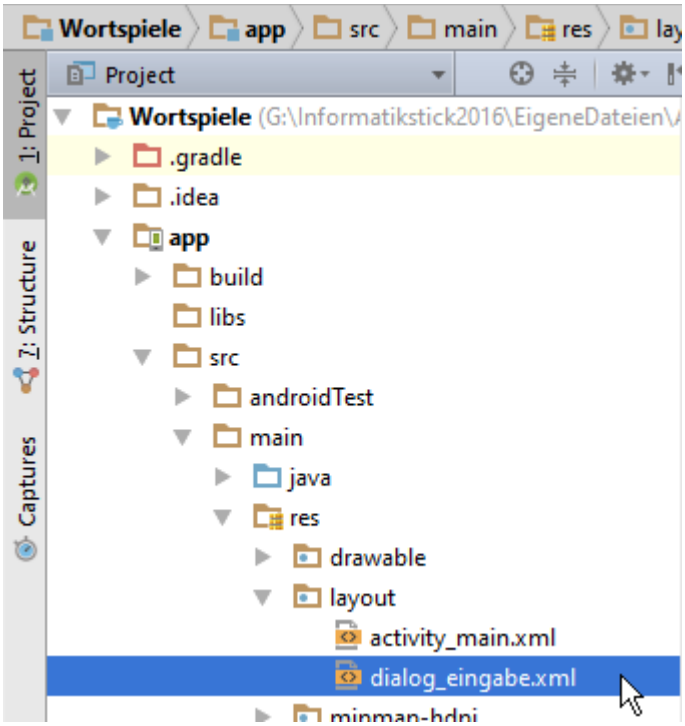
Öffnen Sie die Datei → strings.xml mit einem Doppelklick auf den Dateinamen und erweitern Sie die darin enthaltenen Angaben wie nebenstehend angezeigt.

Hinweis:
Vergleichen Sie die definierten Strings mit der Benutzeroberfläche und identifizieren Sie die Bezeichner.

Eingabehilfe:

```
<resources>
    <string name="app_name">Wortspiele</string>
    <string name="ivLogo_description">
                    Logo Banner</string>
    <string name="etText_hint">
                    Text hier eingeben</string>
    <string name="btTesten">Methode testen</string>

    <string-array name="methode_array">
        <item>Satz oder Wort umdrehen</item>
        <item>Wort suchen</item>
        <item>Umlaute ersetzen</item>
        <item>Aufsteigend sortieren</item>
        <item>Zeichenkette säubern</item>
    </string-array>

    <string name="dialog_titel">
            Eingaben bearbeiten:</string>
    <string name="dialog_eingabe_hint">
            Hier Suchwort eingeben</string>
    <string name="dialog_eingabe_titel">
            Wort/Zeichen:</string>
    <string name="dialog_text_titel">Satz:</string>
    <string name="dialog_text_hint">
            Hier Text eingeben</string>
    <string name="btDialog_positiv">
```

```
                    übernehmen</string>
    <string name="btDialog_negativ">
                    abbrechen</string>
</resources>
```

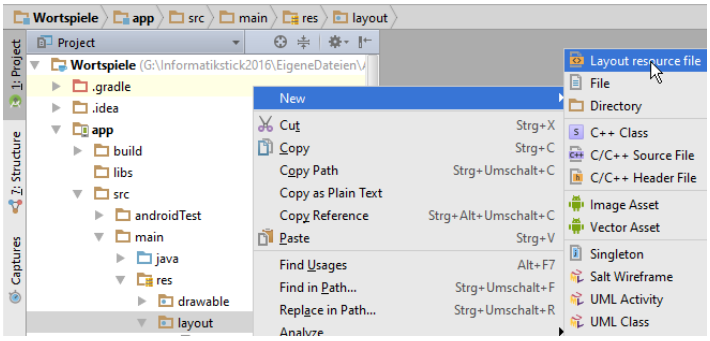


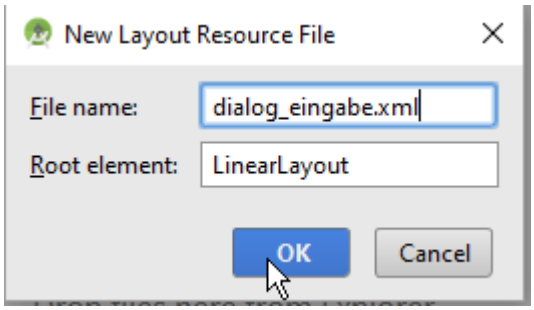

*Layout erstellen.*

Wir werden nun das Layout unserer eigentlichen Benutzeroberfläche erstellen.

Öffnen Sie dazu das Verzeichnis
app → res → layout

Klicken Sie dazu auf → layout und wählen Sie im Kontext-Menü → New → Layout resource file

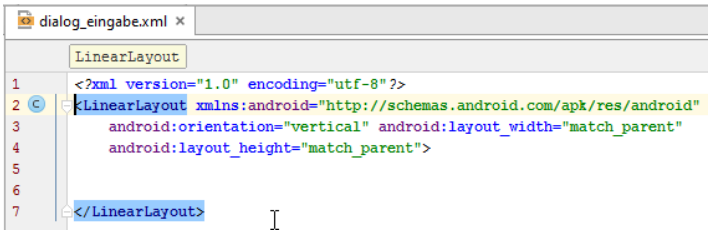



*Layout benennen.*

Geben Sie den Dateinamen und legen Sie als Root Element das Lineare Layout fest.

Klicken Sie auf die Schaltfläche → OK.




Das Lineare Layout (vertikal):
Die in einem vertikalen Linearen Layout platzierten Komponenten werden untereinander angeordnet.

*Layouts verwenden.*

Wir schachteln Layouts für das Dialogfenster.

Wie Sie im folgenden sehen, verwenden wir vertikale und horizontale Lineare Layouts.

Das Lineare Layout (horizontal):
Die in einem horizontalen Linearen Layout platzierten Komponenten werden nebeneinander angeordnet.

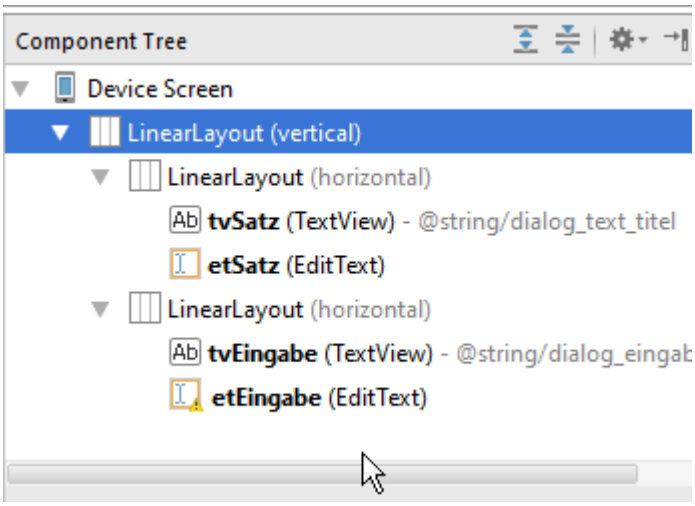
Designer

*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.

Hinweis:
Die Anwendung besitzt ähnlich, wie in Eclipse der 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.

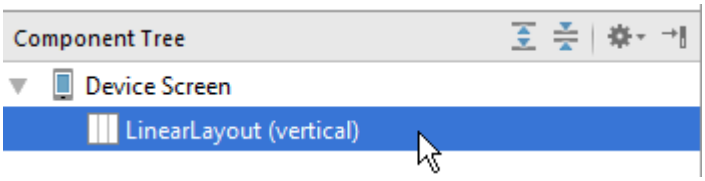


Gewünschtes Ergebnis

*Vorgehensweise: Component Tree.*

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

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



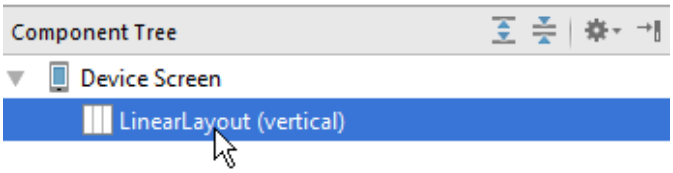
Aktueller Komponenten-Baum

*Der Komponenten-Baum.*

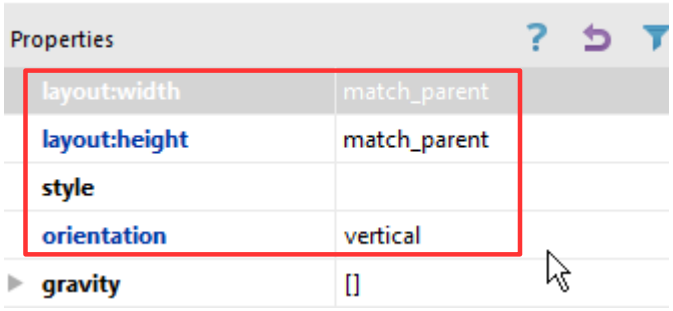
Im oberen, rechten Frame-Fenster wird der Komponenten-Baum (Component Tree) angezeigt.

Als Komponenten werden alle Elemente einer Benutzeroberfläche bezeichnet.

Die Grundlage jeder Benutzeroberfläche sind die Layouts.




Fenster Component Tree




Fenster Properties

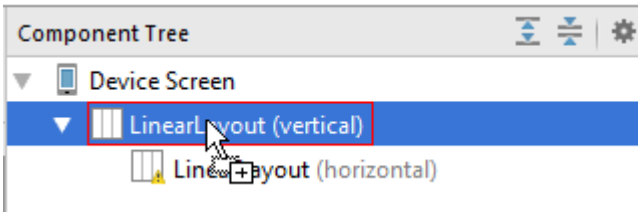
*Eigenschaften des Layouts ändern.*

Klicken Sie dazu im Fenster → Component Tree auf das → LineareLayout (Vertical).

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

Properties:

| | |
|---|---|
| layout:width: | `match_parent` |
| layout:height: | `match_parent` |
| orientation: | `vertical` |

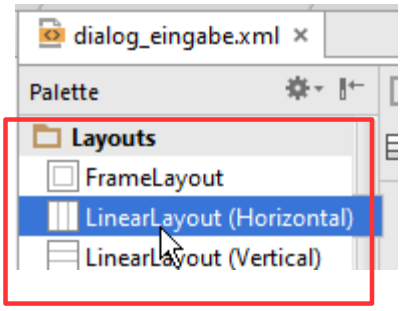



Fenster Component Tree

*LinearesLayout (Horizontal) verwenden.*

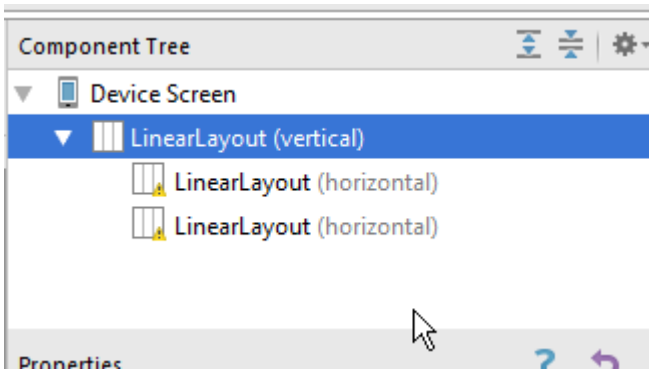
Klicken Sie dazu im linken Frame-Fenster → Palette neben der Design-Bühne auf die Option → LinearLayout (Horizontal)".

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.

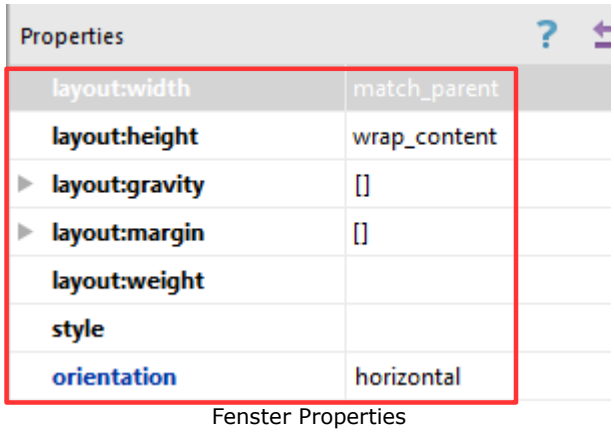



Fenster Palette

Wiederholen Sie diese Maßnahme, um ein zweites Lineares horizontales Layout zu platzieren:




Fenster Component Tree




Fenster Properties

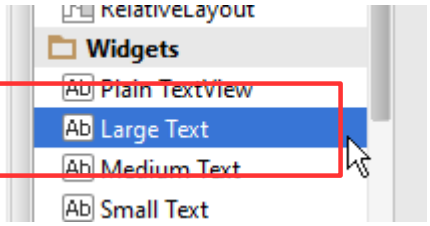
*Eigenschaften des Layouts ändern.*

Klicken Sie dazu im Fenster → Component Tree auf das → LineareLayout (Horizontal).

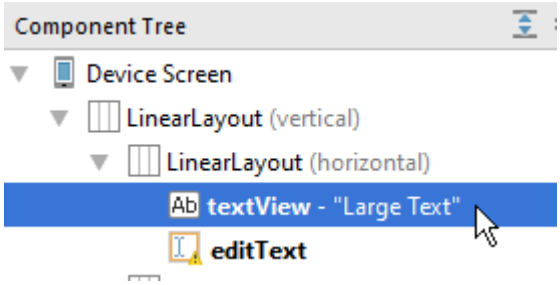
Ändern Sie dann die nebenstehend angezeigten Eigenschaften der Komponente im darunterliegenden Fenster → Properties ab.

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

Übernehmen Sie diese Einstellungen auch für das zweite Lineare horizontale Layout.




Fenster Palette




Fenster Component Tree

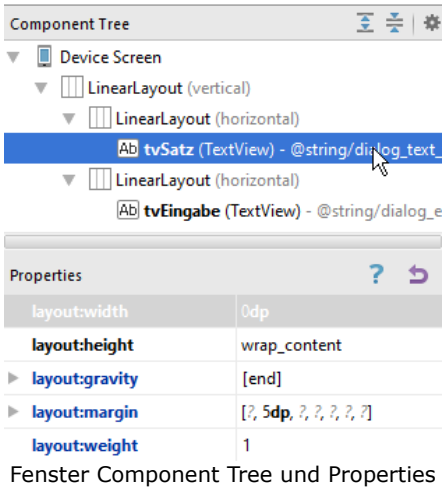
*Bezeichner-Komponenten einfügen.*

Im ersten horizontalen Linearen Layout fügen wir eine Komponente vom Typ TextView ein.

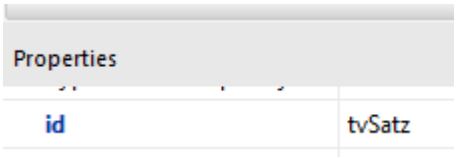
Eine → TextView-Komponente (Large Text, Bezeichner):

Wählen Sie dazu die TextView-Komponente im linken Frame-Fenster → Palette aus.

Ziehen Sie dazu diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree.



Fenster Component Tree und Properties



Fenster Properties

Fenster Properties

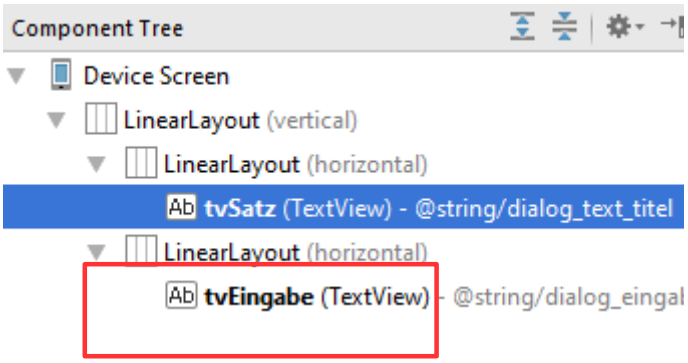
View im Designer

*Eigenschaften für die Bezeichner-Komponenten ändern.*

Klicken Sie die Komponente im Fenster → Component Tree an und nutzen Sie dann die vertikale Bildlaufleiste im Fenster → Properties, um die folgenden Eigenschaftswerte zu ändern.

TextView: für die Bezeichnung des Textes

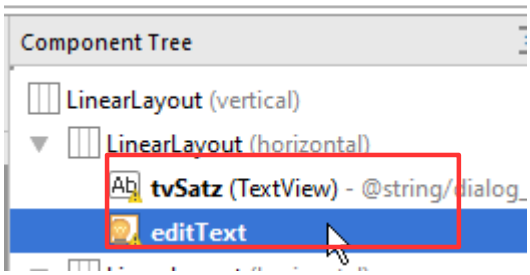
| | |
|---|---|
| layout:width: | 0dp |
| layout:width: | wrap_parent |
| layout:gravity: | [end] |
| layout:margin: | left (5dp) |
| layout:weight: | 1 |
| id: | tvSatz |
| text: | @string/dialog_text_titel |



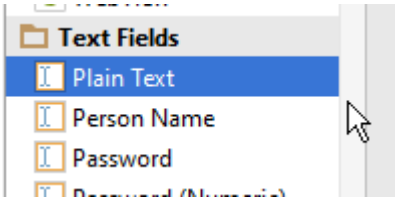

Wiederholen Sie die letzten zwei Schritte und legen Sie die Eigenschaften für die TextView-Komponente → tvEingabe im zweiten horizontalen Linearen Layout , wie folgt fest.

TextView: für die Bezeichnung des Wort/Zeichen

| | |
|---|---|
| layout:width: | 0dp |
| layout:width: | wrap_parent |
| layout:gravity: | [end] |
| layout:margin: | left (5dp) |
| layout:weight: | 1 |
| id: | tvEingabe |
| text: | @string/dialog_eingabe_titel |




Fenster Component Tree




Fenster Palette

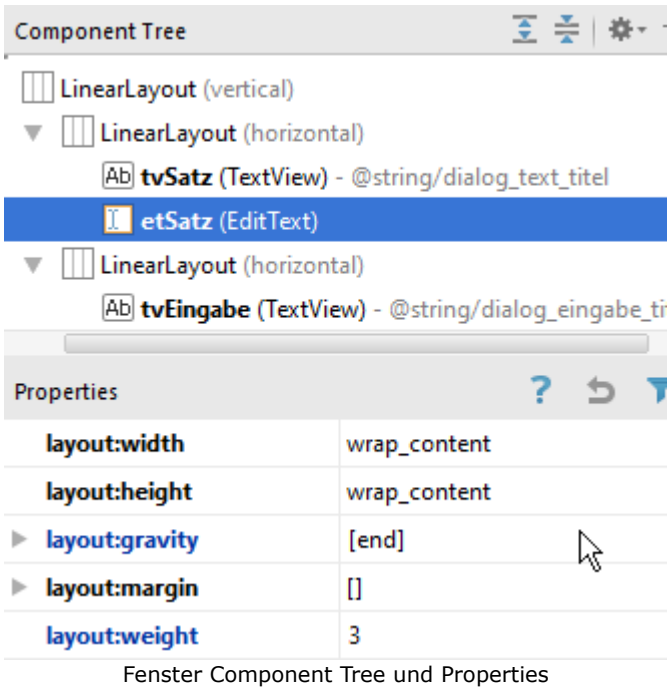
*Eingabe-Komponenten einfügen.*

Im ersten horizontalen Linearen Layout fügen wir eine Komponente vom Typ TextView ein.

Eine → EditText-Komponente (Plain Text, Texteingabefeld):

Wählen Sie dazu die TextView-Komponente im linken Frame-Fenster → Palette aus.

Ziehen Sie dazu diese Komponente mit gedrückter linker Maustaste in das rechte, obere Frame-Fenster → Component Tree.



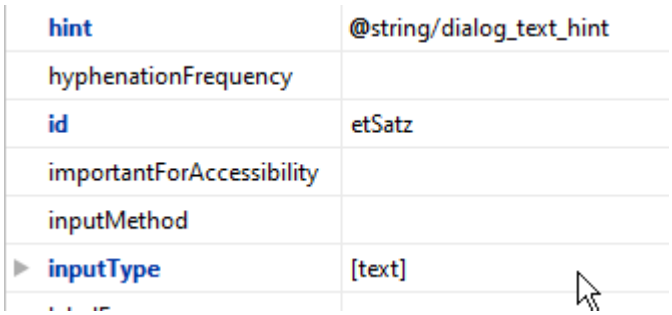

Fenster Component Tree und Properties

*Eigenschaften für die Eingabe-Komponenten ändern.*

Klicken Sie die Komponente im Fenster → Component Tree an und nutzen Sie dann die vertikale Bildlaufleiste im Fenster → Properties, um die folgenden Eigenschaftswerte zu ändern.

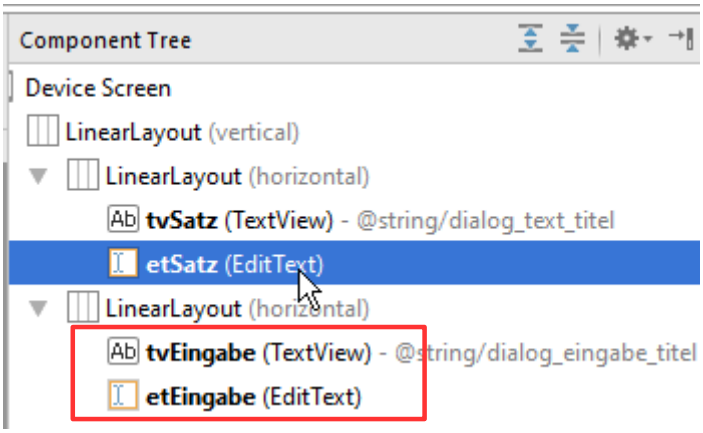
EditText: für die Eingabe des Textes

| | |
|---|---|
| layout:width: | 0dp |
| layout:width: | wrap_parent |
| layout:gravity: | [end] |
| layout:weight: | 3 |
| hint: | @string/dialog_text_hint |
| id: | etSatz |
| inputType: | [text] |




Fenster Properties


View im Designer




Wiederholen Sie die letzten zwei Schritte und legen Sie die Eigenschaften für die EditText-Komponente → etEingabe im zweiten horizontalen Linearen Layout , wie folgt fest.

EditText: für die Eingabe des Wort/Zeichen

| | |
|---|---|
| layout:width: | `0dp` |
| layout:width: | `wrap_parent` |
| layout:gravity: | `[end]` |
| layout:weight: | `3` |
| hint: | `@string/dialog_eingabe_hint` |
| id: | `etEingabe` |
| inputType: | `[text]` |

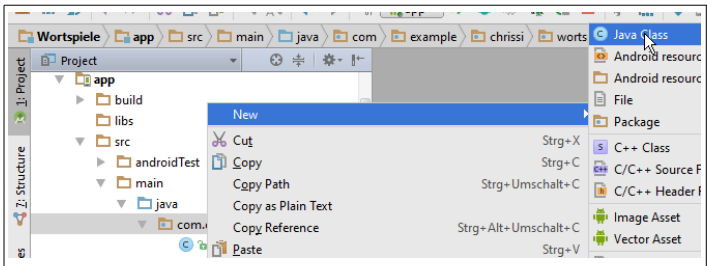
Fertig!

Wir wenden uns im nächsten Schritt der Progammlogik zu und implementieren dazu die Fachklasse Wortspiel.

## 2.4 Modell: Implementierung der Fachklasse Wortspiel

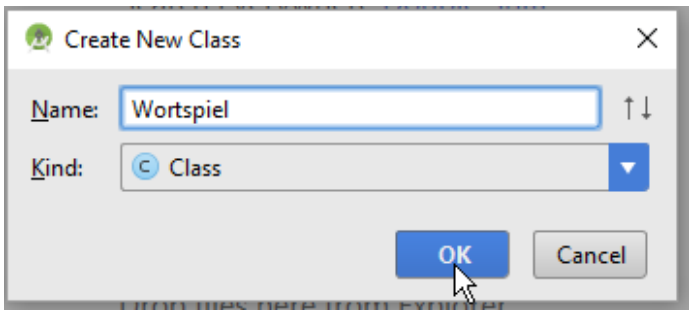
### 2.4.1 Grundgerüst

Wir möchten in unserer App die Datenhaltung und -verwaltung vieler Wortspiele verwalten. Dazu schaffen wir im nächsten Schritt eine Mustervorlage (→ Fachklasse) die wir für alle unsere Wortspiele-Objekte verwenden können.



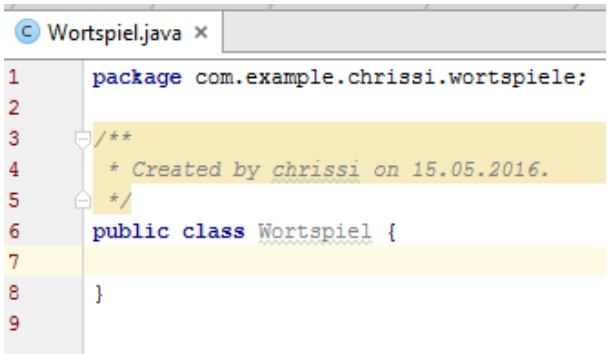

*Neue Fachklasse erstellen.*

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




*Klassenname festlegen.*

Geben Sie als Klassennamen → Wortspiel ein und klicken Sie auf die Schaltfläche → OK.



Wortspiel.java

```
1  package com.example.chrissi.wortspiele;
2
3  /**
4   * Created by chrissi on 15.05.2016.
5   */
6  public class Wortspiel {
7  //Attribute: Deklaration der
8  // Eigenschaften einer Klasse
9
10
11 //Konstruktor: mit Parameter
12
13
14 /*Getter: Ermittelt Eigenschaftswert eines
15 eines Objektes Setter: Übermittelt
16 Eigenschaftswert an das Attribut eines Objektes*/
17
18
19 /*Sonstige Methoden: können mehr als
20 nur er- und übermitteln. Hier: Die von Object
21 vererbte toString-Methode wird überschrieben*/
22
23 }
```

Eingabehilfe:

```
//Attribute: Deklaration der Eigenschaften einer
Klasse


//Konstruktor: mit Parameter


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


/*Sonstige Methoden: können mehr als nur er- und
übermitteln. Hier: Die von Object vererbte to-
String-Methode wird überschrieben*/
```

1. Deklaration der Attribute
2. Deklaration des Konstruktors
3. Get-Methoden (Getter) deklarieren und implementieren.
4. Set-Methode (Setter) deklarieren und implementieren.
5. Sonstige Methoden deklarieren und implementieren

*Grundgerüst einer Klasse festlegen.*

Übernehmen Sie die nebenstehend angezeigten Kommentare.

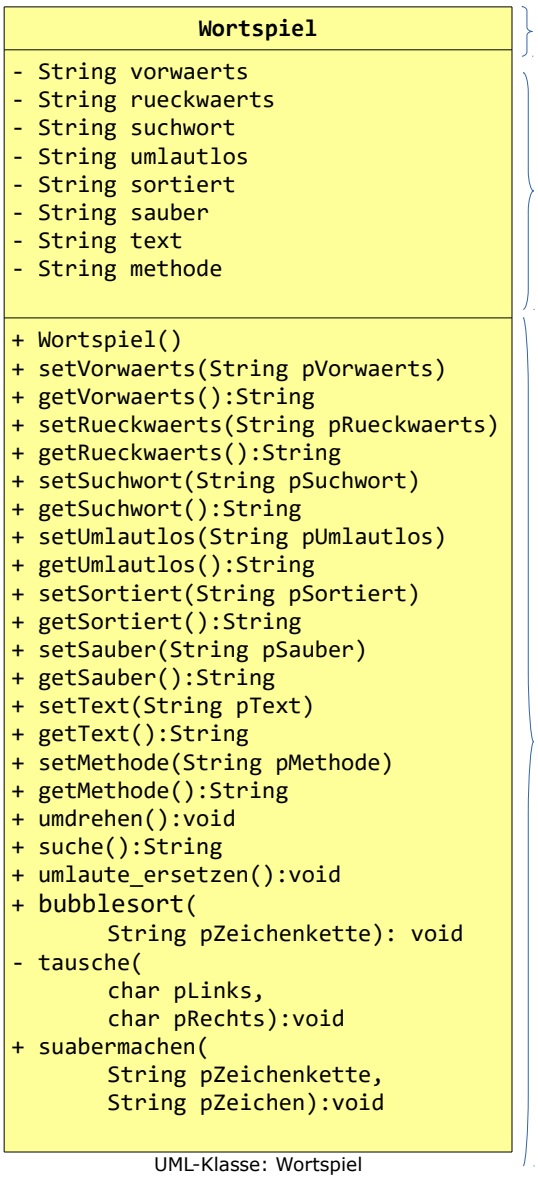
```
public class Wortspiel {
        //hier fehlt Quellcode
}
```

Deklarieren:
In der objektorientierten Programmierung ist mit der Deklaration die

1. Festlegung einer Dimension, eines Bezeichners,
2. eines Datentyp und
3. weiterer Aspekte einer Klasse, eines Konstruktors, einer Eigenschaft (Attribut) oder einer Verhaltensweise (Methode und Signatur),

gemeint.

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) einer bestimmten Programmiersprache. Zumeist handelt es sich um das Anfüllen der Methoden mit dem benötigten Quellcode, also Inhalt einer Methode. Dabei dient der Quellcode dazu, die gewünschten Verhaltensweisen eines Systems (Programms) zu realisieren.



UML-Klasse: Wortspiel

*Fachklasse implementieren.*

Wir erzeugen eine Mustervorlage für all unsere Freunde.

Dazu implementieren die Fachklasse → Wortspiel, indem wir sie mit dem benötigten Quellcode ausstatten.

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

```
7   //Attribute: Deklaration der
8   // Eigenschaften einer Klasse
9       private String vorwaerts;
10      private String rueckwaerts;
11      private String suchwort;
12      private String umlautlos;
13      private String sortiert;
14      private String sauber;
15      private String text;
16      private String methode;
```

*Deklaration der Attribute.*

```
private String vorwaerts;
```

Der Zugriffsmodifikator → private stellt sicher, dass nur die Objekte der Klasse selbst auf die Eigenschaftswerte direkt zugreifen können.

Mit der Bestimmung des geeigneten Datentyps für ein Attribut wird gleichzeitig der maximal benötigte Speicherplatz vorab reserviert.

→ vorwärts ist der Attributname. Attribute wer-

Eingabehilfe:

```
private String vorwaerts;
private String rueckwaerts;
private String suchwort;
private String umlautlos;
private String sortiert;
private String sauber;
private String text;
private String methode;
```

Der Datentyp String:
Der *komplexe Datentyp* → String bestimmt den Wertebereich einer Zeichenkette (Implementierung: Array aus Characters, siehe auch Oracles API → String).

den in Java kleingeschrieben und enthalten keine Umlaute und/oder Sonderzeichen.

Hinweis:
Leerzeichen sind auch Sonderzeichen!

Deklarieren Sie auch die übrigen Attribute.

```
17      //Konstruktor: Default Konstruktor
18      // (Parameterlos, ohne Inhalt)
19          public Wortspiel(){
20
21          }
```

Beispiel Konstruktoraufruf:

```
Wortspiel einWortspiel =
        new Wortspiel();
```

Der Konstruktor einer Klasse sorgt dafür, dass beliebig viele Objekte der Klasse erzeugt, „konstruiert" werden können.

*Deklaration eines Konstruktors mit Parameter.*

Jeder Benutzer erzeugt damit seine eigenen Freunde.

Der Standard Konstruktor enthält keine Parameter und initialisiert keine Anfangswerte. Neu erzeugte Objekte sind also am Anfang ihrer Entstehung „wertelos".

Wir wollen nicht verhindern, dass es „wertelose" Wortspiele gibt und nutzen im vorliegenden Fall einen parameterlosen Konstruktor. Es wird also in unserem System so sein, dass wir in jedem Fall, ein neues Objekt der Klasse → Wortspiel erzeugen können.

```
26      /*Getter: Ermittelt Eigenschaftswert eines
27      eines Objektes
28
29      Setter: Übermittelt
30      Eigenschaftswert an das Attribut eines Objektes*/
31      // Getter und Setter
32          public String getVorwaerts() {
33              return vorwaerts;
34          }
35
36          public void setVorwaerts(String pVorwaerts) {
37              this.vorwaerts = pVorwaerts;
38          }
39
```

Getter und Setter:

```
public String getVorwaerts() {
    return vorwaerts;
}
public void setVorwaerts(String pVorwaerts) {
```

*Deklaration und Implementierung der Get- und Set-Methoden.*

Berücksichtigen Sie, dass wir auf die Eigenschaftswerte und Verhaltensweisen der Wortspiel-Objekte von außerhalb der Klasse (z.B. von der Benutzeroberfläche aus) zugreifen müssen.

Implementieren Sie dafür die Get- und Set-Methoden für alle Attribute, wie nebenstehend für das Attribut → vorwärts angezeigt.

Beispiel: Ermitteln

```
einWortspiel.getVorwärts();
```

Die Zeichenkette → String vorwaerts ist das Ergebnis aus der Verhaltensweise (Methode) →

```
    this.vorwaerts = pVorwaerts;
}
```

umdrehen():void.

Wir widmen und also im nächsten Schritt der Umsetzung konkreter Algorithmen und kapseln diese in sonstigen Methoden-Päckchen.

Eingabehilfe:

```
public void umdrehen(){
      //hier fehlt Quellcode
}


public String suche(){
      //hier fehlt Quellcode
}

public void umlaute_ersetzen(){
      //hier fehlt Quellcode
}


public void bubblesort(){
      //hier fehlt Quellcode
}


private char[] tausche(char links,
                       char rechts,
                       char [] in){
      //hier fehlt Quellcode
}


public void saubermachen(){
      //hier fehlt Quellcode
}
```

*Überblick sonstige Methoden.*

Überblick:

1. Die Methode das eingegebene Wort (Zeichenkette) *umdrehen*
2. Die Methode soll eine beliebige Zeichenkette *suchen*
3. Die Methode soll das Wort durchlaufen und *Umlaute* ermitteln und *ersetzen*
   - ü --> ue
   - ä --> ae
   - ö --> oe
   - ß --> ss
4. *Sortieren* von Buchstaben: Sortiert eine beliebige Zeichenkette aufsteigend durch Vergleichen und tauschen der Zeichen – Bubblesort
5. Die Methode soll einen *String säubern*: Dazu soll ein beliebiges Zeichen bzw. Zeichenkette und aus einer bestehenden Zeichenkette entfernt werden.

## 2.4.2 Algorithmus: umdrehen

```
94      //Dreht die eingegebene Zeichenkette um
95      public void umdrehen(){
96          //Eingabe übernehmen
97          String mZeichenkette= this.satz;
98
99          //Das am Anfang noch leere Ergebnis initialisieren
100         this.rueckwaerts = new String();
101
102         //Druchläuft die Zeichenkette zeichenweise
103         //von hinten beginnend
104         for (int i = mZeichenkette.length()-1;i>=0 ; i--) {
105
106             //Ermittelt das aktuelle Zeichen
107             char zeichen = mZeichenkette.charAt(i)  ;
108
109             //Aktualisiert die aktuelle Zeichenrette rueckwaets
110             //und hängt das ermittelte Zeichen hinten an.
111             this.rueckwaerts = this.rueckwaerts
112                     .concat(String.valueOf(zeichen));
113         }
114
115         //Testausgabe
116         System.out.println(this.rueckwaerts);
117     }
```

*Algorithmus für umdrehen.*

Der Algorithmus dreht eine beliebige Zeichenkette um.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare ein.

Für die Methode:
umdrehen(): void

```
public void umdrehen(){
      //hier fehlt Quellcode
 }
```

Eingabewert aus dem Objekt der Fachklasse ermitteln:

```
String mZeichenkette= this.satz;
```

Das am Anfang noch leere Ergebnis initialisieren:

```
this.rueckwaerts = new String();
```

Dann wird die Zeichenkette zeichenweise von **hinten** beginnend durchlaufen:

```
for (int i = mZeichenkette.length()-1;i>=0 ; i--) {
      //hier fehlt Quellcode
}
```

Ermittelt in jedem Durchlauf das aktuelle Zeichen an der **Stelle i**:

```
char zeichen = mZeichenkette.charAt(i);
```

Aktualisiert die aktuelle, am Anfang noch leere Zeichenkette rueckwaets und hängt das ermittelte Zeichen, als String, hinten an:

```
this.rueckwaerts = this.rueckwaerts
        .concat(String.valueOf(zeichen));
```

Erzeugt eine Testausgabe auf der Konsole:

```
System.out.println(this.rueckwaerts);
```

Fertig!!

## 2.4.3 Algorithmus: suchen

```
126     /*Die Methode soll eine beliebige
127    Zeichenkette ermitteln*/
128     public String suche(){
129         //Eingabe übernehmen
130         String mSatz = this.text;
131
132
133         //Ein String-Objekt erzeugen
134         String meldung = new String();
135
136         //Prüft ob das suchwort im Satz enthalten
137         if(mSatz.contains(this.suchwort)) {
138
139             //Der Satz wird anhand des Leerzeichens
140             //in eine Wortkette umgewandelt
141             String[] wortkette = mSatz.split(" ");
142
143             //Durchlaufe die Wortkette
144             for(int i = 0; i< wortkette.length; i++ ){
145
146                 //Das aktuelle Wort ermitteln
147                 String mWort = wortkette[i];
148
149                 //Für den Fall, dass das Suchwort
150                 //mit dem aktuellen Wort gleich ist
151                 if (this.suchwort.equals(mWort)) {
152
153                     //Ermittet die Stelle des aktuellen Worts
154                     int mStelle = i;
155
156                     //Erzeugt die Meldung
157                     meldung = "Das Wort "+mWort
158                             + "wurde an der Stelle "
159                             + i + " gefunden!";
160                 }
161             }
162
163         }else{
164             //Erzeugt die Meldung
165             meldung = "Das Wort "
166                     +this.suchwort
167                     + " wurde nicht gefunden!";
168         }
169         //Gibt die Meldung zurück
170         return meldung;
171     }
```

Eingabehilfe:

```
public String suche(){
    //Eingabe übernehmen
    String mSatz = this.text;
    //Ein String-Objekt erzeugen
    String meldung = new String();
    //Prüft ob das suchwort im Satz enthalten
    if(mSatz.contains(this.suchwort)) {
        //Der Satz wird anhand des Leerzeichens
        //in eine Wortkette umgewandelt
        String[] wortkette = mSatz.split(" ");
        //Durchlaufe die Wortkette
        for(int i = 0; i< wortkette.length; i++ ){
```

*Algorithmus für suchen.*

Der Algorithmus identifiziert ein beliebiges Wort in der Eingabe (Satz) und meldet als Ergebnis das Wort und dessen Position.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare hinzu.

Für die Methode:

suche(): String

```
public String suche(){
    //hier fehlt Quellcode
}
```

Eingabe übernehmen:

```
String mSatz = this.text;
```

Ein String-Objekt erzeugen:

```
String meldung = new String();
```

Prüft ob das suchwort im Satz enthalten:

```
if(mSatz.contains(this.suchwort)) {
        //Ja-Fall
}else{
        //Nein-Fall
}
```

**Im Ersten Schritt** implementieren wir für den **Ja-Fall**.

Für den **Ja-Fall**.

Dazu wird der Satz anhand des Leerzeichens in eine Wortkette umgewandelt:

```
String[] wortkette = mSatz.split(" ");
```

Die Durchlaufe die Wortkette:

```
for(int i = 0; i< wortkette.length; i++ ){
        //Wiederholung
}
```

Die Wiederholungen:

Ermittle bei jedem Durchlauf das aktuelle Wort.

```
String mWort = wortkette[i];
```

Für den Fall, dass dann das Suchwort mit dem aktuellen Wort gleich ist:

```
            //Das aktuelle Wort ermitteln
            String mWort = wortkette[i];
            //Für den Fall, dass das Suchwort
            //mit dem aktuellen Wort gleich ist
            if (this.suchwort.equals(mWort)) {
                //Ermittet die Stelle des
                 //aktuellen Worts
                int mStelle = i;
                //Erzeugt die Meldung
                meldung = "Das Wort "+mWort
                        + "wurde an der Stelle "
                        + i + " gefunden!";
            }
        }
    }else{
        //Erzeugt die Meldung
        meldung = "Das Wort "
                +this.suchwort
                + " wurde nicht gefunden!";
    }
    //Gibt die Meldung zurück
    return meldung;
}
```

Zusatzaufgabe:
Ein Beispiel für Polymorphie (Mehrgestaltigkeit). Implementieren Sie die parameterlose Methode ohne Rückgabewert:

```
suche(Sting pSuchwort): void
```

Die *parameterlose* Methode mit Rückgabewert soll das gleiche Verhalten aufweisen, wie die zuvor implementierte parameterlose Methode ohne Rückgabewert:

```
suche(): String
```

Wäre es möglich die Suche als Methode *mit Parameter* und *mit Rückgabewert* zu implementieren?

Diese Art:

```
suche(Sting pSuchwort): String
```

Fertig!!

```
if (this.suchwort.equals(mWort)) {
        //Für die Stelle

        //Für die Meldung
}
```

Für die Stelle:

```
int mStelle = i;
```

Für die Meldung:

```
meldung = "Das Wort "+mWort
        + "wurde an der Stelle"
        + i + "gefunden!";
```

Fortsetzung:
**Im Zweiten Schritt** implementieren wir für den **Nein-Fall**.

Für den **Nein-Fall**.

Erzeugt die Meldung:

```
meldung = "Das Wort "
        +this.suchwort
        + " wurde nicht gefunden!";
```

Erst wenn die ganze Wortkette durchlaufen ist, soll abschließend wird die Meldung zurückgegeben:

```
return meldung;
```

## 2.4.4 Algorithmus: sortieren

```
224     /*Sortieren von Buchstaben: Vergleichen und
225     austauschen - Bubblesort*/
226     public void bubblesort(String pUnsortiert){
227         //Deklaration eines noch leeren
228         // Arrays von chars
229         char[] getauscht_char = new char[0];
230
231         //Umwandeln der Zeichenkette in ein
232         // Array von Zeichen
233         char[] mUnsortiert =
234                 pUnsortiert.toCharArray();
235
236
237     //Durchlaufe die Liste
238         for (int i = mUnsortiert.length; i>1;i-- ) {
239
240             //Ermittle zwei Elemente benachbarte
241             // Elemente der Liste
242             for (int j=0; j<i-1; j++){
243                 //Prüfe ob das linke Element größer
244                 // ist als das rechte Element
245                 if (mUnsortiert[j] > mUnsortiert[j+1]){
246
247                     //Beispiel für die Kapselung
248                     int links = j;
249                     int rechts = j+1;
250                     getauscht_char = this.tausche(
251                                         links,
252                                         rechts,mUnsortiert) ;
253                     mUnsortiert = getauscht_char;
254                 } // ende i
255             }
256         }
257         //Setzt die im Ergebnis die sortierte Zeichenkette
258         this.sortiert =  String.valueOf(getauscht_char);
259
260         //Testausgabe der sortierten Zeichenkette auf der
261         //Konsole
262         System.out.println(this.sortiert);
263     }
```

Eingabehilfe:

```
public void bubblesort(String pUnsortiert){
    //Deklaration eines noch leeren
    // Arrays von chars
    char[] getauscht_char = new char[0];
    //Umwandeln der Zeichenkette in ein
    // Array von Zeichen
    char[] mUnsortiert =
            pUnsortiert.toCharArray();
       //Durchlaufe die Liste
    for (int i = mUnsortiert.length; i>1;i-- ) {
        //Ermittle zwei Elemente benachbarte
        // Elemente der Liste
        for (int j=0; j<i-1; j++){
            //Prüfe ob das linke Element größer
            // ist als das rechte Element
            if (mUnsortiert[j] > mUnsortiert[j+1]){
```

*Algorithmus um eine beliebige Zeichenkette zu sortieren.*

Es gibt eine ganze Menge an unterschiedlichen Sortieralgorithmen. Die Algorithmen unterscheiden sich in Ihrer Effizienz und Kapazität. Wir nutzen im vorliegenden Beispiel den → Bubblesort-Algorithmus, um eine beliebige Zeichenkette zu sortieren.

Der Algorithmus vergleicht benachbarte Zeichen und tauscht diese ggf. aus. Mit dieser Vorgehensweise werden große Werte auf die rechte Seite verlagert, kleine Werte werden dagegen auf die linke Seite verlagert.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare hinzu.
Für die Methode:
`bubblesort(String pZeichenkette): void`

```
public String bubblesort(String pZeichenkette){
      //hier fehlt Quellcode
}
```

Deklaration eines noch leeren Arrays von chars

```
char[] getauscht_char = new char[0];
```

Umwandeln der Zeichenkette in ein Array von Zeichen:

```
char[] mUnsortiert =
        pUnsortiert.toCharArray();
```

Durchlaufe die Liste zeichenweise von hinten:

```
for (int i = mUnsortiert.length; i>1;i-- ) {
      //hier fehlt Quellcode für die erste Schleife
}
```

Für jeden Durchlauf der Schleifen soll folgendes passieren.

Ermittele i und j. Nutze i und j und ermittle damit zwei benachbarte Elemente (an der Stelle i und j) der Liste.

```
for (int j=0; j<i-1; j++){
  //hier fehlt Quellcode für die zweite Schleife
}
```

```
                //Beispiel für die Kapselung
                int links = j;
                int rechts = j+1;
                getauscht_char = this.tausche(
                                links,
                                rechts,
                                mUnsortiert) ;
                mUnsortiert = getauscht_char;
            } // ende i
        }
    }
    //Setzt die im Ergebnis
       //die sortierte Zeichenkette
    this.sortiert =  String.valueOf(getauscht_char);
    //Testausgabe der sortierten
       //Zeichenkette auf der
    //Konsole
    System.out.println(this.sortiert);
}
```

Hinweis:
Auch das Alphabet besteht aus kleinen und großen Werten:
A < B < C < D ...usw.

Prüfe Sie innerhalb der zweiten Schleife ob das linke Element größer ist als das rechte Element. Fürden Fall dass, das zutrifft, tausche die Elemente aus.

```
if (mUnsortiert[j] > mUnsortiert[j+1]){
                //Beispiel für die Kapselung
                int links = j;
                int rechts = j+1;
                getauscht_char = this.tausche(
                                links,
                                rechts,
                                mUnsortiert) ;
                mUnsortiert = getauscht_char;
            }
```

Erst wenn die Zeichenkette ganz durchlaufen ist, soll das Ergebnis in den String → sortiert übernommen werden. Schließen Sie beide

Setzt die im Ergebnis die sortierte Zeichenkette:

```
this.sortiert =  String.valueOf(getauscht_char);
```

Testausgabe der sortierten Zeichenkette auf der Konsole.

```
System.out.println(this.sortiert);
```

Die genutzte Hilfsmethode

```
→ tausche(char pLinks, char pRechts, char[] in)
```

existiert noch nicht, deshalb wird sie rot angezeigt. Wir werden diese Methode gleich im Anschluss implementieren.

```
265     //Hilfsmethode: Tausche Zeichen
266 @   private char[] tausche(int links,
267                           int rechts,
268                           char[] pUnsortiert){
269         char mTemp;
270         mTemp = pUnsortiert[links];
271         pUnsortiert[links]=pUnsortiert[rechts];
272         pUnsortiert[rechts]= mTemp;
273
274         return pUnsortiert;
275     }
```

Eingabehilfe:

```
private char[] tausche(int links,
                       int rechts,
                       char[] pUnsortiert){
    char mTemp;
    mTemp = pUnsortiert[links];
    pUnsortiert[links]=pUnsortiert[rechts];
    pUnsortiert[rechts]= mTemp;
    return pUnsortiert;
}
```

*Der Tauschalgorithmus.*

Der Algorithmus tauscht zwei nebeneinander liegende Zeichen in einer Zeichenkette aus. Auf diese Weise wird ein Platzwechsel vollzogen.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare hinzu.

Für die Methode:

```
tausche(String pZeichenkette): void
  public String tausche(char links, char rechts){
       //hier fehlt Quellcode
  }
```

Für den Inhalt der Methode.

Der Platzhalter → mTemp wird deklariert.

```
char mTemp;
```

Der Platzhalter → mTemp erhält den Wert von → links.

| | |
|---|---|
| | `mTemp = pUnsortiert[links];`<br><br>Der Platzhalter → links erhält den Wert von → rechts.<br>`pUnsortiert[links]=pUnsortiert[rechts];`<br><br>Der Platzhalter rechts erhält den Wert von → mTemp.<br>`pUnsortiert[rechts]= mTemp;`<br><br>Rückgabe des neuen Arrays mit den getauschten Werten.<br>`return pUnsortiert;` |

## 2.4.5 Algorithmus: ersetzen

```
164     /*Die Methode soll das Wort durchlaufen
165     und Umlaute ermitteln und ersetzen
166     ü --> ue
167     ä --> ae
168     ö --> oe
169     ß --> ss*/
170
171     public void umlaute_ersetzen(String pZeichenkette){
172
173         //Eingabe übermitteln
174         String mZeichenkette = pZeichenkette;
175
176         //Noch Leeres Ergebnis erzeugen
177         this.umlautlos = new String();
178
179         //Eingabe durchlaufen
180         for (int i =0 ;i<=mZeichenkette.length()-1 ; i++) {
181
182             //Zeichen (Umlaut) ermittelna
183             char zeichen = mZeichenkette.charAt(i)  ;
```

Für die Methode:
umlaute_ersetzen(String pZeichenkette): void

```
public void umlaute_ersetzen(String pZeichenkette)
    {
    //hier fehlt Quellcode
}
```

```
185             //Für 'ü'
186             if (zeichen == 'ü') {
187                 String mErsatz = "ue";
188                 this.umlautlos = this.umlautlos.concat(
189                         String.valueOf(mErsatz));
190                 //Für 'ö'
191             }else if(zeichen == 'ö'){
192                 String mErsatz = "oe";
193                 this.umlautlos = this.umlautlos.concat(
194                         String.valueOf(mErsatz));
195                 //Für 'ä'
196             }else if(zeichen == 'ä'){
197                 String mErsatz = "ae";
198                 this.umlautlos = this.umlautlos.concat(
199                         String.valueOf(mErsatz));
200                 //Für 'ß'
201             }else if(zeichen == 'ß'){
202                 String mErsatz = "ss";
203                 this.umlautlos = this.umlautlos.concat(
204                         String.valueOf(mErsatz));
205                 //Sonst
206             }else{
207                 this.umlautlos = this.umlautlos.concat(
208                         String.valueOf(zeichen));
209             }
210         }
211     }
```

Erklärung:
Für den Fall, dass das aktuelle Zeichen mit einem der Umlaute übereinstimmt, wird ein Ersatzzei-

*Algorithmus für Zeichen (Umlaute) suchen und ersetzen.*

Der Algorithmus identifiziert die Umlaute in der Eingabe (Satz, pZeichenkette) und setzt im Ergebnis die umlautlose Zeichenkette.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare hinzu.

Eingabe übermitteln:

```
String mZeichenkette = pZeichenkette;
```

Ein am Anfang noch leeres Ergebnis erzeugen:

```
this.umlautlos = new String();
```

Die Eingabe zeichenweise durchlaufen:

```
for (int i =0 ;i<=mZeichenkette.length()-1 ; i++) {
        //hier fehlt Quellcode für die Schleife
}
```

In jedem Schleifendurchlauf soll folgendes passieren:

Das aktuelle Zeichen (Umlaut) wird ermittelt:

```
char zeichen = mZeichenkette.charAt(i);
```

Dann vergleichen wir das ermittelte aktuelle Zeichen mittelt der Kontrollstruktur ELSE IF.

```
//Für 'ü'
if (zeichen == 'ü') {
    String mErsatz = "ue";
    this.umlautlos = this.umlautlos.concat(
                     String.valueOf(mErsatz));
    //Für 'ö'
}else if(zeichen == 'ö'){
    String mErsatz = "oe";
    this.umlautlos = this.umlautlos.concat(
                     String.valueOf(mErsatz));
    //Für 'ä'
}else if(zeichen == 'ä'){
    String mErsatz = "ae";
    this.umlautlos = this.umlautlos.concat(
                     String.valueOf(mErsatz));
    //Für 'ß'
}else if(zeichen == 'ß'){
```

chen definiert und an die neue umlautlose Zeichenkette angehängt. Anderenfalls wird das bestehende aktuelle Zeichen an die umlautlose Zeichenkette angehängt.

```
    String mErsatz = "ss";
    this.umlautlos = this.umlautlos.concat(
                     String.valueOf(mErsatz));
    //Sonst
}else{
    this.umlautlos = this.umlautlos.concat(
                     String.valueOf(zeichen));
}
```

Fertig!!

## 2.4.6 Algorithmus: entfernen

```
277     /** Die Methode soll einen String säubern.
278      * In der eingegebene Zeichenkette (Eingabewert1)
279      * werden Zeichen (Eingabewert2) entfernt*/
280     public void saubermachen(){
281         String mEingabe = this.suchwort;
282
283
284         //Ermittle Anzahl an Zeichen
285         int mAnzahl = this.suchwort.length();
286         System.out.println("Anzahl: "+mAnzahl);
287
288         //Entferne Leerzeichen am Anfang und am Ende
289         String mZeichenkette
290                 = this.text.trim();
291
292         //Erzeuge einen neuen noch leeren String
293         this.sauber
294                 = new String();

295
296         //Initialisiere den Zähler i mit 0
297         int i = 0;
298
299         //Durchlaufe die Zeichenkette zeichenweise
300         while ((i + mAnzahl) <= mZeichenkette.length()){
301
302             //Ermittle einen Teilstring von i bis i+mAnzahl
303             String mZeichen
304                     = mZeichenkette.substring(i, i + mAnzahl);
305             System.out.println("Zeichen: "+mAnzahl);
306
307             //Für den Fall, dass die Zeichenkette gefunden wurde
308             if (mZeichen.equals(mEingabe)) {
309                 //wird Sie mit "Nichts" überschrieben
310                 mZeichen = "";
311                 i = i + mAnzahl;
312             }else{
313                 i++;
314             }
315
316             //Das Zeichen wird an das Ergebnis angehängt
317             this.sauber= this.sauber.concat(mZeichen);
318         }
319     }
```

Eingabehilfe:

```
public void saubermachen(){
    String mEingabe = this.suchwort;
    //Ermittle Anzahl an Zeichen
    int mAnzahl = this.suchwort.length();

    //Entferne Leerzeichen am Anfang und am Ende
    String mZeichenkette
            = this.text.trim();
    //Erzeuge einen neuen noch leeren String
    this.sauber
            = new String();
    //Initialisiere den Zähler i mit 0
    int i = 0;
    //Durchlaufe die Zeichenkette zeichenweise
    while ((i + mAnzahl) <= mZeichenkette.length()){
        //Ermittle einen Teilstring
```

*Algorithmus um ein Wort oder Zeichen zu entfernen.*

Wir nennen die Methode → saubermachen.

Der Algorithmus identifiziert ein oder mehrere Zeichen in einer Zeichenkette und liefert ein Ergebnis ohne diese Zeichen bzw. Zeichenfolge.

Übernehmen Sie die Quellcodebestandteile schrittweise und fügen Sie die Kommentare hinzu.

Für die Methode:

```
saubermachen(String pZeichenkette,
             String pZeichen): void
```

```
public void suabermachen(
             String pZeichenkette,
             String pZeichen){
      //hier fehlt Quellcode
}
```

Für den Inhalt der Methode.

Ermittle Eingabe:

```
String mEingabe = this.suchwort;
```

Ermittle Anzahl an Zeichen.

```
int mAnzahl = pZeichen.length();
```

Entferne Leerzeichen am Anfang und am Ende.

```
String mZeichenkette = this.text.trim();
```

Erzeuge einen neuen noch leeren String.

```
this.sauber = new String();
```

Initialisiere den Zähler i mit 0.

```
int i = 0;
```

Durchlaufe die Zeichenkette zeichenweise.

```
while ((i+ mAnzahl)< mZeichenkette.length()){
      //hier fehlt Quellcode für die Schleife
   }
```

Ermittle einen Teilstring von i bis (i+mAnzahl).

```
String mZeichen
          = mZeichenkette.substring(i, i + mAnzahl);
```

Für den Fall dass, die Eingabe mit dem ermittelten Teilstring übereinstimmt, ersetze die Eingabe durch

```
            //von i bis i+mAnzahl
            String mZeichen
              = mZeichenkette.substring(i, i + mAnzahl);

        //Für den Fall, dass die Zeichenkette
            //gefunden wurde
        if (mZeichen.equals(mEingabe)) {
            //wird Sie mit "Nichts" überschrieben
            mZeichen = "";
            i = i + mAnzahl;
        }else{
            i++;
        }
        //Das Zeichen wird an das Ergebnis
            //angehängt
        this.sauber= this.sauber.concat(mZeichen);
    }
}
```

„Nichts" und der Zähler wird um (i + mAnzahl) erhöht.

```
if (mZeichen.equals(pZeichen)) {
                mZeichen = "";
                i = i + mAnzahl;
            }
```

Ansonsten wird die Zählervariable um 1.

```
else{
                i++;
}
```

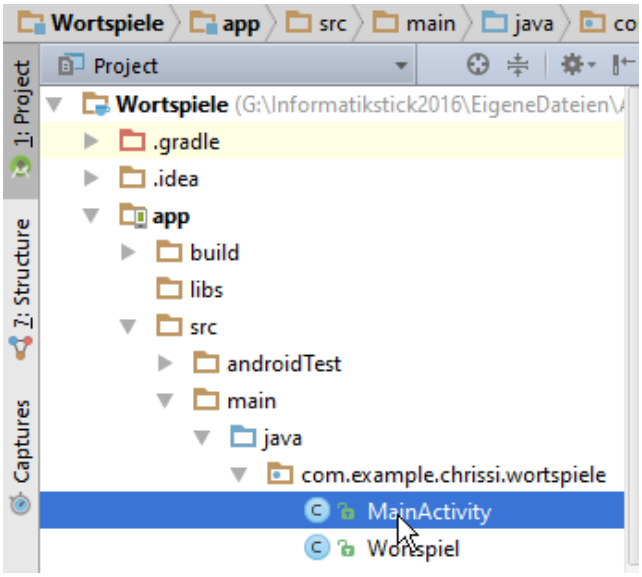
Im Anschluss an die Prüfung wird mZeichen an den neuen sauberen String angehängt.

```
this.sauber= this.sauber.concat(mZeichen);
```

Fertig !!

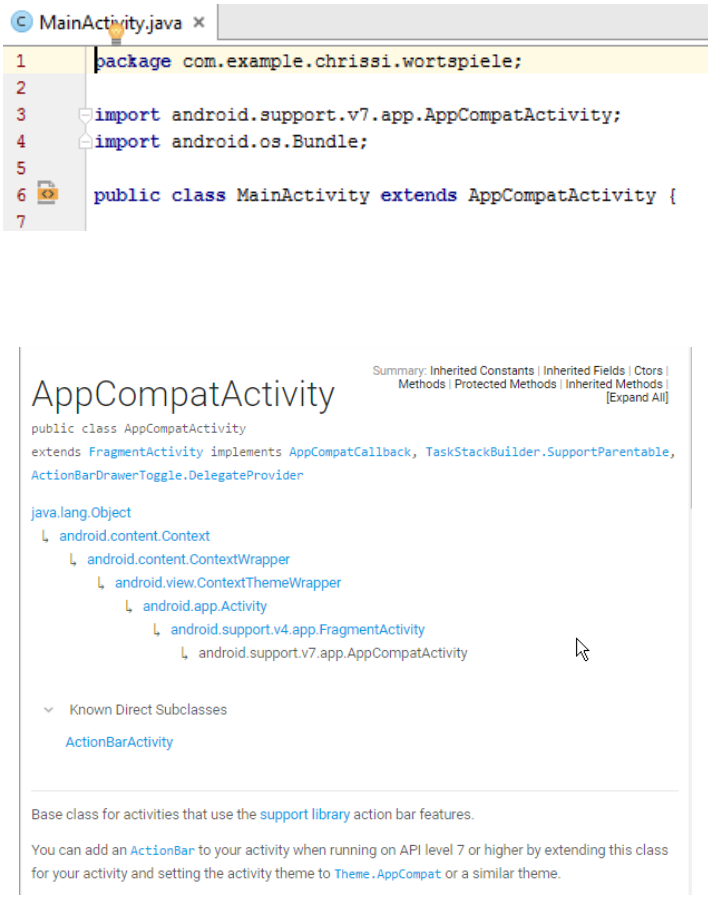
Wir widmen uns nun im nächsten Kapitel der Ereignissteuerung.

## 2.5 Controller: Implementierung der Ereignissteuerung




*Öffnen Sie die Klasse MainActivity.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.




*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 MainActivity, 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 MainActivity-Klasse u.a. die Funktionalität der Main-Methode einer konventionellen Java-Anwendung.

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

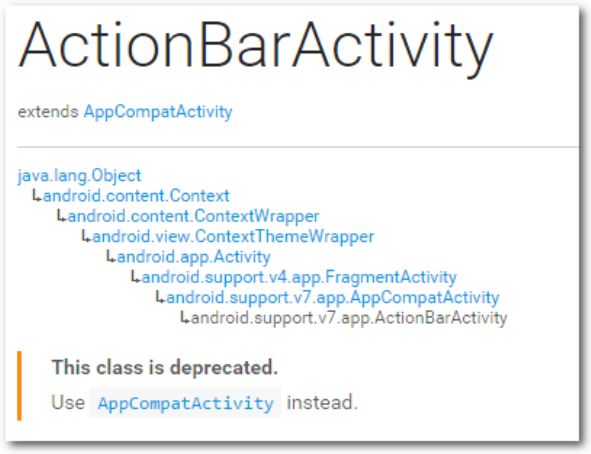
Die MainActivity erbt zwischenzeitlich standardmäßig von der Klasse AppCompatActivity:

```
MainActivity extends AppCompatActivity
```

Bei den meisten älteren Projekten erbt die

Hinweis zu älteren Projekten:

→ ActionBarActivity*




MainActivity noch von der ActionBarActivity

```
MainActivity extends ActionBarActivity
```

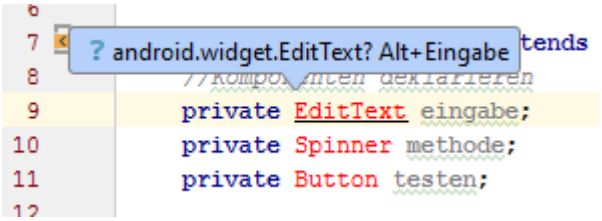
Die Verwendung der Klasse ActionBarActivity ist hinfällig (→ depreceated).

```
9  public class MainActivity extends AppCompatActivity {
10     //Komponenten deklarieren
11     private EditText eingabe;
12     private Spinner methode;
13     private Button testen;
```

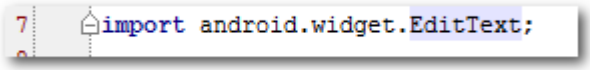
*Komponenten deklarieren.*

Deklarieren Sie im Anschluss an die Klassendeklaration die Komponenten der Benutzeroberfläche.

```
10     //Komponenten deklarieren
11     private EditText eingabe;
12     private Spinner methode;
13     private Button testen;
```




Klicken Sie auf die rot gekennzeichneten Klassennamen für die Komponente und folgen Sie der Empfehlung mit der Tastenkombination ALT + Eingabe (Enter) die Klasse zu importieren:




```
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         setContentView(R.layout.activity_main);
17     }
```

*Die onCreate-Methode.*

In der onCreate-Methode sollte das beim Aufruf des Activity-Objektes benötigte Layout (XML-Datei) übermitteln und in einem Objektbaum entfalten.

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.

```
15      @Override
16      protected void onCreate(Bundle savedInstanceState) {
17          super.onCreate(savedInstanceState);
18          setContentView(R.layout.activity_main);
19
20          //Initialisierung der Komponenten
21          eingabe = (EditText) findViewById(R.id.etText);
22          methode = (Spinner) findViewById(R.id.spMethode);
23          testen = (Button)findViewById(R.id.btTesten);
```

*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:

```
21          eingabe = (EditText) findViewById(R.id.etText);
```

- eingabe:
  Ist ein Klassenattribut der Activity-Klasse vom Typ Edittext (siehe Deklaration).
- (EditText):
  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.etText
  R liefert zum String etErgebnis den entsprechenden int-Wert zurück. Den entsprechenden Schlüsselwert.

```
27          //Listener für das Texteingabefeld
28          eingabe.addTextChangedListener(
29                  new TextWatcher() {
30
31                  });
32
```

*Listener in der onCreate-Methode.*

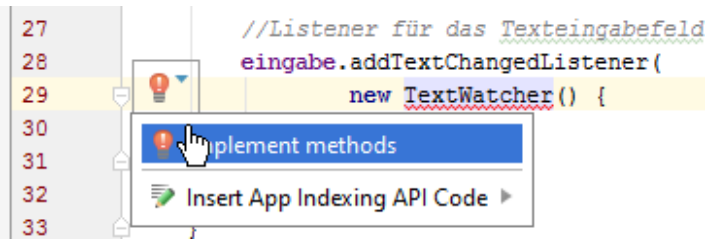
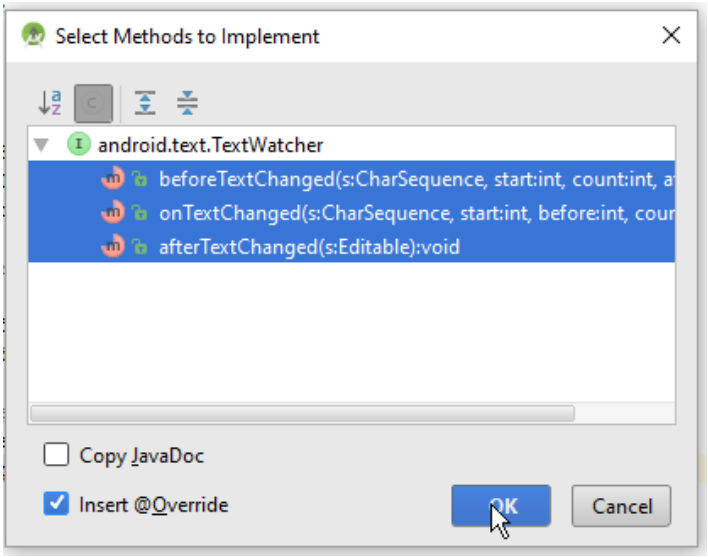
Ein Listener ist wie ein Fühler der Veränderungen auf der Benutzeroberfläche registriert und in Form eines Impulses an das System weiterreicht.

Wir fügen dem editierbaren Objekt → eingabe mit dem Methodenaufruf

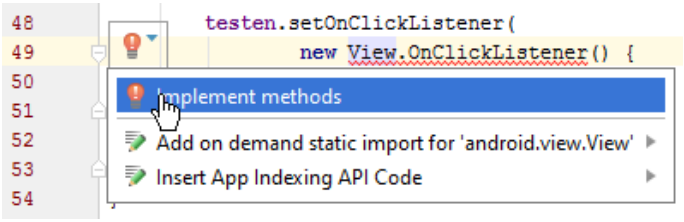
eingabe.addTextChangedListener(TextWatcher watcher)

```
new TextWatcher(){
```

den Listener hinzu. Als Parameter wird ein neu erzeugtes TextWatcher-Objekt übergeben.

| | |
|---|---|
| | Implementieren Sie den Methodenaufruf wie nebenstehend angezeigt. |
|  | *TextWatcher-Methoden-Deklaration einfügen.*<br><br>Klicken Sie auf den Klassennamen TextWatcher. Mit einem Klick auf die kleine rote Glühbirne am linken Rand und der Tastenkombination ALT+ Eingabe (Enter) werden die fehlenden Methoden implementiert.<br><br>  |
| public void **onTextChanged**(CharSequence s, int start, int before, int count)<br><br>Die Methode wird aufgerufen, um uns Veränderungen innerhalb der eingegebenen Zeichenkette „s“ anzuzeigen.<br><br>public void **beforeTextChanged**(CharSequence s, int start, int count, int after)<br><br>Die Methode wird aufgerufen, um uns vorab über die Veränderungen innerhalb der eingegebenen Zeichenkette „s“ zu informieren.<br><br>public void **afterTextChanged**(Editable s)<br><br>Die Methode wird aufgerufen, um über die Veränderungen innerhalb der eingegebenen Zeichenkette „s“ zu informieren, nachdem sie bereits vorgenommen wurde. | *TextWatcher-Methoden implementieren.*<br><br>Dieses Objekt bringt von sich aus drei Verhaltensweisen (Methoden) mit sich. Diese Methoden sind nun deklariert. Die Implementierung müssen wir bei Bedarf selbst vornehmen.<br><br>@Override<br>Der Vermerk signalisiert uns, dass es sich um eine vererbte Methode handelt. Wir werden diese Methoden bei Bedarf überschreiben.<br><br>Wir möchten den Fühler für die Zeichenkette aus der EditText-Komponente → eingabe freigeben, sodass wir jede Eingabeänderung nachträglich angezeigt bekommen.<br><br>Dafür implementieren wir die Methode wie folgt: |

```
public void afterTextChanged(Editable s){
        eingabe.setEnabled(s.length() >= 0);
}
```



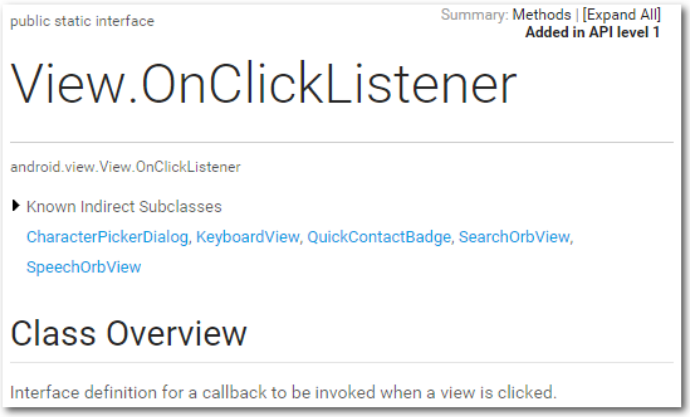

*Listener für die Button Komponente.*

Auch der Button braucht einen Fühler der Aktivitäten registriert.

Wählen Sie im Dropdown-Menü die Option

```
OnClickListener{...}
```

Auszug der API:




View.OnClickListener

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

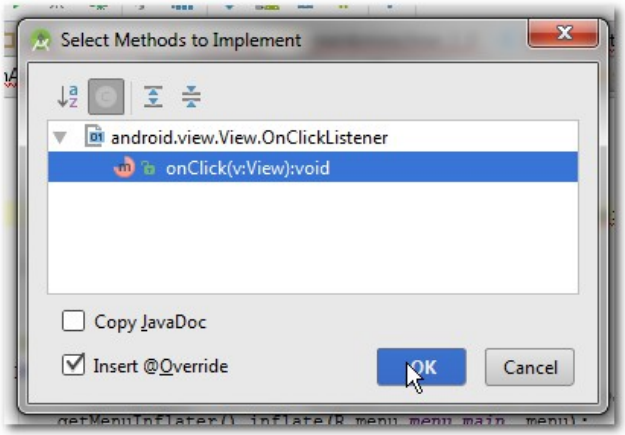
Ergebnis:

```
//Listener für die Button-Komponente
testen.setOnClickListener(
        new View.OnClickListener() {

            @Override
            public void onClick(View v) {

            }
        });
```

*View.OnClickListener-Methode deklarieren.*

Gehen Sie auf die gleiche Weise vor wie zuvor für den TextWatcher.

Alternativ klicken Sie auf den Klassennamen View. Mit einem Klick auf die kleine rote Glühbirne am linken Rand und der Tastenkombination ALT+ Eingabe (Enter) werden die fehlenden Methoden implementiert.




Implementieren Sie den Listener wie nebenstehend angezeigt.

```
59      //FÜR DEN ADAPTER DER SPINNER-KOMPONENTE
60
61      //Deklaration und Initialisierung des Adapters
62      ArrayAdapter<CharSequence>
63              spMethode_adapter=
64              ArrayAdapter.createFromResource(this,
65                      R.array.methode_array,
66                      android.R.layout.simple_spinner_item);
67
68      //Spezifiziere das Layout für das Drop-Down-Menü
69      spMethode_adapter.setDropDownViewResource(
70              android.R.layout
71                      .simple_spinner_dropdown_item);
72
73      //Das Adapter-Objekt wird gesetzt
74      methode.setAdapter(spMethode_adapter);
```

Zeile 62:

```
ArrayAdapter<CharSequence>
      spMethode_adapter
              = ArrayAdapter
       .createFromResource(
       this,
      R.array.methode_array,
      android.R
              .layout.simple_spinner_item);
```

Zeile 69:

```
spMethode_adapter.setDropDownViewResource(
        android.R.layout
                .simple_spinner_dropdown_item);
```

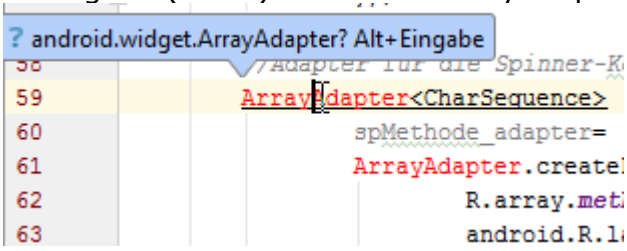
Zeile 76:

```
methode.setAdapter(spMethode_adapter);
```

*Adapter für die Spinner-Komponenten.*

Unterhalb der Listener-Aufrufe erzeugen wir die Array Adapter.

Importieren Sie der Tastenkombination ALT+Eingabe (Enter) die Klasse ArrayAdapter.




Für → spMethode:
Implementieren Sie den Adapter für den Spinner → spMethode, wie nebenstehend angezeigt.

```
78          //Listener für die Spinner-Komponente
79          methode.setOnItemSelectedListener(
80                  new AdapterView.OnItemSelectedListener() {
81                      public void onItemSelected(
82                              AdapterView<?> parent, View view,
83                              int position, long id) {
84                          showToast("Methodenauswahl: position="
85                                  + position + " id=" + id);
86                      }
87
88                      public void onNothingSelected(
89                              AdapterView<?> parent) {
90                          showToast("Methodenauswahl: keine");
91
92                      }
93                  });
```

Zeile 79:

```
methode.setOnItemSelectedListener(
        new AdapterView.OnItemSelectedListener() {
            public void onItemSelected(
                    AdapterView<?> parent, View view,
                    int position, long id) {
                            //hier fehlt Quellcode
                    }
            public void onNothingSelected(
                    AdapterView<?> parent) {
                            //hier fehlt Quellcode
                    }
        });
```

Hilfsmethode:

```
97      //Hilfsmethode
98      private void showToast(CharSequence msg) {
99          Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
100     }
```

*Listener für die Spinner-Komponenten.*

Toasts
Erfüllen in der Android Anwendungen den Zweck klassischer PopUps. Man verwendet Toasts allerdings hauptsächlich zur Ausgabe von kleinen textuellen Hinweisen für den Benutzer.

Im nebenstehenden Beispiel werden sie zur Anzeige einer Testausgabe verwendet.

Zeile 84:
Anzeige der Position und Id des Spinners für den Fall, dass der Benutzer ein Objekt aus dem spinner (Drop-Down-Menü) → von gewählt hat.

```
showToast("Methodenauswahl: position="
 + position + " id=" + id);
```

Zeile 90:
Anzeige der Meldung „unselected" für den Fall, dass der Benutzer kein Objekt aus dem spinner (Drop-Down-Menü) → von gewählt hat.

```
showToast("Methodenauswahl: keine");
```

Wir implementieren dazu die erste Hilfsmethode am unteren Rand der MainActivity-Klasse.

```
private showToast(CarSequence msg){
        //hier fehlt Quellcode
}
```

Implementieren Sie die benötigte Hilfsmethode, wie nebenstehend angezeigt.

```
@Override
public void onClick(View v) {
    if(v == testen ){
        //Eingabe

        //Verarbeitung

        //Ausgabe

    }else{
        finish();
    }
}
```

*Die Implementierung der Methode:*

```
public void onClick(View v){
  /*hier fehlt Quellcode*/
}
```

Wir wenden uns erneut dem Listener unserer Button-Komponente zu und implementieren die onClick-Methode.

Wir wenden bei der Umsetzung drei weitere Prinzipien an. Unser Fokus: Die Prinzipien „Zerlegung", Kapselung und „Wiederverwendung".

Wir Zerlegen also im ersten Schritt unser logisches Problemchen:

| | |
|---|---|
| | IF-Fall<br>Für den Fall, dass die Schaltfläche → testen angeklickt wurde soll, wie folgt vorgegangen werden:<br><br>//**E**ingabe<br>1. Lesen der → eingabe und Übergabe des Wertes an ein temporäres Attribut → mEingabe<br>2. Lesen der Methodenauswahl (testen) und Übergabe des Wertes → methode an ein temporäres Attribut → mMethode<br>3. Übermitteln des Wertes → mEingabe an die Objekteigenschaft → text der Fachklasse<br>4. Übermitteln des Wertes → mMethode an die Objekteigenschaft → methode der Fachklasse<br><br>//**V**erarbeitung<br>5. Ermittlung des → ergebnisses am Objekt der Fachklasse. Dazu wird zuvor mittels einer Fallunterscheidung der gewünschte Methodenaufruf ausgeführt.<br><br>//**A**usgabe<br>6. Übermittlung des jeweiligen → ergebnisses aus dem Objekt der Fachklasse und Anzeige des Wertes im Dialogfenster der Benutzeroberfläche.<br><br>ELSE-Fall:<br>Ansonsten soll die Aktivität geschlossen werden. |
| 20<br>21 //Assoziation zur Fachklasse Wortspiel<br>22 private Wortspiel einWortspiel = new Wortspiel(); | *Assoziation. MainActivity → Wortspiel*<br><br>Die Verarbeitung der Eigenschaftswerte soll am Objekt der Fachklasse (Modell) erfolgen. Damit brauchen wir in unserer Activity eine Wortspiel-Objekt das wir nutzen können.<br><br>Deklarieren und initialisieren Sie dieses Objekt unterhalb der bereits deklarierten Komponenten in der Klasse → MainActivity.java. |
| Lese-Methode mit Rückgabewert für die Eingabe (EditView) :<br><br>`private String leseEingabe(){` | *EVA-Prinzip. Die Eingabe.*<br><br>Wir implementieren dazu die Lese-Methoden |

```
    String mEingabe=
            eingabe.getText().toString();
    return mEingabe;
}
```

Erläuterung: *Von Innen nach Außen*

`eingabe.getText().toString()`
Der Wert für die → eingabe wird ermittelt und in einen String umgewandelt.

`String mEingabe =`

Der Wert des lokalen Attributs wird zurückgegeben.
`return mEingabe;`

Lese-Methode mit Rückgabewert für den Spinnerwert Ausgangswährung (von):

```
private String leseMethode(){
    String mMethode =
       methode.getSelectedItem().toString().trim();

    return mMethode;
}
```

Erläuterung: *Von Innen nach Außen*

`methode.getSelectedItem().toString().trim();`
Der Wert für den ausgewählten Wert → methode wird ermittelt, in einen String umgewandelt. Mit `trim()` werden vor- bzw. nachgelagerte Leerzeichen entfernt.

`String mMethode =`
Der Wert wird einem lokalen Attribut → `mMethode` zugewiesen.

`return mMethode;`
Der Wert des lokalen Attributs wird zurückgegeben.

am unteren Rand der MainActivity-Klasse. Erzeugen Sie einen Kommentar

```
108
109        //Hilfsmethoden
```

damit Sie die Methoden künftig schnell finden.

Implementieren Sie die Lese-Methode für die eingabe, wie nebenstehend angezeigt.

Erzeugen Sie dann die Lese-Methode für den Spinnerwert (→ methode) nach dem gleichen Muster.

Rufen Sie dann an entsprechender Stelle der `onClick(View v)`-Methode, die Lese-Methoden auf und fangen Sie den Rückhgabewert in den likalen Attribute → mEingabe, mMethode auf. Anschließend werden die Eigenschaftswerte an das Objekt der Fachklasse übermittelt.

Methodenaufrufe in der `onClick(View v)`-Methode

```
@Override
public void onClick(View v) {
    if (v == testen) {
        //Eingabe lesen
        String mEingabe = leseEingabe();
        String mMethode = leseMethode();

        //Eingaben setzen
        einWortspiel.setText(mEingabe);
        einWortspiel.setMethode(mMethode);
```

```
@Override
public void onClick(View v) {
    if (v == testen) {
        //Eingabe lesen
        String mEingabe = leseEingabe();
        String mMethode = leseMethode();

        //Eingaben setzen
        einWortspiel.setText(mEingabe);
        einWortspiel.setMethode(mMethode);

        //Verarbeitung
        String mAusgabe
                = methode_aufrufen(mMethode, mEingabe);
```

*EVA-Prinzip. Die Verarbeitung.*

Im Rahmen der Verarbeitung müssen wir ermitteln welche Methode im Drop-Down-Menü (Spinner) ausgewählt wurde. Erst dann können wir den entsprechenden Methodenaufruf veranlassen.

Implementieren Sie den Methodenaufruf an entsprechender Stelle der `onClick(View v)`-Methode.

Diese Hilfsmethode implementieren wir nun im

nächsten Schritt.

```
149     //Methodenaufruf auswählen
150     private String methode_aufrufen(String pMethode,
151                                      String pEingabe) {
152
153         //ein noch leerer String
154         String mAusgabe = new String();
155
156         //Fall: umdrehen
157         if (pMethode.equals("Satz oder Wort umdrehen")) {
158             einWortspiel.umdrehen();
159             mAusgabe = einWortspiel.getRueckwaerts();
160
161             //Fall: suchen
162         } else if (pMethode.equals("Wort suchen")) {
163             Log.d(LOG_TAG,
164                     "Eingabe erfolgt.");
165
166             //Deklaration und Initialierung
167             //des Bearbeitungsfensters
168             AlertDialog dialog
169                     = createInputDialog(einWortspiel,pMethode);
170
171             //Anzeige des Bearbeitungsfensters
172             dialog.show();
```

```
173
174             //Fall: umlaute ersetzen
175         } else if (pMethode.equals("Umlaute ersetzen")) {
176             einWortspiel.umlaute_ersetzen(pEingabe);
177             mAusgabe = "Ergebnis: "+ einWortspiel.getUmlautlos();
178
179             //Fall: sortieren
180         } else if (pMethode.equals("Aufsteigend sortieren")) {
181             einWortspiel.bubblesort(pEingabe);
182             mAusgabe = "Ergebnis: "+ einWortspiel.getSortiert();
183
184             //Fall: säubern
185         } else if (pMethode.equals("Zeichenkette säubern")) {
186             einWortspiel.setText(pEingabe);
187             Log.d(LOG_TAG,
188                     "Eingabe erfolgt.");
189
190             //Deklaration und Initialierung
191             //des Bearbeitungsfensters
192             AlertDialog dialog
193                     = createInputDialog(
194                             einWortspiel,
195                             pMethode);
196
197             //Anzeige des Bearbeitungsfensters
198             dialog.show();
199
200         }
201         return mAusgabe;
202     }
```

Eingabehilfe:

```
private String methode_aufrufen(String pMethode,
                                 String pEingabe) {
    //ein noch leerer String
    String mAusgabe = new String();

    //Fall: umdrehen
    if (pMethode.equals("Satz oder Wort umdrehen")) {
        einWortspiel.umdrehen();
```

*Hilfsmethode für die Auswahl implementieren.*

Implementieren Sie die Hilfsmethode schrittweise und fügen Sie zum besseren Verständnis die Kommentare ein, wie nebenstehend angezeigt.

Wir vergleichen Fall für Fall die durch den Benutzer ausgewählte Methode mit der Operation und führen dann die noch fehlenden Anweisungen aus.

Deklaration der Methode einfügen:

```
private void methode_aufrufen(String pMethode,
                              String pEingabe){
        //hier fehlt Quellcode
}
```

Für den Inhalt der Methode (Implementierung) unterscheiden wir die fünf Fälle. Ergänzen Sie den Quellcode und die Kommentare.

Deklaration eines noch leeren Strings für die Ausgabe:

```
    String mAusgabe = new String();
```

Fall: umdrehen

```
    if(pMethode.equals("Satz oder Wort umdrehen")){
        einWortspiel.setSatz(pEingabe);
        einWortspiel.umdrehen();
        mAusgabe = einWortspiel.getRueckwaerts();
        }
```

Fall: suchen

```
    else if(pMethode.equals("Wort suchen")){
        AlertDialog dialog
          = createInputDialog(einWortspiel,pMethode);
        dialog.show();
    }
```

Fall: umlaute ersetzen

```
else if (pMethode.equals("Umlaute ersetzen")) {
        einWortspiel.umlaute_ersetzen(pEingabe);
        mAusgabe = "Ergebnis: "
                + einWortspiel.getUmlautlos();
    }
```

Fall: sortieren

```
    else if(pMethode
                .equals("Aufsteigend sortieren")){
        einWortspiel.bubblesort(pEingabe);
        mAusgabe = "Ergebnis: "
```

```
        mAusgabe = einWortspiel.getRueckwaerts();

        //Fall: suchen
    } else if (pMethode.equals("Wort suchen")) {

        //Deklaration und Initialierung
        //des Bearbeitungsfensters
        AlertDialog dialog
          = createInputDialog(einWortspiel,pMethode);

        //Anzeige des Bearbeitungsfensters
        dialog.show();

        //Fall: umlaute ersetzen
    } else if (pMethode.equals("Umlaute ersetzen")) {
        einWortspiel.umlaute_ersetzen(pEingabe);
        mAusgabe = "Ergebnis: "
              + einWortspiel.getUmlautlos();
        //Fall: sortieren
    } else if (pMethode.equals(
                      "Aufsteigend sortieren")) {
        einWortspiel.bubblesort(pEingabe);
        mAusgabe = "Ergebnis: "
                      + einWortspiel.getSortiert();
        //Fall: säubern
    } else if (pMethode.equals(
               "Zeichenkette säubern")) {
        einWortspiel.setText(pEingabe);

        //Deklaration und Initialierung
        //des Bearbeitungsfensters
        AlertDialog dialog
       = createInputDialog(einWortspiel,pMethode);

        //Anzeige des Bearbeitungsfensters
        dialog.show();
    }
    return mAusgabe;
}
```

```
                      + einWortspiel.getSortiert();

    }
```

Fall: umlaute ersetzen

```
    else if(pMethode
        .equals("Aufsteigend sortieren")){
        einWortspiel.bubblesort(pEingabe);
        mAusgabe = einWortspiel.getSortiert();

    }
```

Fall: säubern

```
else if(pMethode
        .equals("Zeichenkette säubern")){
        einWortspiel.setText(pEingabe);
        AlertDialog dialog
       = createInputDialog(einWortspiel,pMethode);
        dialog.show();
    }
```

Rückgabe des Ergebisses:

```
return mAusgabe;
```

Hinweis:
Für den Fall, dass wir für die Verarbeitung zusätzliche Eingabewerte benötigen werden wir ein extra Eingabefenster nutzen.

Da wir die dazu benötigte Hilfsmethode

```
→ createInputDialog(Wortspiel pW, String pM)
```

ist bisher weder deklariert noch implementiert haben, wird sie rot angezeigt.

Wir widmen uns also im nächsten Schritt dieser Hilfsmethode.

```
204     //Dialogfenster
205     //Ein Inputdialog bietet die Möglichkeit
206     //zusätzliche Eingaben zu übernehemen
207     public AlertDialog createInputDialog(final Wortspiel pWortspiel,
208                                          final String pMethode) {
209         //Erzeugt eine Dialogfenster-Objekt
210         // und übermittelt das Activity-Objekt
211         AlertDialog.Builder builder
212                 = new AlertDialog.Builder(this);
213
214         //Den LayoutInflater (Befüller) initialisieren
215         LayoutInflater inflater
216                 = getLayoutInflater();
217
218         //Initialisiert (befüllt) die View des
219         // Dialogfensters mit dem XML-Layout
220         View dialogsView
221                 = inflater.inflate(
222                 R.layout.dialog_eingabe, null);
223
224         //Initialisiert die Eingabe-Komponente
225
226         final EditText etEingabe_dialog
227                 = (EditText) dialogsView
228                 .findViewById(R.id.etEingabe);
229
230         final EditText etText_dialog
231                 = (EditText) dialogsView
232                 .findViewById(R.id.etSatz);
233
234         //Wert übernehmen
235         etText_dialog.setText(einWortspiel.getText());
236
237         //Titel im Bearbeitungsfenster anzeigen
238         builder.setView(dialogsView)
239
240                 //Titel im Bearbeitungsfenster anzeigen
241                 .setTitle(R.string.dialog_titel)
242
243         //Ereignissteuerung für die Schaltfläche
244                 // Eintrag ändern im Bearbeitungsfenster
245         .setPositiveButton(R.string.btDialog_positiv,
246                 new DialogInterface.OnClickListener() {
247             @Override
248             public void onClick(
249                     DialogInterface dialog, int id) {
250                 //Lesen der geänderten Daten
251                 String eingabe
252                         = etEingabe_dialog.getText().toString();
253
254                 String text
255                         = etText_dialog.getText().toString();
256
257                 //Prüfung ob Eingaben in den
258                 //Texteingabefeldern fehlen
259                 if ((TextUtils.isEmpty(eingabe))||
260                         (TextUtils.isEmpty(text))) {
261                     Log.d(LOG_TAG,
262                             "Ein Eintrag enthielt keinen Text. " +
263                                     "Daher Abbruch der Änderung.");
264                     return;
265                 }
266
267                 // An dieser Stelle übermitteln wir das Suchwort
268                 einWortspiel.setSuchwort(eingabe);
269                 einWortspiel.setText(text);
270
271                 // An dieser Stelle übermitteln
272                 //wir das Suchwort
273                 if(pMethode.equals("Wort suchen")) {
274                     //Führen die Suche aus
275                     String mMeldung = einWortspiel.suche();
276
277                     //Erzeugen die Ausgabe
278                     showToast(mMeldung);
```

*Erzeugung des Dialogfensters implementieren.*

Für den Fall, dass zusätzliche Eingaben notwendig sind erzeugen wir eine Dialog.

Deklaration der Methode:

```
public AlertDialog createInputDialog(
        final Wortspiel pWortspiel,
        final String pMethode) {
        //hier fehlt Quellcode

}
```

Inhalt der Methode. Implementieren Sie dazu schrittweise den fehlenden Quellcode und ergänzen Sie die Kommentare.

Erzeugt eine Dialogfenster-Objekt und übermittelt das Activity-Objekt.

```
    AlertDialog.Builder builder
            = new AlertDialog.Builder(this);
```

Der Inflater (Befüller) initialisieren

```
    LayoutInflater inflater
            = getLayoutInflater();
```

Initialisiert (befüllt) die View des Dialogfensters mit dem XML-Layout.

```
    View dialogsView
            = inflater.inflate(
            R.layout.dialog_eingabe, null);
```

Initialisiert die Eingabe-Komponente.

```
    final EditText etEingabe_dialog
            = (EditText) dialogsView
            .findViewById(R.id.etEingabe);
    final EditText etText_dialog
            = (EditText) dialogsView
            .findViewById(R.id.etSatz);
```

Wert übernehmen

```
    etText_dialog.setText(einWortspiel.getText());
```

Titel im Bearbeitungsfenster anzeigen

```
    builder.setView(dialogsView)
```

Titel im Bearbeitungsfenster anzeigen

```
            .setTitle(R.string.dialog_titel)
```

Ereignissteuerung für das Ereignis, dass der Benutzer auf die Schaltfläche im Dialogfenster klickt benötigen wir ein Listener-Objekt.

```
    builder.setPositiveButton(R.string.btDialog,
```

```
                //Für den Fall, dass ein Zeichen
                //oder ein Wort entfernt werden soll
            }else if(pMethode.equals("Zeichenkette säubern")){
                //Führe saubermachen aus
                einWortspiel.saubermachen();

                //Ergebnis ermitteln
                String mSauber = einWortspiel.getSauber();

                //Erzeugen die Ausgabe
                showToast("Ergebnis: "+ mSauber);

            }

            //Log-Meldung erzeugen und im Logcat ausgeben
            Log.d(LOG_TAG,
                    "Eingabe erfolgt: "
                            +einWortspiel.getSauber());


            //Schließen des Bearbeitungsfensters
            dialog.dismiss();
        })
```

```
            //Ereignissteuerung für die Schaltfläche
            // Abbrechen im Bearbeitungsfenster
            .setNegativeButton(
                    R.string.btDialog_negativ,
                    new DialogInterface.OnClickListener() {
                        public void onClick(
                                DialogInterface dialog, int id) {

                        //Abbruch des Vorgangs und
                            // Schließen des Bearbeitungsfensters
                            dialog.cancel();
                        }
                    });

    return builder.create();
}
```

Eingabehilfe:

```
public AlertDialog createInputDialog(
        final Wortspiel pWortspiel,
        final String pMethode) {

    AlertDialog.Builder builder
            = new AlertDialog.Builder(this);

    LayoutInflater inflater
            = getLayoutInflater();


    View dialogsView
            = inflater.inflate(
            R.layout.dialog_eingabe, null);

    final EditText etEingabe_dialog
            = (EditText) dialogsView
            .findViewById(R.id.etEingabe);

    final EditText etText_dialog
            = (EditText) dialogsView
            .findViewById(R.id.etSatz);
```

```
            new DialogInterface.OnClickListener() {
                //hier fehlt die Quellcode für die onClick
            });
```

Für die onClick-Methode implementieren Sie den folgenden Quellcode.

```
        @Override
        public void onClick(
                DialogInterface dialog, int id) {

            //Lesen der geänderten Daten
            String eingabe
            = etEingabe_dialog.getText().toString();
            String text
            = etText_dialog.getText().toString();

            //Prüfung ob Eingaben in den
            //Texteingabefeldern fehlen
            if ((TextUtils.isEmpty(eingabe))||
                    (TextUtils.isEmpty(text))) {
                Log.d(
                    LOG_TAG,
                    "Ein Eintrag enthielt keinen Text. "
                    +"Daher Abbruch der Änderung.");
                    return;
                }
                // An dieser Stelle übermitteln
                    //wir das Suchwort
                einWortspiel.setSuchwort(eingabe);
                einWortspiel.setText(text);

                    //Für den Fall, dass die Suche
                    //erfolgen soll
                if(pMethode.equals("Wort suchen")) {

            //Führen die Suche aus
            String mMeldung = einWortspiel.suche();

                    //Erzeugen die Ausgabe
                    showToast(mMeldung);

                    //Für den Fall, dass ein Zeichen
                    //oder ein Wort entfernt werden soll
                }else if(
            pMethode.equals("Zeichenkette säubern")){


                    //Führe saubermachen aus
                    einWortspiel.saubermachen();

            //Ergebnis ermitteln
            String mSauber = einWortspiel.getSauber();
                    //Erzeugen die Ausgabe
                    showToast("Ergebnis: "+ mSauber);
                }


                //Schließen des Bearbeitungsfensters
                dialog.dismiss();
            }
        })
```

```
    etText_dialog.setText(einWortspiel.getText());

    builder.setView(dialogsView)

            .setTitle(R.string.dialog_titel)

    .setPositiveButton(R.string.btDialog_positiv,
            new DialogInterface.OnClickListener() {

        @Override
        public void onClick(
                DialogInterface dialog, int id) {

            String eingabe
            = etEingabe_dialog.getText().toString();
            String text
            = etText_dialog.getText().toString();

            if ((TextUtils.isEmpty(eingabe))||
                    (TextUtils.isEmpty(text))) {
               Log.d(
                  LOG_TAG,
                  "Ein Eintrag enthielt keinen Text. "
                  +"Daher Abbruch der Änderung.");
                   return;
            }

            einWortspiel.setSuchwort(eingabe);
            einWortspiel.setText(text);

            if(pMethode.equals("Wort suchen")) {

               String mMeldung
                      = einWortspiel.suche();

               showToast(mMeldung);

            }else if(
        pMethode.equals("Zeichenkette säubern")){

               einWortspiel.saubermachen();

               String mSauber
               = einWortspiel.getSauber();

               showToast("Ergebnis: "+ mSauber);
            }

            dialog.dismiss();
        }
    })

            .setNegativeButton(
             R.string.btDialog_negativ,
             new DialogInterface.OnClickListener() {

           public void onClick(
               DialogInterface dialog, int id) {
                           dialog.cancel();
                       }
                   });
    return builder.create();
```

Ereignissteuerung für die Schaltfläche Abbrechen im Bearbeitungsfenster

```
            .setNegativeButton(
             R.string.btDialog_negativ,
             new DialogInterface.OnClickListener() {

           public void onClick(
               DialogInterface dialog, int id) {

           //Abbruch des Vorgangs und
           // Schließen des Bearbeitungsfensters
                           dialog.cancel();
                       }
                   });
```

Abschließend der Methodenaufruf für die Erzeugung des Dialogfensters.

```
return builder.create();
```

```
}
```

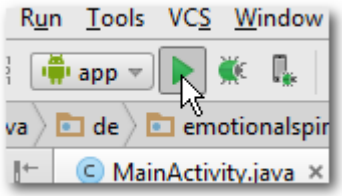
```
@Override
public void onClick(View v) {
    if (v == testen) {
        //Eingabe lesen
        String mEingabe = leseEingabe();
        String mMethode = leseMethode();

        //Eingaben setzen
        einWortspiel.setText(mEingabe);
        einWortspiel.setMethode(mMethode);

        //Verarbeitung
        String mAusgabe
                = methode_aufrufen(mMethode, mEingabe);

        //Ausgabe
        showToast("Ergebnis: " + mAusgabe);

    } else {
        finish();
    }
}
```

*EVA-Prinzip. Die Ausgabe erzeugen.*

Wir ergänzen abschließend noch die Ausgabe des Ergebnisses in einem Meldungsfenster (Toast).

Erweitern Sie dazu die onClick-Methode, wie folgt:

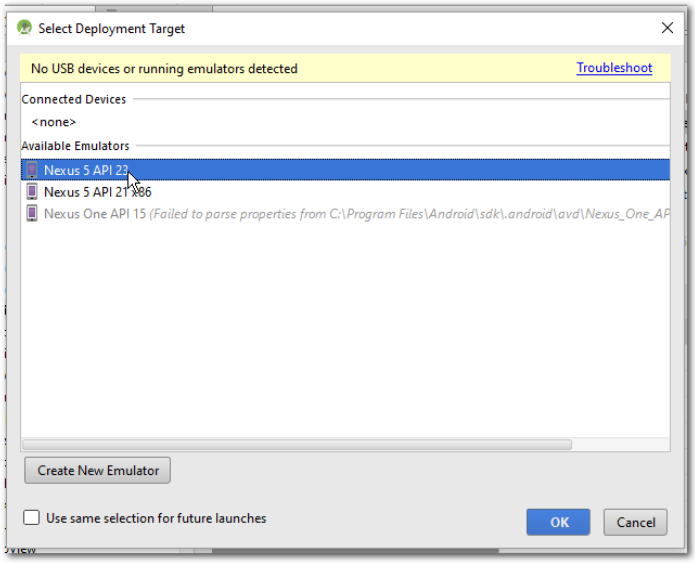
```
showToast("Ergebnis: "+ mAusgabe);
```




*Prototyp testen.*

So nun sollte unsere kleine, WortspieleApp funktionieren.

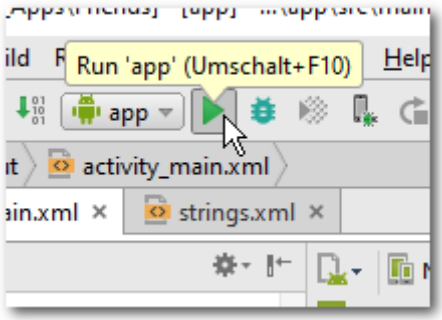
Klicken Sie auf den grünen Pfeil in der Symbol-Leiste oberhalb des Designers.




Alternativ → Create New Emulator:
Für wenig leistungsfähige Rechner empfiehlt sich ein neues Gerät → Nexus One Device mit

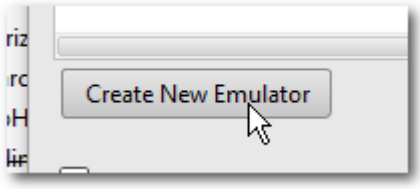
*Testen der View.*

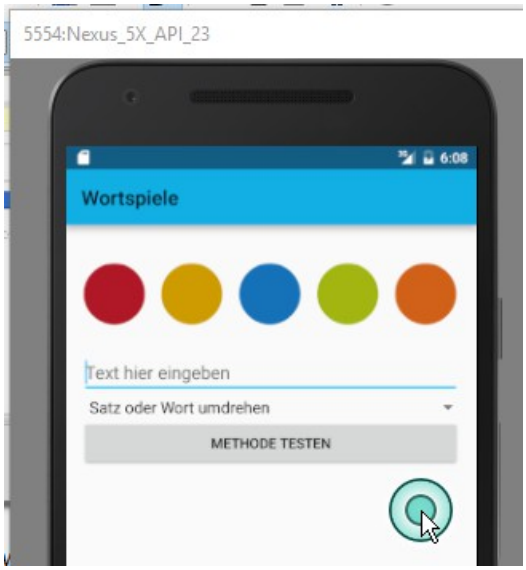
Wir starten nun den Emulator.




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

API 15 (SanwichIceCream) zu erzeugen:




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.

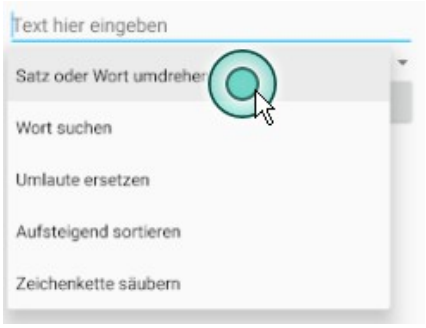
*Der Emulator öffnet sich.*

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-Prozesses von selbst.

Im Ergebnis sollte die Benutzeroberfläche erscheinen.




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